Active Learning Methods for Dynamic Job Shop Scheduling using Genetic Programming under Uncertain Environment

by

Deepak Karunakaran

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Doctor of Philosophy
in Computer Science.

Victoria University of Wellington
2019

Abstract

Scheduling is an important problem in artificial intelligence and operations research. In production processes, it deals with the problem of allocation of resources to different tasks with the goal of optimizing one or more objectives. Job shop scheduling is a classic and very common scheduling problem. In the real world, shop environments dynamically change due to events such as the arrival of new jobs and machine breakdown. In such manufacturing environments, uncertainty in shop parameters is typical. It is of vital importance to develop methods for effective scheduling in such practical settings.

Scheduling using heuristics like dispatching rules is very popular and suitable for such environments due to their low computational cost and ease of implementation. For a dynamic manufacturing environment with varying shop scenarios, using a universal dispatching rule is not very effective. But manual development of effective dispatching rules is difficult, time consuming and requires expertise. Genetic programming is an evolutionary approach which is suitable for automatically designing effective dispatching rules. Since the genetic programming approach searches in the space of heuristics (dispatching rules) instead of building up a schedule, it is considered a hyper-heuristic approach.

Genetic programming like many other evolutionary approaches is computationally expensive. Therefore, it is of vital importance to present the genetic programming based hyper-heuristic (GPHH) system with scheduling problem instances which capture the complex shop scenarios capturing the difficulty in scheduling. Active learning is a related concept from machine learning which concerns with effective sampling of those training instances to promote the accuracy of the learned model.

The overall goal of this thesis is to develop effective and efficient genetic programming based hyper-heuristic approaches using active learning techniques for dynamic job shop scheduling problems with one or more objectives.

This thesis develops new representations for genetic programming enabling it to incorporate the uncertainty information about processing times of the jobs. Furthermore, a cooperative co-evolutionary approach is developed for GPHH which evolves a pair of dispatching rules for bottleneck and non-bottleneck machines in the dynamic environment with uncertainty in processing times arising due to varying machine characteristics. The results show that the new representations and training approaches are able to significantly improve the performance of evolved dispatching rules.

This thesis develops a new GPHH framework in order to incorporate active learning methods toward sampling DJSS instances which promote the evolution of more effective rules. Using this framework, two new active sampling methods were developed to identify those scheduling problem instances which promoted evolution of effective dispatching rules. The results show the advantages of using active learning methods for scheduling under the purview of GPHH.

This thesis investigates a coarse-grained model of parallel evolutionary approach for multi-objective dynamic job shop scheduling problems using GPHH. The outcome of the investigation was utilized to extend the coarse-grained model and incorporate an active sampling heuristic toward identifying those scheduling problem instances which capture the conflict between the objectives. The results show significant improvement in the quality of the evolved Pareto set of dispatching rules.

Through this thesis, the following contributions have been made. (1) New representations and training approaches for GPHH to incorporate uncertainty information about processing times of jobs into dispatching rules to make them more effective in a practical shop environment. (2) A

new GPHH framework which enables active sampling of scheduling problem instances toward evolving dispatching rules effective across complex shop scenarios. (3) A new active sampling heuristic based on a coarse-grained model of parallel evolutionary approach for GPHH for multi-objective scheduling problems.

List of Publications

1. Karunakaran, Deepak, Yi Mei, Gang Chen, and Mengjie Zhang. "Dynamic job shop scheduling under uncertainty using genetic programming." In Proceedings of the Intelligent and Evolutionary Systems, Springer, 2017, pp. 195-210, Canberra.

2. Karunakaran, Deepak, Yi Mei, Gang Chen, and Mengjie Zhang. "Evolving dispatching rules for dynamic Job shop scheduling with uncertain processing times." In Proceedings of IEEE Congress on Evolutionary Computation (CEC), IEEE, 2017, pp. 364-371, San Sebastian.

3. Karunakaran, Deepak, Yi Mei, Gang Chen, and Mengjie Zhang. "Toward evolving dispatching rules for dynamic job shop scheduling under uncertainty." In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), ACM, 2017, pp. 282-289, Berlin.

4. Karunakaran, Deepak, Gang Chen, and Mengjie Zhang. "Parallel multi-objective job shop scheduling using genetic programming." In Proceedings of Australasian Conference on Artificial Life and Computational Intelligence (ACALCI), Springer, 2016, pp. 234-245, Canberra.

5. Karunakaran, Deepak, Yi Mei, Gang Chen, and Mengjie Zhang. "Sampling Heuristics for Multi-objective Dynamic Job Shop Scheduling Using Island Based Parallel Genetic Programming." In Proceedings

of International Conference on Parallel Problem Solving from Nature (PPSN), Springer, 2018, pp. 347-359, Coimbra.

6. "Active Learning Methods for Dynamic Job Shop Scheduling under Uncertainty Using Genetic Programming based Hyper-heuristics" (Journal paper in preparation).

Acknowledgments

I would like to thank my parents and wife for their continuous support and sacrifices throughout this journey.

I am grateful to Victoria University of Wellington for the financial support through the Victoria Doctoral Scholarship and Victoria Doctoral Submission Scholarship. I am also grateful to the administrative staff at the Engineering and Computer Science school for their support throughout my study.

I would like to acknowledge my gratitude to my supervisors Prof.Mengjie Zhang, Dr.Aaron Chen and Dr.Yi Mei for their support and supervision toward the completion of this PhD thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter provides the introduction to the thesis. It starts by describing the problem statement followed by the motivations for this research work. The research goals and the major contributions are described next. Finally, the organization of the thesis is presented.

1.1 Problem Statement

The job shop scheduling problem is a combinatorial optimization problem [75]. It has a wide range of practical applications in industrial process [179], airline scheduling [61], distributed computing systems [15], and many other domains [63]. The job shop scheduling problem deals with the assignment of tasks or jobs to different resources or machines. The quality of schedule depends on the objective(s) of the problem, e.g., the completion time or makespan. In practice, most of these problems are NP-hard [78]. When the complete information of jobs and machines is known and it does not change with time, it is known as a static job shop scheduling problem. On the other hand, when new jobs arrive with no prior information known about them, the problem is more challenging and is known as a *dynamic* job shop scheduling problem (DJSS) [194, 156]. Furthermore, in practice, the parameters of the jobs such as processing times and release

dates are **uncertain**. For example, the release of a job maybe pushed back due to unavailability of raw material, the processing of a job may get delayed due to sudden power outage. Generating (near) optimal schedules under dynamic and uncertain environments makes the job shop scheduling problem more challenging and mostly infeasible.

Exact optimization methods have been used for finding solutions to particular problem instances of job shop scheduling. These methods are slow and need to be performed again for new instances [182]. For dynamic scheduling, when a new (set of) jobs arrive, the algorithm needs to be run all over again. Heuristic methods are fast and appropriate for practical scenarios but do not give any analytical bounds on optimality. Developing heuristics (*dispatching rules*) is time consuming but once generated, the dispatching rules are able to generate schedules very fast. Therefore, dispatching rules are more suitable in such practical applications [225, 161]. Dispatching rules generally use machine and job attributes [172]. Examples of simple dispatching rules [172] are: (select job with) shortest processing time (SPT), fewest operations remaining (FOPNR), minimum setup-time (MINSEQ), operation where machine has least work (WINQ), etc. Dispatching rules can further be composed of multiple job and machine attributes, e.g., SPT/FIFO selects the job by sorting them shortest processing time and then by the arrival time. Such dispatching rules are called *composite* dispatching rules.

However, manually designing dispatching rules is challenging because it requires domain expertise and rigorous experimentation. Therefore, machine learning methods have been successfully used to design composite heuristics which can capture complex relationships among the variables of a job shop [187]. At the interface of machine learning and operations research lies the area of hyper-heuristic approaches toward automating the design and adaptation of heuristic methods. Basically, hyper-heuristics are heuristics which in turn produce problem-solving heuristics rather than the final solutions. In order to generate the dispatching rules automat-

ically, *hyper-heuristics* approach has been proposed [42]. The heuristics produced are typically dispatching rules which are then used to generate the schedules. Genetic programming based hyper-heuristic (GPHH) approaches have been successfully used for developing the dispatching rules [30, 41, 157, 161] in job shop scheduling. Considering the flexible representation of genetic programs this approach has shown to be very promising [157]. Many effective dispatching rules particularly for dynamic job shop scheduling problems have been developed recently e.g. ASP2013-Rule #6 [162], EC2014-TREE_EXT_NORM_ND-Rule [33], etc.

In practice, uncertainty is ubiquitous in shop environments [110, 141, 142]. DJSS problems are characterized by continuous arrival of new jobs to the shop and no prior information about them is known. Most research on job shop scheduling use a deterministic model [109]. In the deterministic case, once the information of a new job is known, it stays constant. However, in an uncertain scenario, the information varies at the time of realization of the schedule. For example, the processing time of a job varies when a schedule is realized and is different from its *expected* value. Moreover, the dynamic nature of DJSS problems leads to variability in the shops which is closely associated to the uncertainty. In practice, the job shop environment always has uncertainty which makes scheduling a challenging and difficult task [125]. Handling uncertainty during scheduling is of practical importance. Processing time variability, change in job arrival pattern, equipment downtime, resource outage, demand uncertainty, poor performance of control systems, etc. are some of the many sources of uncertainty [141].

Furthermore, when such a practical shop environment is considered, a number of complex scenarios arise, more so, when multiple objectives are considered. Developing methods for scheduling by dealing with the challenges of such *practical* environments is the major motivation behind this research. It has been shown in literature [137] that dispatching rules deal better with uncertainty compared to other algorithmic solutions. Considering the flexible representation of genetic programs and its ability to

represent complex features of job shop scheduling, developing heuristic approaches under the purview of uncertainty is a current research direction [34, 164]; particularly for dynamic scheduling [240]. Even though GPHH has been a good tool to design dispatching rules, more work is required for their applicability to practical shop environments, for example, by dealing with the aspects of representation and computational aspects in GPHH. The current studies on GPHH do not focus on these issues.

1.2 Motivations

In shop environments with uncertainty, the dynamic arrival of jobs manifests into varied scenarios for scheduling. In order to further demonstrate the kinds of issues arising due to uncertainty, we cite more practical examples from scheduling problems [104, 130, 189, 191]. An automobile production line is required to be configured to produce cars with different specifications, e.g., specific leather seating, choice of standard or premium wheels, exterior paint color and other specifications for accessories. Variability in these specifications causes uncertainty in set up times, and other delays. Similarly, in print industry, where scheduling and planning is an important activity, the jobs arrive dynamically to the print shop. A print job requires resources like printers, cutters, collators and other similar equipment. Machine breakdowns, operator's breaks and complex machine set-ups leading to delays are some of the sources of uncertainty in this shop environment [104]. The varying characteristics of the arriving jobs along with the sources of uncertainty has a detrimental effect on the scheduling objective(s). The arriving jobs show a pattern in their characteristics [104], e.g., recurring marketing print jobs, transactional reports etc. Consequently, in order to address these problems, [104] considers dividing the shop into cells and assigning the jobs to different cells based on their specifications. They also take into account the effects of machine breakdowns and operator breaks in this consideration. Furthermore, [190]

proposes grouping the jobs by classifying them based on their features (e.g. number of resources required) and then use specific scheduling policies (simple dispatching rules) for the different cells. Essentially, in order to alleviate the problems described above, these methods are trying to divide the DJSS problems based on different shop **scenarios** and then solve them independently using specific scheduling policies.

We had stated earlier that for DJSS problems the dynamic nature of shop and the uncertainty are closely related. In the previous example this point was illustrated many times. For instance, the arrival of jobs with different characteristics requires complex set-ups of machines. This causes uncertainty in processing times of the jobs. For generating good schedules in practical shop environments taking uncertainty into account is crucial. Previous works have leveraged the flexible representation of genetic programs to evolve good dispatching rules for DJSS problems [100, 158]. It has been shown that GPHH has the ability to incorporate complex shop information in the form of machine and shop attributes into the evolved rules. Therefore, considering the importance of taking into account the effect of uncertain shop parameters in the shops, developing methods to incorporating the uncertainty information into the dispatching rules seems to be an important research direction and a good step to generate effective schedules.

But the performance of an evolutionary algorithm and GP in particular is highly dependent on the choice of representations which in turn is highly influenced by the characteristics of the problem and its difficulty [197]. Therefore, unless aided by better representations, the existing GPHH approaches will not be able to evolve rules for practical JSS problems in an uncertain shop environment. To conclude, though GPHH has shown potential and is a good candidate for evolving dispatching rules which take uncertainty into account, its success will depend on the underlying choice of representations. The new GPHH representations should be able to incorporate the uncertainty information into the dispatching

rules e.g. through newly defined terminals. Furthermore, motivated by the ability of genetic programs to incorporate complex features, it is also worth exploring methods for classifying the shop scenarios corresponding to the different levels of uncertainty which could aid the evolution of an effective set of rules.

From the examples of practical job shops described earlier, we can recognize the importance of designing scenario-specific dispatching rules. There are some other works [88, 209], which have highlighted the fact that designing a single dispatching rule to work well across all the scenarios is not possible. From the perspective of hyper-heuristics, particularly GPHH, it is desired to *automatically* design rules for different shop scenarios. In a DJSS problem, the variability in the shop arising due to its dynamic nature defines the shop characteristics. For a GPHH system to automatically design scenario-specific rules, it is necessary to extract information (features) from the shop characteristics e.g. [213]. This brings about the need to utilize machine learning techniques to identify the shop scenarios and group them. Theoretically, it is possible to identify infinite number of such shop scenarios. However, GPHH is a computationally expensive approach and evolving rules for a very large number of shop scenarios is not feasible. Therefore, which of the shop scenarios to select for learning the rules, or more particularly, which of the DJSS problem instances representing the shop scenarios should be selected is an important question.

Active learning [204], which is a sub field of machine learning, is based on the idea that a learning algorithm will perform better with less training if it is allowed to choose its training data. More formally, active learning problems involve selectively and adaptively sampling from the input space toward estimating unknown parameters. Active learning approaches have been quite successful in areas like semi-supervised learning [51]. Genetic programming could also be considered as a machine learning tool. With respect to the research question described above, we

want the GPHH system to be provided with training instances which can aid in learning of effective dispatching rules. Clearly, the research question can be seen under the purview of an active learning problem.

The current active learning methods, like uncertainty sampling technique [3], when applied to problems like classification, try to leverage the underlying distribution of the input space and sample those instances which present with maximum information to the learning system. It is also quite straightforward to provide feedback to the active learning method, for example by measuring the error in a classification problem which can then be used by the method in further iterations (e.g. expected-error reduction technique [204]). In the context of GPHH, this is not straightforward because GPHH is a hyper-heuristic approach and by using a sample of DJSS instance for training, if a dispatching rule is evolved, it cannot directly convey any information about the efficacy of the used sample instance. The evolved dispatching rule needs to be further evaluated on a set of new DJSS instances and its performance on these new instances is then indirectly a measure of the quality of the sampled DJSS instance. Clearly, we need a more powerful GPHH framework which can handle both these tasks simultaneously. One task focuses on evolving rules using sampled DJSS instances while the other task evaluates the sampled DJSS instance indirectly evaluating the evolved rule. Since our computational budget is limited, we essentially need to find a tradeoff between assigning computational resource to these tasks. Essentially, this is a multi-armed bandit problem.

In a multi-armed bandit (MAB) problem [16], a fixed set of resources must be allocated to competing choices (arms) to maximize profit. The intuition is developed by imagining a gambler who has to pull a slot machine with multiple arms. Each arm is associated with a different probability of winning, which are unknown to the gambler. The gambler, who has limited resources (number of times he can pull the arm), faces the dilemma of whether to explore the arms to find the optimal arm which gives maxi-

mum reward or to exploit the arm which is currently the most rewarding. This is called the *exploration versus exploitation* dilemma. By analogy, exploring the space of DJSS training instances while exploiting the already identified good training instances is the multi-armed bandit problem in our context.

With this background, we can see that these ideas motivate a research direction toward using MAB framework and active learning techniques in GPHH to evolve dispatching rules which can perform well on a large number of complex shop scenarios. There are some similar works which explore such ideas for other problems [12, 32]. Furthermore from these discussions, it is also clear that the existing GPHH framework lacks the provisions to incorporate these techniques and efficiently address the exploration versus exploitation dilemma discussed above.

As we had mentioned earlier, a major motivation for this research is to develop methods for practical scheduling problems. In practice, it is important for considering more than one objective in a scheduling problem. For example, considering the earlier example [104] from print industry apart from makespan some of the other objectives considered were number of late print jobs and turn around time. GPHH becomes computationally more expensive when dealing with an increasing number of objectives because of the higher complexity of EMO algorithms [57]. When we consider the complex shop scenarios for multi-objective DJSS problems the application of active learning techniques becomes more complicated. This is because the solution of a multi-objective optimization problem is a Pareto set of solutions instead of a single solution. Evaluation of Pareto sets is computationally more expensive than evaluating a single solution. Therefore the exploration task discussed earlier with respect to the single objective case becomes much more expensive. Therefore, the computational issues and the difficulty in using the existing GPHH framework for MAB and active learning techniques become exacerbated for multi-objective DJSS.

Parallel EAs have been very useful when it comes to speeding up the evolutionary process. Because of ease of parallelizability of evolutionary algorithms, it is encouraging to address the aforementioned computational issues using parallel evolutionary algorithms [220]. There are two main categories of parallelization, *parallelizing an independent run* and *island models*. Island models are particularly interesting because of their ability to deal with local optima [220]. The island model uses a spatially structured network of subpopulations (on different processors) to exchange promising individuals among each other in an effective approach. One of the major advantages of the island model is that it inherently captures the dynamics of *exploration* versus *exploitation* through the design of its migration policies and topologies. Due to this feature it is motivating to consider island model for developing active sampling methods which exploit this feature along with parallelization of GPHH for multi-objective DJSS problems. The success of some works like [244] in applying island models for MOEAs is further motivating. Even though there is good theoretical foundation for island models, we still lack understanding of many aspects of island model. Moreover, the design choices such as migration frequency and island topology have a big impact on the effectiveness of island models. Therefore, for solving any problem using the island model for parallel EAs, determining these design parameters is crucial.

1.3 Research Goals

The overall goal of this thesis is to develop effective and efficient genetic programming based hyper-heuristic approaches using active learning techniques for dynamic job shop scheduling problems for one or more objectives.

Following are the more specific research questions to be addressed in this thesis.

- *How to incorporate uncertainty information of the shop into the dispatching*

rules? How to evolve rules for different levels of uncertainty in the shop?

The representation in genetic programming is a key aspect which decides the performance of GP for any learning task. In fact, for JSS problems many different GP representations have been considered [161]. We aim to develop new representations for genetic programming which can incorporate uncertainty information into the dispatching rules. More specifically, we will try to find what kind of new terminals could be used in genetic programming when uncertainty in processing times of the jobs is considered. Moreover, since the uncertainty levels vary in a dynamic shop, we will try to develop new methods which can detect these variations and explore the ability of GPHH to develop dispatching rules for these varying scenarios.

- *How to use machine learning techniques e.g. clustering, to identify and group the varying shop scenarios? What active sampling methods to use for identifying good training instances while evolving dispatching rules?*

  Since GPHH is computationally demanding, it is not feasible to consider the training instances from all shop scenarios for training. Moreover, like in machine learning problems, not all training instances have the same potential to be used for evolving good rules. For example, a very easy problem instance might not challenge the evolutionary algorithm, resulting in evolved rules which cannot generalize well on more difficult problems. A major challenge is to identify these potentially good training instances while simultaneously evolving the rules. Therefore, we need a strategy to actively search for good training instances while using the already sampled training instances. Basically using the currently sampled DJSS instances for evolution is exploitation where as searching for better instances is exploration. Thus, one of the major research goals is to tackle this exploration versus exploitation dilemma.

Furthermore, in order to aid the active sampling approach, we need to associate the DJSS instances with shop scenarios. Essentially, we aim to develop methods which can extract features from DJSS problem instances so that we can cluster them together where each cluster corresponds to a shop scenario. We will identify the features of the shop which characterize the different shop scenarios and develop a feature extraction procedure for the DJSS instances.

- *How to use parallel evolutionary methods for evolving a Pareto front of dispatching rules for multi-objective DJSS problems? How to incorporate active sampling techniques in this framework?*

  Evolutionary multi-objective algorithms are generally computationally more expensive than their single-objective counterparts [57]. Therefore, extending the active sampling methods to multi-objective DJSS problems is hard because active sampling framework requires exploration, which in turn will need frequent and expensive evaluation of Pareto fronts. Moreover, island models have the inherent ability of tackling exploration versus exploitation dilemma [168].

  Thus one of the important research questions is, how to use the *island model* framework to evolve dispatching rules for multi-objective DJSS problems? Secondly, how to leverage the potentially useful features of island models, i.e. its migration policies, to develop an active sampling method for multi-objective DJSS problems?

In order to address these research questions, the following research objectives have been framed.

1. Develop new terminals for genetic programs and training methods for genetic programming to incorporate uncertainty information into dispatching rules. Develop a co-operative co-evolutionary approach to evolving dispatching rules for different levels of uncertainty.

We will develop new methods and training approaches which focus on encapsulating the uncertainty information directly into the terminals of the dispatching rules.

The varying uncertainty levels in the shop manifest as changes in the bottleneck characteristics of the machines [124]. Also in [106], bottleneck characteristics have been considered for a static JSS problem. This needs further investigation when we consider a dynamic JSS problem because the continuous arrival of jobs could result in variations in these bottleneck levels. Classification of these bottlenecks could help in evolving rules with different characteristics and will therefore capture the uncertainty information. The objective is to develop a cooperative co-evolutionary approach to evolve dispatching rules for machines with different bottleneck levels.

2. Develop a new GPHH framework which incorporates active sampling strategies toward identifying potentially good training instances and evolve scenario-specific dispatching rules effectively for DJSS problems under uncertainty.

   Toward developing methods for effective active sampling of DJSS instances in GPHH tackling the exploration versus exploitation is identified as a key research goal. We will develop multi-armed bandit techniques which can tackle this issue. Due to characteristics of the hyper-heuristic approaches (which essentially search in the heuristic space and then the searched heuristic generates a solution), incorporating MAB techniques into current GPHH framework is problematic. Therefore we will develop a new GPHH framework which will enable the active sampling and MAB techniques to work together.

3. Develop island model approaches to evolving dispatching rules for multi-objective JSS problems. Develop active sampling heuristics using the island model approach for DJSS problems under uncertainty.

Designing a parallel EA system using island model requires identifying appropriate design parameters for the island model to be effective. These parameters depend on the problem domain. Therefore, we will explore different design choices for the island model toward evolving dispatching rules for the multi-objective DJSS problem.

Furthermore, we will develop an active sampling heuristic which identifies potentially useful DJSS instances by leveraging the island model topology and migration policies. The heuristic will promote evolution of better Pareto set of dispatching rules for multi-objective DJSS problems.

1.4 Major Contributions

This thesis has made the following major contributions:

- This thesis has shown how to incorporate uncertainty information into GPHH for DJSS. New methods are investigated to incorporate uncertainty information, each exploiting the ability of genetic programming to have flexible and novel representations. Furthermore, a cooperative co-evolutionary method is developed to classify the machines as bottle-neck and non-bottleneck (which are an effect of varying uncertainty levels in processing times) and to evolve a pair of dispatching rules for each machine type. The results show that our proposed methods are successful in incorporating the uncertainty information into the evolutionary learning framework and are able to evolve dispatching rules which show significant improvement in performance.

  Part of this contribution has been published in:

  - Karunakaran, Deepak, Yi Mei, Gang Chen, and Mengjie Zhang. "Dynamic job shop scheduling under uncertainty using genetic

programming." In Proceedings of the Intelligent and Evolutionary Systems, Springer, 2017, pp. 195-210, Canberra.

- Karunakaran, Deepak, Yi Mei, Gang Chen, and Mengjie Zhang. "Evolving dispatching rules for dynamic Job shop scheduling with uncertain processing times." In Proceedings of IEEE Congress on Evolutionary Computation (CEC), IEEE, 2017, pp. 364-371, San Sebastian.
- Karunakaran, Deepak, Yi Mei, Gang Chen, and Mengjie Zhang. "Toward evolving dispatching rules for dynamic job shop scheduling under uncertainty." In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), ACM, 2017, pp. 282-289, Berlin.

- This thesis has shown how a new GPHH framework is established to support the methods trying to tackle the exploration versus exploitation dilemma. The salient feature of the new framework is that it introduces a validation step into GPHH which aids the active sampling method. Essentially, the training and validation are exploitation and exploration respectively. Using this framework two active sampling approaches are developed to sample potentially good training instances, namely ϵ-greedy method and Gaussian process bandits approach. Both these methods are inspired from multi-armed bandits literature. The results show that GPHH is more effective when evolving scenario-specific rules for DJSS problems.
  - "Active Learning Methods for Dynamic Job Shop Scheduling under Uncertainty Using Genetic Programming based Hyper-heuristics" (Journal paper in preparation).

- This thesis investigates the migration policies and topologies for island model based parallel evolutionary approach for job shop scheduling problems. Then this approach is extended for DJSS problems to-

wards developing an active sampling heuristic which exploits features of island model to identify potentially more useful training instances. The proposed heuristic enables the GPHH to successfully evolves Pareto front of dispatching rules which outperforms the rules evolved using EMO approaches without using such a heuristic. Further analysis of the sampled training instances shows that they represented scenarios which highlight the conflicting nature of objectives, which is in line with our expectations.

Part of this contribution has been published in:

- Karunakaran, Deepak, Gang Chen, and Mengjie Zhang. "Parallel multi-objective job shop scheduling using genetic programming." In Proceedings of Australasian Conference on Artificial Life and Computational Intelligence (ACALCI), Springer, 2016, pp. 234-245, Canberra.
- Karunakaran, Deepak, Yi Mei, Gang Chen, and Mengjie Zhang. "Sampling Heuristics for Multi-objective Dynamic Job Shop Scheduling Using Island Based Parallel Genetic Programming." In Proceedings of International Conference on Parallel Problem Solving from Nature (PPSN), Springer, 2018, pp. 347-359, Coimbra.

1.5 Organization of thesis

This thesis is organized as follows. Chapter 2 presents a literature survey of related works. Chapters 3-5 address the three objectives and Chapter 6 concludes the thesis.

Chapter 2 presents a detailed description of DJSS problems and GPHH framework. The basics concepts of genetic programming are explained. Background and related work on job shop scheduling under uncertainty is provided. It also provides review of current research in hyper-heuristics with more focus on work related to research objectives of this thesis.

Chapter 3 presents the methods developed to incorporate uncertainty information into dispatching rules. This chapter discusses two main approaches which are developed for achieving this objective. The results from both the methods are analysed to get more insights.

Chapter 4 presents a new GPHH framework and its three components, namely training, testing and validation. The chapter presents two active sampling methods and compares the evolved rules with the standard GPHH framework.

Chapter 5 proposes island model approaches for DJSS problems with multiple objectives. Firstly, static job shop scheduling problems are considered for investigating the island models. Then active sampling heuristics based on island models for DJSS problems under uncertain processing times are presented.

Chapter 6 summarizes the thesis. In particular, the key findings and conclusions are listed. Also opportunities for future work are presented.

Chapter 2

Literature Review

This chapter starts by introducing the concepts of scheduling, in particular, the formal introduction of job shop scheduling problems. Then we provide a background for considering uncertainty in scheduling problems. After the problem discussion, a review of basic concepts in machine learning and hyper-heuristic approaches is presented. Following that we provide an overview of the evolutionary computation approaches. Then we introduce the active learning methods and also highlight evolutionary approaches which have used active learning concepts. We also provide a detailed overview of parallel evolutionary algorithms. The key concepts of genetic programming and GP based hyper-heuristics are introduced. We present a detailed review of solution approaches for job shop scheduling, with a particular focus on GPHH approaches. We also discuss approaches for dealing with uncertainty in scheduling problems. This is followed by a brief review of parallel hyper-heuristics and MOEAs. Finally, the summary of literature review highlighting the gaps in the literature is presented.

2.1 Scheduling

Scheduling [182, 52] is the decision making process of allocating limited resources to task over a time period with the goal of optimizing one or more objectives. For different application domains the resources and tasks take different forms. For example, in an airport, the runways and the gates are the resources and take-off and landings are the tasks; in a construction project, crews and workmen are resources who need to complete different construction tasks; in a computing environment, computers are the resources and execution of the computer programs are tasks. Scheduling is an important process in most manufacturing industries and computing environments along with most of the service industries.

Considering the large number of applications of scheduling the number of scheduling models considered is also high. We could describe a scheduling problem with a triplet [52] as $\alpha \mid \beta \mid \gamma$. The parameter α is used to define the machine environment viz., single machine (1), identical machines in parallel (Pm), machines in parallel with different speeds (Qm), unrelated machines (Rm), flow shop (Fm), flexible flow shop (FFc), job shop (Jm), flexible job shop (FJc), open shop (Om) etc. The parameter β could be associated with multiple entries and define the processing constraints and restrictions e.g. precedence constraints (prec), breakdowns (brkwdn) etc. The final parameter γ is associated with the scheduling objective. Makespan, weighted tardiness, flow time are some of the scheduling objectives frequently considered. We describe them in more detail later.

For example, a flow shop consists of m machines and each job goes through all the machines in a fixed route. A generalization of the flow shop is a flexible flow shop which has many stages and there are identical machines in each stage. The jobs move from stage to stage in fixed sequence but can be processed in of the machines in a stage. With respect to the notation above, $FFc \mid r_j \mid \sum w_j T_j$ denotes a flexible flow shop with

release dates (r_j) and the objective is to minimize weighted tardiness. Another example, $Jm||C_{max}$ is a job shop with makespan as the scheduling objective. Each job visits a machine only once. This is a classical model for scheduling and a large number of research works have considered it. In this thesis, we focus on job shop scheduling.

2.1.1 Job Shop Scheduling Problem

Job shop scheduling problem, also known as the *sequencing* problem, deals with assigning jobs to machines (resources) at particular times. We formally describe this problem by using the notation in Table 2.1.

Symbol	Definition
$\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_m$	are m machines.
$\mathcal{J}_1$,$\mathcal{J}_1 \ldots \mathcal{J}_N$	are N jobs.
μ_j	number of operations in a job $\mathcal{J}_j$.
$\mathcal{O}_{ji}$	$(\mathcal{M}_{j1}, p(j,1)), \ldots, (\mathcal{M}_{j\mu_j}, p(j,\mu_j))$ is a sequence of μ_j operations where each operation is the (machine,processing time) pair in job $\mathcal{J}_j$.
$R_{\mathcal{O}_{ji}}$	operation ready time.
$R_{\mathcal{J}_j}$	job release time.
$\mathcal{S}$	A job shop schedule.
$\mathcal{C}_j$	completion time of job $\mathcal{J}_j$.
$\mathcal{C}_{max}$	makespan.
$\mathcal{T}_j$	tardiness of job $\mathcal{J}_j$
$\mathcal{D}_j$	Due date for a job $\mathcal{J}_j$.

Table 2.1: Notation

In a job shop with m machines, each job $\mathcal{J}_j$ that arrives has μ_j operations. Each operation $\mathcal{O}_{ji}$ could be denoted as a tuple. For example, the

i^{th}operation, $(\mathcal{M}_{ji}, p(j,i))$ of the job $\mathcal{J}_j$ should be processed on the machine $\mathcal{M}_{ji}$ and is associated with a processing time of $p(j,i)$. Thus a job $\mathcal{J}_x$ will have μ_x such tuples. The operation ready time $R_{\mathcal{O}_{ji}}$ is the time when an operation is prepared and set to begin processing over the machine. The operation ready time of the first operation in a job is called its job release time $R_{\mathcal{J}_j}$.

An operation is performed on a machine without interruption. In other words the operations are *non-preemptive*. The operations follow *precedence* constraints i.e. an operation cannot be started till its preceding operation is not completed. If all jobs have same order of processing of operations, then it is called the $flow$ shop scheduling. A variant of job shop scheduling is the *flexible* job shop scheduling problem in which an operation has the flexibility of being processed on more than one machine. If the information about the jobs is not known apriori and new jobs can randomly arrive at the job shop over time, it is known as *dynamic* job shop scheduling problem. When the number of jobs is fixed and their information is known, it is called *static* job shop scheduling problem.

Classes of Schedules

A schedule is called as *non-delay* when no machine is forced to be idle if an operation is queued at it for processing. A schedule is *active* if a new schedule cannot be generated from it by changing the order of processing on the machines with at least one operation finishing earlier but none later. An optimal schedule must be active [216]. Non-delay schedules are a subset of active schedules as shown in the Figure 2.1 from [182]. A *semi-active* schedule is a schedule such that none of the operations can be completed earlier without altering the order of processing in any one of the machines. An active schedule is also a semi-active schedule but not necessarily the other way round.

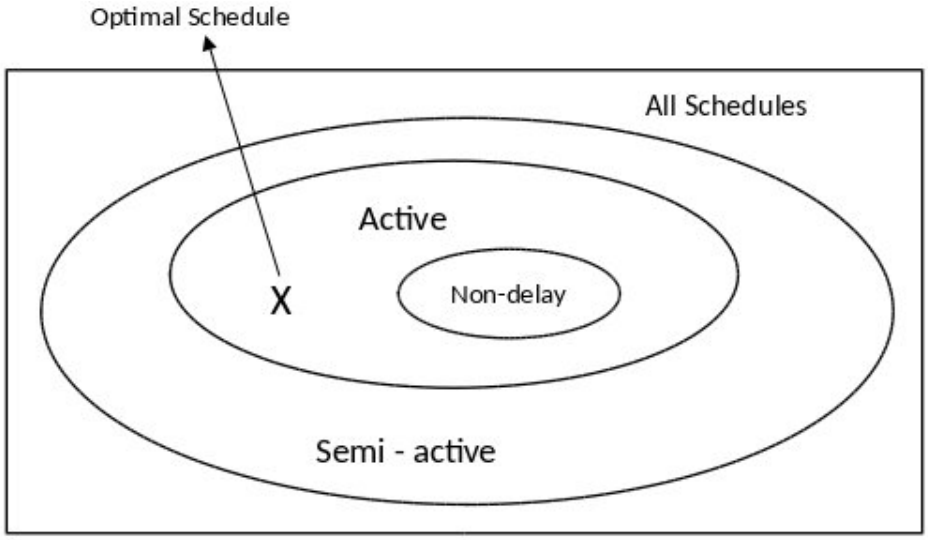


Figure 2.1: Venn diagram of classes of non-preemptive schedules [182].

Objectives for Job Shop Schedules

A schedule generated for a job shop should optimize some objective. Job shop scheduling problem is NP-hard [138]. Depending on the objective which the schedules tries to optimize, the hardness of the problem varies. The most frequently used quality measure is the *makespan*. If $\mathcal{C}_i$ is the completion time of the job $\mathcal{J}_i$, then makespan is defined as:

$$\mathcal{C}_{max} = \max_{\mathcal{J}_i} \mathcal{C}_i. \tag{2.1}$$

An optimal schedule for makespan is thus expected to result in minimum makespan for the job shop.

For a job shop scheduling problem, a schedule could satisfy many different criterion [227]. Flowtime and tardiness are the two other objectives which are considered. They are more directly linked to the cost of processing and customer satisfaction [21, 225]. Flowtime of a job is the time spent on the shop floor, which is the difference between release time and completion time. A schedule with this objective should have minimum mean

flowtime for all jobs. The mean flowtime [182] is given below:

$$mean\ flowtime = \frac{1}{N}\sum_{i=1}^{N}(\mathcal{C}_i - \mathcal{R}_{\mathcal{J}_i})$$

The jobs are also associated with a *due date*, $\mathcal{D}_j$. A schedule should be such that, *tardiness* which is the positive difference between the completion time $\mathcal{C}_j$ and due date $\mathcal{D}_j$ is minimized. If the jobs are weighted (penalty factor w_j), then the objective to be minimized is known as total weighted tardiness and is of the form shown below:

$$total\ weighted\ tardiness = \sum_{j=1}^{N} w_j \times max(\mathcal{C}_j - \mathcal{D}_j, 0)$$

Different works have considered different sets of objectives for job shop scheduling problem. Mean flowtime, maximum flowtime and variance of flowtime, percentage of tardy jobs, mean tardiness, maximum tardiness and variance of tardiness are some of the objectives discussed in [193] for dynamic job shop and flow shop scheduling. They have developed dispatching rules for scheduling while considering each objective at a time. Similarly maximizing machine utilization, workforce utilization, minimizing total waiting time, waiting time variance, lateness, total machine workload, critical machine workload etc. are many other relevant objectives. With rising energy costs and concerns for climate change objectives related to energy usage which have come into significance lately. Some of them are energy minimization, peak power minimization, power cost minimization [145].

Now we discuss the different scheduling methods for job shop scheduling problems. A large number of scheduling problem instances are NP-hard [182]. Therefore, many optimization techniques based on heuristics and meta-heuristics have been proposed to solve them [248]. Based on the task availability the scheduling problems could be split in to dynamic and static. In static problems the information about all the jobs is available when the scheduling process starts, where as in dynamic problems,

the jobs continuously arrive at the shop and their information is available only after they have arrived. For these two kinds of problems the different scheduling approaches are discussed later.

2.1.2 Uncertainty in Scheduling

Most of the existing works assume that during scheduling the known data (e.g. processing times, release date) is constant [109]. In recent years consideration of uncertainties in the job shops has got more attention. In practice, a model of a scheduling environment is far from accurate. Process variables like processing times of operations, set-up times etc. are *uncertain*. Even though scheduling theory has been developed for many decades, it is still not used in practice as much as it should be, because of this very reason [109]. Aytug et al. [17] proposes four dimensions for categorizing uncertainty. (i) *Cause* is the object (e.g, material,process,etc.) and the (ii) state (ready, damaged, broken down etc.) leading to uncertainty. (iii) *Context* refers to environmental conditions e.g. operator skills. Majority of the research is context-free [17]. (iv) *Impact* refers to the consequence of the uncertain event e.g., quality,delay etc. It is impossible to address all sources of uncertainty but it is essential to reasonably alleviate the ones which are significant.

The sources of uncertainty could be classified into two categories. One is from the shop environment, e.g. machine failure, operator's error, delay in supply of raw material, etc. The other is from the parameters of the jobs, e.g. sudden change in job arrival rates, change in due dates, change in job priority, cancellation of jobs, etc. The uncertainty of the manufacturing system could be described as a *bounded form* description when there is not enough information to develop a probabilistic model. Otherwise generally a *probability description* is generated form the historical data. In [110] the probabilistic description of uncertainty has been discussed with normal probability distribution, difference of normal probability distribution,

general discrete probability distribution, binomial probability and poisson probability distribution for processing times. *Fuzzy description* has also been used to model uncertainty [239].

2.1.3 Dynamics in Scheduling

Most of the existing research works focus on *static* job shop scheduling problems [158]. In static JSSP, the information about all the jobs and the parameters of the shop are known and no new information is presented once the processing of jobs begins. In *dynamic* JSS problems, new information is revealed with time and the problem configuration changes e.g. new jobs can arrive continuously at the shop. Thus with every new job the scheduling problem is changed. When arrival of new jobs is considered, the problem definition in Section 2.1.1 must incorporate some more parameters. The arrival of the jobs is considered as a Poisson process as it is a good approximation of the job arrival in practice [162]. The Poisson process is defined with a parameter λ which controls the rate of job arrival.

In Section 2.1.2, uncertainty in the shop parameters was discussed. The uncertainty in a JSS problem refers to a situation when the values of shop parameters are known but not *exactly*. On the other hand, in the dynamic problem the problem configuration itself changes with time, e.g. due to arrival of new jobs, removal of machines due to breakdown, etc. Furthermore, a dynamic JSS problem could also be uncertain in the values of its parameters. It must be noted that many research works use the terms 'uncertain' and 'dynamic' to refer the same shop environment, generally referring to an uncertain shop as dynamic.

In Section 2.7, the solution approaches to job shop scheduling problems are discussed. It must be noted that the characteristics of solution methodologies are dependent on the nature of job shop scheduling problems. For a static problem, since the problem configuration does not change, it is possible to employ computationally expensive methods from mathemati-

cal optimization. On the other hand for dynamic JSS problems which vary with time it is not feasible to solve the problem again and again whenever a dynamic *event* occurs. In this context, an event is responsible for the change in problem configuration e.g. arrival of a job is an event.

2.2 Machine Learning and Hyper-heuristics

We introduce basic concepts of machine learning and hyper-heuristic technologies which are closely related to the objectives of this thesis.

2.2.1 Machine Learning

Machine learning is a field of artificial intelligence which primarily uses statistical techniques and enables the systems to learn patterns from data in order to improve performance on specific tasks, without providing explicit instructions.

Machine learning methods consist of three categories [28], namely (a) supervised learning, (b) unsupervised learning and (c) reinforcement learning. In supervised learning, the system learns from already existing data which contains the information about desired output obtained from past observations. Classifying emails into spam and useful is an example, where the past actions from the user who are used to determine the class of a new email. In other words, the data is labelled under the purview of supervised learning. In unsupervised learning, the training data is unlabelled and clustering is one example of learning from unlabelled data. Reinforcement learning concerns with an agent who must learn to take optimal actions in a specific environment towards an objective. In reinforcement learning [222], there is no strict input-output pair in training data, but the focus is on performance requiring a balance between exploration and exploitation. Some popular machine learning techniques include [28] decision tree, logistic regression, artificial neural networks and

genetic programming.

Building machine learning models requires data which usually resides in multiple data sets. Generally, three data sets are used to develop a machine learning model.

- **Training set**

  This is the first data set which is used to build the machine learning model. For example, in a neural networks model, the weights of the connections between neurons are determined using an appropriate algorithm (supervised learning) which uses the training set. For supervised learning, the training set usually consists of pairs of data, with an input vector and a corresponding class label. The learning process usually involves adjusting the model parameters toward fitting the model accurately on the training data. Gradient descent is an example of one such frequently used method. The model fitting could also include instance selection while estimating the parameters.

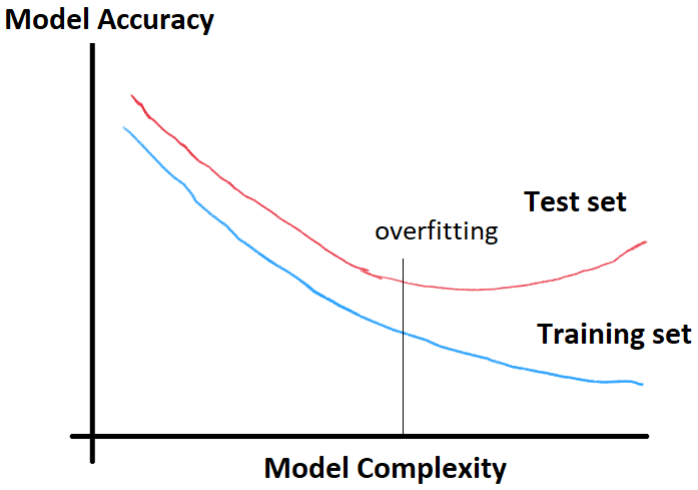


Figure 2.2: Overfitting

- **Validation set**

  Before a machine learning model is fit, some parameters are set for the algorithms which are called as hyper-parameters. These hyper-parameters have an impact on the quality and accuracy of the final model. Selecting these hyper-parameters is an important task. A validation set is generally used for this purpose. The model fitted on the training set is used to predict the responses on the validation set. This model accuracy is then used to finetune the hyper-parameters e.g. number of hidden units in a neural network. Validation stage could be used to prevent **overfitting** via regularization. When the machine learning model is being fit on the training set it is important to control the model complexity as eventually the model needs to be used on unseen data. If the model **overfits** the training set then it will not generalize well on unseen data. Model complexity could be regularized using validation stage, by stopping the training if the accuracy on validation set starts to drop. Note that a validation set cannot be used to induce/learn the learned model.

- **Test set**

  The test set is the dataset which is used to evaluate the final machine learning model. It consists of unseen data and gives an idea about the quality of the learned model.

In Figure 2.2, overfitting is explained in terms of model complexity. As the model complexity increases, the accuracy of the model on training set increases but it does not generalize well on the test set. The point at which the overfitting starts (shown using the vertical bar) is the point at which the training should be stopped.

2.2.2 Heuristics, Meta-heuristics and Hyper-heuristics

Heuristics are developed to solve problems which are computationally expensive when using methods, also called exact solvers, which determine optimal solutions. A large number of combinatorial optimization problems fall into this category, e.g. routing, bin pacing, scheduling [182]. According to [151] following are some of the important reasons for the difficulty in solving such problems: modeling complexity, large and heavily constrained search spaces, poor applicability of human problem solvers, etc. Heuristics are essentially thumb rules which are used to solve such problems with produce sufficiently good solutions with low computational cost but with no guarantee of optimality.

Meta-heuristics are non problem specific strategies to guide the search process employed to solve difficult optimization problems with the goal of efficiently exploring the search space and find near-optimal solutions. They are generally non-deterministic and use the low level heuristics to explore the solution search space. Based on their search strategy, they could be classified into *local search* and *population based*. The general tenet of local search methods is to start with a solution and then through improvement iteratively move towards better solutions. e.g. simulated annealing [121], tabu search [77], variable neighborhood search [85] etc. *Population based* heuristics e.g. ant colony optimization, evolutionary optimization and particle swarm optimization [184] consider a population of solutions to search the solution space. Many of these methods are nature-inspired and develop strategies based in evolution, ant behavior etc. Similar to heuristic approaches these methods do not give any guarantee to the optimality of a solution.

Hyper-heuristics are a class of search methods which generate heuristics to solve optimization problems [53]. Their goal is to automate the design of heuristics which solve the optimization problems, appropriately called as 'heuristics to search heuristics' [39]. Essentially, they search for a method which can solve the problem effectively rather than a direct solu-

tion to the problem. Hyper-heuristics approaches are generally not problem specific [39].

Even through heuristics have been effective in solving real world problems, designing them and tuning them for new problems or new problem instances is hard and time consuming. In particular, designing problem specific heuristics which are more effective is challenging. This requirement has led to the research in hyper-heuristic techniques which automate the design and tuning of heuristics for complex real world problems [40]. More specifically, the motivation behind the development of hyper-heuristics is based on two fundamental ideas. Firstly, recognizing the design and selection of efficient heuristics as a computational search problem in itself and secondly recognizing the potential of learning mechanisms to improve the search methodologies toward adaptively guiding the search [39]. In this thesis, we consider the following definition of a hyper-heuristic [39]: "A hyper-heuristic is a search method or learning mechanism for selecting or generating heuristics to solve computational search problems."

Hyper-heuristic approaches can be broadly classified into: (i) heuristic selection and (ii) heuristic generation. Heuristic selection methodologies focus on searching or selecting from the existing heuristics. Heuristic generation approaches are the class of methodologies which generate new composite heuristics by utilizing the components of the existing ones. Genetic programming based hyper-heuristics (GPHH) [41] is considered to better than many other learning mechanisms toward heuristic generation and have been very popular in the recent years. We will discuss GPHH in more detail later.

Another classification of hyper-heuristics is: learning and non learning learning hyper-heuristics [39]. If a hyper-heuristic approach utilizes feedback from the search process then it is called a learning hyper-heuristic and non-learning hyper-heuristic in the other case.The learning hyper-heuristics are further classified into (i) online learning hyper-heuristics

and (ii) offline learning hyper-heuristics depending on the source of feedback. In online learning hyper-heuristics the feedback is provided while solving a problem instance; use of meta-heuristics e.g. [38] for high level search strategies in one example. In the offline learning hyper-heuristics, the training instances are used to gather knowledge in the form of rules and programs which are expected to effectively solve unseen instances. Learning classifier systems and GPHH are two prominent examples.

Hyper-heuristic approaches have been successfully used to solve real world problems like production scheduling [158], bin packing [178], constraint satisfaction [11] and vehicle routing [150].

2.3 Evolutionary Computation

Evolutionary computation (EC) is a sub-field of artificial intelligence comprising of algorithms which are inspired from biological evolution. They are primarily population based algorithms with stochastic characteristics. Basically, a population of solutions are evolved using steps like selection and mutation toward improving their fitness (fitness definitions are problem specific).

2.3.1 Evolutionary Algorithms

Evolutionary algorithms (EA) are a main area of research in EC. Some popular algorithms EA algorithms from literature are:

- Genetic algorithms [152] (GA), which are frequently used for optimization and search problem. The individuals are represented as fixed-length arrays of bits, real numbers, integers, etc. and rely on nature-inspired crossover, mutation and selection operators.

- Genetic programming [126] (GP), which is closely related to GA; they use variable length computer programs to represent individuals. This flexible-representation enables GP to be applied to complex

problem in machine learning and artificial intelligence. We present a more detailed review later in Section 2.6.

- Evolution strategy is based on biological evolution and predominantly uses mutation and selection operators. Covariance matrix adaptation based evolutionary strategy (CMA-ES) [84] is a popular method for ES.

- Evolutionary programming [245] is similar to GP where the structure of its programs are of fixed length but the numerical parameters are evolved. They are capable of producing highly optimized solutions to problems. [19]

2.3.2 Swarm Intelligence

Swarm intelligence (SI) is a decentralized and self-organized system exhibiting collective behaviour which is employed in work on artificial intelligence [31]. SI systems consist of a population of agents whose interaction with each other as well as the environment is defined by simple rules. This interaction results in a emergent behaviours. Ant colonies, bird flocking, fish schooling and microbial intelligence are some of the examples of swarm intelligence in nature.

Two of the popular swarm intelligence techniques are ant colony optimization and particle swarm optimization.

- **Ant colony optimization** (ACO) [64] is a technique which is used very frequently applied to combinatorial optimization problems e.g., vehicle routing. Basically it simulates the behavior of an ant which leaves a pheromone on its path which keeps on vaporizing and is reinforced only if an ant takes the same path again. Eventually, the ants start using the shortest path. This idea has led to many extensions of ACO and have been applied to many real world problems.

- **Particle swarm optimization** (PSO) [119] is a nature inspired method which optimizes a problem by iteratively improving a candidate solution. Basically, it utilizes a population of particles and moves them toward the best solution by using a simple mathematical formulae which uses the position and velocity of the particle. PSO was inspired by fish schooling or bird flocking and has been successfully applied to problems in many areas including biomedical engineering, finance, networks, combinatorial optimization etc.

2.4 Active Learning

One of the challenges in machine learning problems is to obtain labelled instances for training [3]. For example, in problems like classification and filtering problems, speech recognition and information extraction a large number of labelled instances are required to achieve the required accuracy [204]. Moreover, many of these labelled instances are noisy and a higher accuracy can be achieved if such instances are removed from the training set. Active learning is a sub-field of machine learning which is based on the key idea that "a machine learning algorithm can achieve greater accuracy with fewer training labels if it is allowed to choose the data from which it learns" [204].

Broadly there are *three* strategies for active learning based on the learning problem. Firstly, *membership query synthesis* where the learner actively synthesizes instances from the input space, e.g., choosing arbitrary spatial co-ordinates for a sea surface temperature prediction system [3]. Secondly, *selective or sequential sampling* [204] in which case the samples are drawn from the input space and the learner makes a decision if it should be labelled and included for the training. Thirdly, *pool based sampling* is relevant for practical scenarios where there is a large pool of unlabelled data and a small set of labelled data. The active learner draws the samples from this pool and decides if it should be added to the training set. Most of the

active learners consider pool based sampling [127].

All these active learning approaches require the evaluation of the synthesized or sampled instance for it to be included into the training set. This is called as the query strategy. The key question for a query strategy to answer is: "Which sample should be selected so as to maximize the accuracy of the classification process?" [3].

Broadly the query strategies are classified into three categories.

- Heterogeneity-based models: This is one of the most commonly used query frameworks where the instance which is most dissimilar to the existing training data is chosen. Three of the widely used methods under this model are: (i) Uncertainty sampling; the active learner tries to label those instances for which the certainty of the label to be correct is the lowest [140]. (ii) Query-by-committee; a committee of different classifiers are trained on the existing set of labelled instances, and then these classifiers label each of the unlabelled instance. The instance with the highest disagreement among the classifiers is selected by the learner [207]. (iii) Expected model change, is applicable only to models where gradient-based training is used e.g. discriminative probabilistic models. It is essentially a decision theoretic approach which selects that particular instance which has the potential to show to contribute to the maximum change in the model *if* the sample were used to train the model [206].

- Performance-Based Models: One of the drawbacks of the heterogeneous based models is that heterogeneity could actually lead to noisy data. For such cases, performance based models which depend on the performance of classifier are employed. Two of these models are: (i) Expected error reduction, is a method which complements the uncertainty sampling in a way that it selects those instances which when added to the training data minimizes the label uncertainty [81], (ii) Expected variance reduction is similar to the

previous except that it reduces the variance and helps in reducing the computational costs [201].

- Representativeness-Based Models: These models query the data in such a way that the overall representativenes of the distribution of data is better. In other words, the selected instances are informative not only if they are uncertain but also need to be representative. Density based models are an example [205].

2.4.1 Active Learning in Evolutionary Algorithms

Active learning framework has been integrated into evolutionary algorithms to achieve different goals, though in a limited way. We survey such research works in this section.

Katsuna et al. [131] present an active learning algorithm based on genetic algorithms and self-organizing maps to interactively annotate images and accurately classify them. In [235] an active learning approach is used for classification problems with microarray data in conjunction with feature selection approaches based on genetic algorithms. [237] presents a surrogate assisted particle swarm optimization algorithm using committee based active learning method. Essentially, it uses an ensemble consisting of polynomial regression model, Kriging model and radial basis function model in a query-based committee active learning framework as the surrogate. The optimizer is a particle swarm optimization method. [200] develop an active learning approach based on Gaussian processes for multi-objective optimization problem using NSGA-II. Basically, it is also a surrogate assisted optimization applied to problems arising in sustainable building design.

2.5 Parallel Evolutionary Algorithms

Due to the advancements in computing hardware there has been a surge in the number of recent research works which employ parallel evolutionary algorithms [25, 115]. Evolutionary algorithms are embarrassingly parallel [43]. Parallel evolutionary algorithms are known to converge faster and also provide better performance [79].

Parallel EAs have been applied to many challenging problems, including NP-hard problems particularly from combinatorial optimization. [9] presents a survey of applications of parallel EAs. Applications of parallel meta-heuristics have been surveyed in [7, 54]. The survey focuses on applications like graph coloring, partition problems, travelling salesman problem and vehicle routing problems among others. [147] employs parallel EAs for workforce planning, natural language processing and problems from bioinformatics.

The parallelization of evolutionary algorithms can be done using many ways. For example specific operations could be parallelized or the whole evolutionary process itself could be parallelized. Some of the popular methods from the literature are discussed below.

Master Slave Models

Under master slave model [115], one of the machines represents the master which distributes the workload among the other machines which are the slaves. It is particularly useful for parallelizing specific operations on separate processors. For example, function evaluations which are generally expensive are good candidates for this model of parallelization. One of the disadvantages [80] of master slave models is that scalability beyond a certain level could lead to a situation where the master becomes a bottleneck. Moreover, keeping the workload balanced in heterogeneous systems is challenging. The master slave model is generally synchronous whose behaviour is not much different from sequential EAs [115]. Though asyn-

chronous models are also possible [115] provided the master does not wait for too long.

Independent Runs

By independently running [44] the evolutionary algorithms on different machines, where each run is different from the other, the probability of obtaining the best result in less time is increased. This is called probability amplification [136]. Essentially there is no communication between the different machines and for that reason the setting up such a system is relatively easy. Some communication could be introduced between the machines so that they can stop if the optimal solution is found in one of them. Furthermore, they have been used to explore the parameter space of evolutionary algorithms [115]. If there is large variance in running times among different runs, employing independent runs is rewarding in problems with local optima, because the optimization time is the time until global optima is found [70, 243].

Island Models

The previous two approaches for parallelization not consider any systematic form of communication between the different machines. The main advantage of parallelization in both the cases is more efficient running times [115]. In the island model [115], the population associated with each machine is called an island, but unlike the independent runs, the islands periodically exchange solutions among them.

The island models are also known as coarse grained model or multi-deme model. The process of periodically exchanging the solutions is called migration. The benefit of this migration between the islands was first shown in [133]. The island model requires a migration topology, which is a directed graph with islands as its nodes. The connected islands can communicate with each other through the exchange of individuals, also

called as migrants. This communication is defined by a directed graph whose each node is an island. An island which is stuck in local optima could use a good individual from another successful island and consequently focus on more promising regions of the search space. Thus the migration enables the effective usage of available resources.

Lassig et al. [133] show that a proper migration policy and choice of topology is essential for the parallelization based on island model to be useful. Martin et al. showed that the different islands maintain diversity by maintaining different promising regions of search space. Essentially, the island models, by design, incorporate the dynamics of **exploration and exploitation**. Through diversity maintenance across the different islands the exploration is promoted while focusing on a specific region of search space, the islands promote exploitation [55]. The dynamics of exploration and exploitation have been empirically demonstrated further in [168].

These works have highlighted that the choice of appropriate design of island model is very important for it to be effective, which is defined by the following elements.

Emigration policy decides if the migrant individual should actually be copied or removed from the source island. The selection of individuals for migration is very important e.g. selection of best, worst or random individuals could lead to different outcomes.

Immigration policy decides which individuals should be replaced by the migrants in the destination island. For example, the similar individuals could be identified using a crowding operator [59] and replaced. Another approach could be to replace the worst individual in the population.

Migration interval is the time between migrations which essentially characterizes the communication intensiveness of the model. A high frequency of migration will result in rapid spread of information.

The migration frequency directly controls the duration of exploration and exploitation.

Number of migrants is an important parameter which determines how much an island controls its neighbors. The exploitation could be intensified By transferring large number of individuals and vice versa.

Migration topology: Unidirectional rings, grid graphs, hypercubes etc. are some of the many topologies which have been considered for island models [115]. The choice of topology has an impact on the extent to which the information is propagated to the different islands [211]. [135] discusses an interesting idea of adaptively changing the number of islands depending upon the observed improvement. If there is no improvement, the number of islands is doubled and if there is improvement the number of islands is reduced. They observed considerable speedups in many cases i.e. different topologies.

In [134], it has been experimentally shown that an appropriate choice of these parameters is very important for the islands to successfully avoid local optima on the one hand and also maintain diversity among themselves on the other hand. They also studied the robustness of migration intervals across different topologies. Even though rigorous experiments have thrown light over the underpinnings of the island models, the understanding is still limited and it is being explored vigorously [115].

Cellular EAs

Cellular EAs, also known as fine-grained models and are connected in a fixed topology predominantly using a ring or a grid topology [8]. Each island (or cell) in the cellular EA is a just an individual and it mates only with its neighbors, every generation. Therefore, the migration interval is

just 1. Due to such characteristics they are aptly used with parallel hardware like FPGAs (field-programmable gate arrays). Cellular EAs have a lot of similarity to cellular automata.

2.6 Genetic Programming

Genetic Programming (GP) [126] is an evolutionary computation method, which is inspired by the Darwinian evolution observed in nature. Evolutionary computation algorithms are optimization methods which use a population of candidate solutions. They simulate the concept of "the survival of the fittest" over the candidate solutions which breed with each other and generate solutions to the optimization problem. In the case of GP, these candidates are automatically generated programs. GP has been applied to numerous applications [19, 146] including the job shop scheduling problem.

Representation

Individuals in a GP population need not have a fixed length. They are constructed from a terminal set and a function set. For the commonly used tree-based representation of GP, the terminal set consists of symbols which form the leaves of the tree. An example of the tree-based genetic program is given in the Figure 2.3. The terminal set is $\{X, Y, \mathbb{R}^+\}$ and the function set is $\{+, -, *, \div, cos\}$.

Depending on the application, the terminal set and the function set could use complex representations of the problem. For example, for the job shop scheduling problem one possible terminal set could be (as used in [161]) $\{RJ, RO, RT, PR, W, DD, RM, \#\}$ which denotes operation ready time, number of remaining operations, work remaining of job, operation processing time, weight, due date, machine ready time and constant respectively.

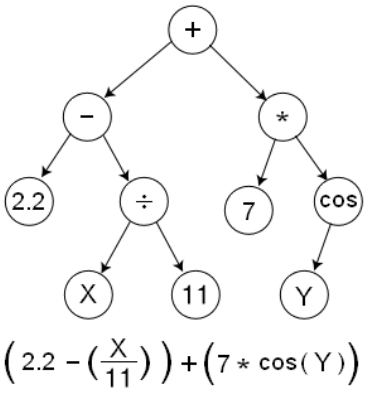


Figure 2.3: Example of a genetic program [1]

Inititalization

Evolutionary computation methods start their evolution process with an initial set of individuals. Various initialization methods have been used with GP. Two of the popular approaches [126] are *full* and *grow*. In these methods, the depth of the tree, largest number of edges needed to reach the leaf node, is restricted. In the case of full approach, full trees are generated with terminals located at the leaf nodes. First the nodes are selected using the function set only and when the tree is grown till the leaf node, the members of the terminal set are used. The growth approach does not require the genetic programs to have all its terminals to be at maximum depth. Therefore, the nodes are randomly picked from both the terminal set and the function set. In *ramped half-and-half* [126] half of the population is generated using full approach and the other half using the grow approach.

Evaluation

The fitness of the evolved program must be determined for the selection step under the evolutionary process. Fitness is also referred to as the objective function. In the case of multiple objectives, the fitness could be a

vector instead of a single numerical value. In the case of tree representation of genetic programs, the evaluation is done by tree traversal.

In the case of job shop scheduling problem, a genetic program is applied as a priority-based dispatching rules on the training instances which result in different schedules. The quality of these schedules is then evaluated depending on the objectives. For example, if minimization of the tardiness is the desired objective of the schedule, then the mean of total tardiness across all the training instances, which results from a particular schedule; which in turn is generated by the GP individual is its fitness. In case of static job shop scheduling, the training instances are basically a dataset giving details about a jobs and machines.

In the case of dynamic job shop scheduling, a discrete-event stochastic simulation [34] is used for generating jobs. The whole fitness evaluation is done on a job shop simulation environment. It should be noted that fitness evaluation in the case of job shop scheduling is a time consuming process. Many works have contributed to this problem of expensive optimization [226].

Selection

After the evaluation of genetic programs, the assigned fitness values are used to select individuals for breeding. This is similar to the Darwinian principle of evolution where fit individuals are more likely to survive and pass their traits to the offsprings (children). Roulette wheel selection method and the tournament selection are the two popular methods [18] used for selection. In roulette wheel selection method (also known as fitness proportionate selection), fitness value is used to assign probability values to individuals. Thus individuals with higher fitness values have a higher 'chance' of selection. This could be a problem sometimes when individuals with poor fitness values might have very less chance for selection. In tournament selection, a sample of individuals is selected randomly from the whole population. The fittest individual from this sample

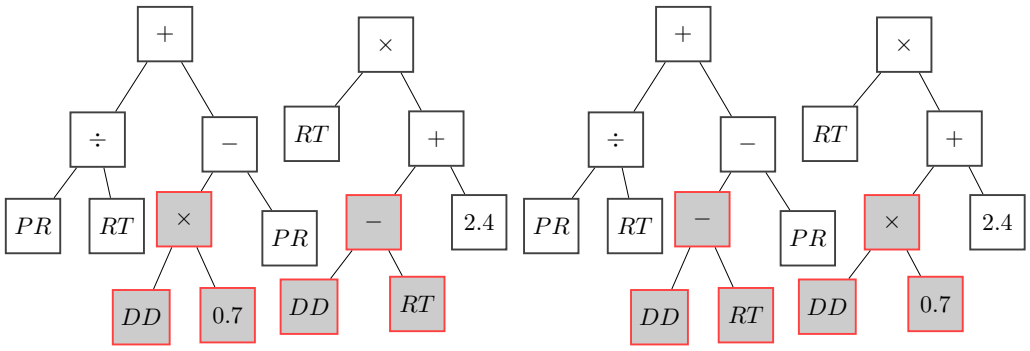


Figure 2.4: Example of a crossover operation in genetic programming. left to right: Parent A, Parent B, Child A, Child B

is then selected and the rest of the individuals are sent back to the population. Tournament selection is more frequently used with genetic programming [126].

Genetic Operators

Crossover and mutation are the two main genetic operators used in evolutionary computation. To perform crossover (Fig. 2.4), in a genetic programming representation, two individuals from the population are selected, and a random node is identified as the crossover point from each of them. This is known as subtree crossover. In the Fig. 2.4, Parent A and Parent B are selected for crossover. The two individuals exchange the selected subtrees to create new Child A and Child B.

In the case of mutation only one parent is needed to produce a new child. A random node is chosen from the selected parent and the subtree from that node is removed. Then a new subtree is grown at that node. This is known as subtree mutation. An example of mutation is shown in the Fig. 2.5.

Crossover rate and mutation rate are the parameters which decide the probability of crossover and mutation respectively, for each individual in every generation.

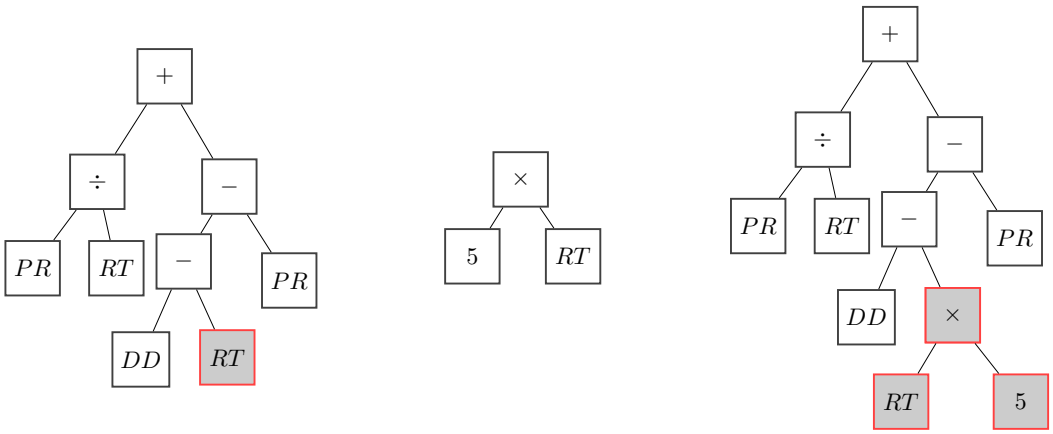


Figure 2.5: Example of a mutation operation in genetic programming.left to right: Parent, Replacement, Child.

Basic GP Algorithm

We present a basic genetic programming algorithm towards optimization (minimization) of a function in Algorithm 1. p^*, the best performing individual GP found at the end of the evolutionary algorithm is returned.

In line 1 of the algorithm, the initialization is done to create a population $\mathcal{P}$ of size n. The best solution p^* is set to $\emptyset$ in line 3 and its fitness is initialized to 0 (line 4). After initialization the evaluation, selection, mutation and crossover are applied on the population. In lines 6-8, each individual in the population is evaluated. In lines 9-14, each individual in the population is compared with the fitness of the current best individual. If a new individual with better fitness is identified then it is initialized as the new current best. In lines 15-20, the selection, mutation and crossover operators are applied to produce the next population P_+. The size of the new population is also n.

The whole process of evaluation, selection, application of genetic operators is repeated for every generation till $MaxGeneration$ are completed. Finally, the best individual obtained at the completion of $MaxGeneration$ generations is returned as the solution.

Algorithm 1: Genetic Programming Algorithm

Initialization. Using the available functions and terminals generate a random population $\mathcal{P}$ of size n, $\mathcal{P} \leftarrow \{p_1, p_2, \ldots, p_n\}$
$gen \leftarrow 0$
$p^* \leftarrow \emptyset$, (best solution)
$fitness(p^*) \leftarrow \infty$
while $gen < MaxGenerations$ **do**
  for $i = 1 \ldots n$ **do**
    Evaluate and assign fitness to p_i
  end
  for $i = 1 \ldots n$ **do**
    if $fitness(p_i) < fitness(p^*)$ **then**
      $p^* \leftarrow p_i$
      $fitness(p^*) \leftarrow fitness(p_i)$
    end
  end
  $\mathcal{P}_+ \leftarrow \emptyset$
  while $|\mathcal{P}_+| < n$ **do**
    Apply **Mutation**,**Crossover**, **Elitism** operators to generate new individual p using selected individuals using **selection** operator on $\mathcal{P}$
    $\mathcal{P}_+ \leftarrow \{\mathcal{P}_+, p\}$
  end
  $\mathcal{P} \leftarrow \mathcal{P}_+$
end
Return, p^*

2.6.1 Multi-objective GP (MOGP)

Multi-objective optimization deals with optimization problems which have many conflicting objectives. Many practical decision making problems in

the real-world require more than one criterion to be satisfied.

One of the ways to perform multi-objective optimization is to do a linear combination of the objectives to transform the problem into a single objective problem and then use the methods for single optimization to solve it. Deciding the weights for linear combination might require some domain knowledge about the problem which might be unavailable.

The other prominent method to handle multi-objective optimization is to determine a set of solutions instead of a single solution using the concept of Pareto dominance. According to this concept, a solution dominates another solution only if it is inferior to it one all the objectives. When there are many conflicting objectives this will result in a large number of non-dominating solutions. In order to assess the performance of this Pareto set of solutions many measures have been proposed [250] like, hypervolume indicator, generalized spread, inverted generational distance etc.

The following are two of the popular evolutionary approaches for multi-objective optimization problems which also work well for genetic programming. Both employ elitist strategies.

- Non-dominated sorting genetic algorithm II (NSGA-II) [59]: is based on the non-dominated sorting approach and the crowding distance operator which are utilized to rank the population of candidate solutions. The best solutions are selected based on their rank and the genetic operators are employed to generate the next generation of population.

- Strength Pareto evolution algorithms (SPEA2) [251] utilizes an archive of solutions which is updated every generation. The fitness of the individual is a combination of its strength and area density. Strength is determined by the number of individuals in the population and the archive which this individual dominates.

Another popular MOEA technique in the literature is MOEA/D [171].

NSGA-III [58] is popular for solving problems with more than three conflicting objectives, which is typically named as *many-objective optimization*.

2.6.2 Genetic Programming based Hyper-heuristics (GPHH)

Genetic programming based hyper-heuristics (GP-HH) has become prominent [41] because of the ability of GP to represent and evolve complex programs or rules. The work by Bolte and Thonemann [30] is one of the first in using genetic programming to evolve new heuristics for quadratic assignment problem. Their results showed that their method could find near optimal solutions for the quadratic assignment problem outperforming the existing simulated annealing algorithms. Using a local search algorithm for satisfiability problem, Fukunaga [73] used a genetic programming representation. Other prominent applications of genetic programming based hyper-heuristics for combinatorial optimization problem are in bin packing [41], traveling salesman problem [118] and time tabling heuristics [20].

2.7 Related Work to Job Shop Scheduling

In this section, we discuss the large number of solution approaches for job shop scheduling problems in the literature.

2.7.1 Solution Approaches for Static JSS

There are a large number of different techniques for scheduling in job shops like disjunctive programming, constraint programming, shifting bottleneck [182], branch and bound [132] etc. We discuss some of them below.

Johnson's Rule

Johnson's rule [114] is a popular method in operations research for scheduling. It is a method considered for a basic environment in the job shop

consisting of only two machines. The procedure is described below. The following are the preconditions required by this method. The method is presented in the Algorithm 2.

- The processing times are constant.
- The processing time of jobs must be mutually exclusive of the job sequence.
- Each job is first processed on the first machine and then on the second.
- All jobs have equal priority.

Algorithm 2: Johnson's rule for 2-machine shop

1 List the jobs and their times at each work center.
2 Select the job with the shortest processing time. If this processing time corresponds to Machine 1, then schedule the job first else schedule it last.(Tie break is arbitrarily.)
3 Eliminate the shortest job from further consideration.
4 Repeat steps 2 and 3, working towards the center of the job schedule until all jobs have been scheduled.

In the Table 2.2, the description of the jobs used in our example is presented. There are five jobs denoted as A, B, C, D and E. Their processing times on the two machines are given. We illustrate the application of this rule using lines 2 and 3 of the Algorithm 2 to generate the complete schedule.

The shortest processing time corresponds to Job B (1.4 hours) and since it is associated with Machine 2, it is scheduled the last.

$$? \rightarrow ? \rightarrow ? \rightarrow ? \rightarrow B$$

Table 2.2: Example: Johnson's rule

Job	Machine 1	Machine 2
A	2.9	4.1
B	4.6	1.4
C	2.1	5.1
D	5.7	4.1
E	3.0	2.7

The next shortest processing time corresponds to Job C (2.1 hours) which is scheduled first as it is associated with Machine 1. Job C is eliminated from further consideration.

$$C \rightarrow ? \rightarrow ? \rightarrow ? \rightarrow B$$

The next smallest processing time after that is for Job E (2.7 hours) in Machine 2. This is scheduled from the last before B. Eliminate Job E from further consideration.

$$C \rightarrow ? \rightarrow ? \rightarrow E \rightarrow B$$

Similarly, Job A is scheduled from the first. Job A is eliminated from further consideration.

$$C \rightarrow A \rightarrow ? \rightarrow E \rightarrow B$$

Job D which is the only job left is then scheduled.

$$C \rightarrow A \rightarrow D \rightarrow E \rightarrow B$$

So, the jobs must be processed in the order

$$C \rightarrow A \rightarrow D \rightarrow E \rightarrow B$$

and must be processed in the same order on both the machines.

Branch and bound

Branch and bound [132] is an exact method for combinatorial optimization problems as it determines optimal solutions. It is an enumerative search algorithm which systematically enumerates candidate solutions by means of state space search. A branching rule partitions the set of candidate solutions into subsets. Then it is checked against the upper and lower estimated bounds of the optimal solution and is discarded if it is not better than the current best. The efficiency of algorithm depends on the estimation of these bounds. Branch and bound has been applied to many scheduling problems [186]. It is computationally expensive for large scheduling problems.

Lagrangian approximation

Lagrangian relaxation [91] method approximates a difficult optimization problem into a simpler problem and then solves that to determine an approximate solution to the original problem. One approach to use Lagrangian relaxation is to model the job shop scheduling problem as mixed integer linear program then apply Lagrangian relaxation toward solving this problem. This method is also computationally expensive like branch and bound for scheduling problems [186]. Moreover, Lagrangian relaxation could be combined with branch and bound as well.

Meta-heuristic approaches

Finding optimal solutions using exact approaches [75] for JSS problems is hard, therefore meta-heuristic approaches have been employed for static JSS problems. Meta-heuristics do not guarantee optimality of solutions. Local search method [230], Tabu search [60], simulated annealing [232], guided local search [236], etc., are some of the prominent method in this category. These methods start with a solution and rely on neighborhood search to iteratively move to the next solution. Basically they require three

entities namely search space, cost function and neighborhood relation. Search space is related to the combinatorial optimization e.g. in the case of scheduling permutation of tasks on the different machines forms the search space. The neighborhood relation defines the neighborhood of a candidate solutions (a local modification which leads to the neighbor) and the cost function is used to assess the quality of a solution. [232] presents a simulated annealing approach for job shop scheduling and outperform the existing heuristics while considering makespan as the scheduling objective. [60] presents a Tabu search method for JSS problem which perform similar to the existing heuristics and other iterative methods.

Evolutionary algorithms have been successfully applied to JSS problems. In particular, GA has been a very popular meta-heuristic for scheduling problems [27]. [27] presents one of the initial applications of GA for job shop scheduling in manufacturing industry. Many hybrid algorithms combining GA and other meta-heuristics have also been proposed. For example, [103] presents a multi-objective local search based genetic algorithm for flow shop scheduling with promising results. [48] presents a survey of many such hybrid approaches.

Particle swarm optimization methods have also been explored to solve scheduling problems. For example, [143] presents a PSO based memetic algorithm to for flow shop scheduling to minimize makespan. Through their experiments they demonstrated the superiority of their algorithms in terms of search ability and robustness. [208] presents a multi-objective PSO for job shop scheduling. In their study they modified the particle position representation, particle movement, and particle velocity and through this modification achieved better results. [99] presents a hybrid algorithm using Tabu search and ant colony optimization to obtain competitive results for job shop scheduling. They even find the best known result on some of the problems. [29] develops an ant colony optimization algorithm for open shop scheduling problems and improves upon the best known results for more than 50% of the problems considered. Recently, [159] develops PSO

based hyper-heuristic (PSOHH) approach for DJSS problems. They develop new representations for PSO which are flexible enough to represent diverse dispatching rules for DJSS problems. They compare their method with GPHH approaches (from [166]) and show the effectiveness of PSOHH.

2.7.2 Solution Approaches for Dynamic JSS

Dispatching Rules for JSS

Dispatching rules are heuristics which assign priority to every operation queued on a machine. We explain this in Algorithm 3. In lines 1-3, for each operation queued on a machine in the job shop, priority values are assigned using the dispatching rule. The operations are then sorted based on the assigned priority values. This forms the sequence of operations for processing on the machine. For a dynamic JSS problem, when a new job arrives, the queue on the machine changes and the sequencing has to be done all over again. Compared to a meta-heuristic approach or an exact algorithm the computational cost for assigning priority values is very low (line 2). Therefore, for DJSS problems dispatching rules are the preferred method.

Algorithm 3: Sequencing using Dispatching Rule

Input: Dispatching rule, DR. Operations queued on a machine $\mathcal{M}_a$, $\mathcal{Q}_{\mathcal{M}_a}$

Output: Sequence of operations to be processed $\mathcal{S}_{\mathcal{M}_a}$

1 **for** *each operation* $o \in \mathcal{Q}_{\mathcal{M}_a}$ **do**

2 | assign priority value to o using DR

3 **end**

4 $\mathcal{S}_{\mathcal{M}_a}$ = Sequence generated by sorting the operations based in their assigned priority values.

5 Return, $\mathcal{S}_{\mathcal{M}_a}$

A large number of dispatching rules have been proposed in the literature. We can classify them into the following categories.

- *Simple priority rules* [172, 203] are the rules which are mostly based on the characteristics of the jobs. A list of such rules is presented in Table 2.3.
- *Combination of DRs [203]*: Depending on the prevailing shop floor characteristics linear combination of dispatching rules are employed for scheduling.
- *Composite dispatching rules [111]* combine a number of simple priority rules, not necessarily in a linear way. [225] evolve composite dispatching rules for flow shop scheduling using genetic programming and show that they outperform the composite rules developed through human expertise.

Table 2.3: List of Dispatching rules

Dispatching Rule	Description
SPT	Shortest processing time
LPT	Longest processing time
ECT	Earliest completion time
STPT	Shortest total processing time
LTPT	Longest total processing time
FCFS	First come first serve
EDD	Earliest due date first
LTWR	Least total work remaining
MTWR	Most total work remaining

Many other classifications of dispatching rules have been proposed for dispatching rules e.g. Pinedo [182] classifies the dispatching rules as *static* and *dynamic* rules [46]. Another classification could be done on the basis

of the information which the dispatching rule captures. If the dispatching rule relies only on the local information of the jobs then it is called a local rule. On the other hand if the dispatching rule captures information from the characteristics of the shop as a whole, then it is called a global rule. [172] provides a comprehensive survey of dispatching rules which are widely used along with their classifications.

2.7.3 GPHH for scheduling

Genetic program representation of dispatching rules have been used in many works.

Single machine scheduling

GP has been used to evolve priority tree based dispatching rules for static single machine scheduling problem in [62] with minimizing the tardiness as the objective. They have reported that their dispatching rules perform better than the traditional dispatching rules. Similarly in [107] dispatching rules have been used for parallel machines for both static and dynamic scheduling problems. Some other prominent examples of application of genetic programming based hyper-heuristics in single machine scheduling from the literature are [76, 108, 247]. [76] use GPHH to solve simple (polynomial time) as well complex instances (NP-hard) of job shop scheduling problems. [108] use GPHH for solving single machine scheduling problems with non-zero release dates to minimize total weighted tardiness.

Job shop scheduling

Miyashita [153] have used genetic programming to evolve dispatching rules in a multi-agent setting. They consider the machines as agents and evolve multi-agent dispatching rules. They consider three different models namely homogeneous, distinct and mixed agent models. The first

model tries to evolve a homogeneous dispatching rule for all agents, the second one considers a separate dispatching rule for each agent and the third one is a hybrid version of the two. Their motivation is to leverage GP to automatically evolve dispatching rules with same level of performance as observed with single machine scheduling. [161] performs a computational study of different representations of genetic programming for toward evolution of dispatching rules and also propose new representations which take into consideration the machine and system attributes. Through rigorous experimentation, they show that their proposed representations can evolve effective rules outperforming existing representations. [175] investigates ensemble combination schemes for GPHH approaches to DJSS problems. They investigate four ensemble approaches based on majority voting, linear combination, weighted linear combination and weighted majority voting and through experimentation found that linear combination schemes are better than other ensemble techniques. [149] highlights the importance of feature selection in GPHH for DJSS problems. They develop a niching based search framework for extracting a diverse set of dispatching rules while reducing the computational complexity of fitness evaluations by using surrogate models. Their experiments reveal that their approach is both efficient and effective in evolving rules. [101] tries to address the lack of global perspective dispatching rules by evolving "less-mypoic" dispatching rules for DJSS problems using GPHH. They achieve this by incorporating wider shop characteristics and their results show significant improvement in performance of the evolved rules. A comprehensive survey of application of GPHH for job shop scheduling is presented in [35, 158, 165].

Scenario-specific dispatching rules for complex environments

Now we review some of the works which consider evolving multiple dispatching rules for different scenarios arising in a complex shop. Pickardt et al. [177] combine GP and an EA heuristic to develop a two-stage hyper-

heuristic approach. In the first stage the GPHH approach evolves dispatching rules which is combined with existing rules from the literature using an evolutionary algorithm in the second stage. Essentially, with two stages they try to search two different search spaces. Finally, they obtain work-centre specific rules. This approach is applied to a semiconductor manufacturing industry to minimize total weighted tardiness. Their experiments show that the evolved rules using their approach substantially outperform the manually designed rules. Jakobovic et al. [107] use GPHH for both single machine and job shop environments. They considered the bottleneck and non-bottleneck states of the machines separately and evolved specific rules for them. They developed a GP3 method which consists of three rules, one each for the specific machines and a third rule acts as a binary classifier and separates the machines into the bottleneck and non-bottleneck classes. Their results showed that the approach using machine specific rules outperformed the one using a universal rule. Another work exploring similar idea by Miyashita [153] in which the author considered a mixed agent model for evolving dispatching rules specific for bottleneck and non-bottleneck machines. Recently, Heger et al. [88, 87, 89] have developed many works which consider dynamically switching of the dispatching rules for different shop scenarios. [89] considers multiple shop scenarios in a dynamic shop environment and apply Gaussian process regression to switch dispatching rules between EDD, MOD, 2PTPlusWinqPlusNPT [193]. They have considered the utilization and due date factor as the two features to determine the state of the shop. Their rule switching approach outperforms the single rule scheduling approaches. One limitation of their work is that they have considered only utilization and due date factor as features defining a shop scenario where as in a practical shop scenario there are many other factors which are important, e.g., the varying characteristics of the jobs arriving at the shop in terms of their processing times and number of operations per job.

Multi-objective JSS

Now we discuss some of the works in literature which consider more than one objective for scheduling. [21] have considered flowtime and tardiness as the two objectives for scheduling. [225] have developed new dispatching rules for minimizing mean makespan, mean tardiness and mean flowtime for static job shop scheduling. They used a scalar combination of the objectives for minimization. Ngyuyen et.al. [162] have used 5 objectives (mean flowtime, max flowtime, perentage of tardy jobs, maximum tardiness and mean tardiness) for generating dispatching rules using a genetic programming representation. It is of interest to note that 2 of the objectives namely maximum tardiness and maximum flowtime show a correlation among each other in the approximated Pareto front.

Recent energy aware scheduling has become very important across manufacturing industries [5]. Therefore, minimization of energy and its cost have been considered as one of the objectives for scheduling. [56] consider an energy aware scheduling approach for flow shop problem using a simulated annealing approach. They consider makespan and total energy consumption as the two objectives for minimization. Some of the recent works which consider energy aware scheduling in a multi-objective setting are [145, 167, 210].

2.7.4 Cooperative Co-evolution for JSS

Cooperative co-evolution algorithms (CCEA) are characterized by two or more interacting subspaces within a search space such that the fitness of an individual is evaluated based on its interactions with other individuals (subspace) [241]. A problem is decomposed into subproblems and solutions to the subproblems are then evolved. These are then combined together to form the final solution. The subpopulations belong to different "ecological niches" [185].

CCEA have been employed before for different DJSS problems. Park et

al. [173] propose a cooperative co-evolution based multi-level genetic programming approach to evolve ensembles of dispatching rules for DJSS. They had also developed a similar co-evolutionary approach to evolve ensembles of rules for static JSS problems [176]. In a related work [120], a co-evolutionary algorithm is proposed to integrate the planning and scheduling activities in flexible manufacturing systems. Nguyen et al. [160] co-evolve scheduling policies by considering due-date assignment and scheduling simultaneously.

2.7.5 Difference in Solution Approaches to Static and Dynamic JSS Problems

Due to the difference in the nature of static and dynamic problems their solution approaches were discussed separately in this section. Moreover, when uncertainty JSS problems is considered, the same arguments hold for employing different types of solution methodologies for the two kinds of JSS problems. In fact, the research works, as can be seen in this review, are characteristically different for the two types of problems and the accordingly the state-of-the-art methods drastically vary. For example, a local search method or a genetic algorithm approach (explained in previous subsections) are suitable for a static problem but infeasible for a dynamic JSS problem. On the other hand, a GPHH approach is much more suitable for a dynamic JSS problem. In the research works reviewed above, the benchmark approaches are also selected appropriately. For example, a mixed-integer linear programming approach [128] cannot be used to compare with a GPHH approach [158] when solving a DJSS problem as the former is simply not feasible for this problem.

2.7.6 Approaches for Dealing with Uncertainty in JSS

In an uncertain environment, two main steps for scheduling are schedule generation and schedule revision [50, 199]. Schedule generation could be

off-line where all available jobs are scheduled at once or it could be on-line which is done when needed.

To deal with uncertainty in the production environment, *reactive scheduling* is applied which modifies the existing schedule to adapt to the uncertainty. Most of the reactive scheduling methods use dispatching rules which use the local information to generate schedules. There are also works which allow the system to select dispatching rules dynamically e.g. [88]. Reactive scheduling is thus a short-term scheduling problem. The approaches under reactive scheduling update the current schedule to provide an immediate response to the unexpected event e.g. machine breakdown, change in job priority, etc. A number of approaches in literature have used mixed integer linear programming for reactive scheduling [6, 10, 183]. Sophisticated dispatching rules have been developed by researchers to tackle uncertainty. Chen et al. [47] propose a neural network approach to predict the dispatching rule to be used under a certain system state. Sabuncuoglu et al. [199] show that dispatching rules perform well in the face of stochastic disturbances. Gurel et al. [83] generate heuristics for minimizing the number of jobs delayed and the manufacturing cost. Jain et al. [105] examine different strategies for using complex heuristics for handling uncertainties in automated manufacturing systems. Jensen et al. [112] develop robust and flexible schedules with low makespan using genetic algorithms. They show that for machine breakdown, their method performs significantly better than other methods. Park et al. [174] investigate DJSS problem with the dynamic arrival of jobs while also considering machine breakdowns. They propose new terminals for GP and evaluates their effectiveness for DJSS problems.

As compared to reactive scheduling, *preventive scheduling* uses the historical data and forecasting methods to derive information about uncertainties. *Stochastic scheduling* is very commonly used approach in literature for preventive scheduling [66, 72, 188]. In two-stage stochastic programming first a single policy is developed and then a collection of recourse

actions to compensate the uncertainty are taken. In chance-constrained stochastic scheduling (e.g. [169]) the focus is on the reliability of the system i.e., the schedule meeting its quality in an uncertain environment. The reliability is expressed by imposing the requirement on probability of satisfying constraints. Simulation approaches for stochastic scheduling have been proposed, like [97] where the Monte Carlo sampling [97] is used to generate random instances of the uncertain parameters and developing schedules for each instance. *Robust Scheduling* is a preventive scheduling approach to minimizing the effects of disruption on the performance measure such that the realized schedule and predictive schedule have a low difference. An important concept is the distinction between schedule robustness and model robustness developed by [154] where a schedule (solution) is robust if it remains close to optimal for all scenarios and model robust if it is feasible for a very large number of scenarios. Some of the prominent works in this direction are [26, 110, 142]. Jia and Ierapetritou [113] proposed a multi-objective robust optimization model with model robustness, schedule robustness and solution robustness as the three objectives for scheduling under uncertainty. *Fuzzy programming method* is another preventive scheduling approach which could be used when the probabilistic models to describe the uncertain parameters are not available. In this approach, the uncertainty is represented using fuzzy set theory and interval arithmetic [14, 221, 239].

2.8 Parallel Hyper-heuristics and MOEAs

Multi-objective optimization algorithms are more complicated than their single objective counterparts because unlike single-objective case their solution consists of a Pareto set of solutions. This makes development of parallel MOEAs more difficult. Many of the parallel MOEAs are extensions of well known MOEAs like NSGAII, MOEA/D etc. In [67] a parallel version of NSGAII based on master slave model is presented. [36] develops a

cone separated NSGAII method which uses the island model to focus on specific areas of Pareto front. In [69] a multi-objective allocation problem for drinking water distribution is solved using an island model based NSGAII algorithm. In [68], a parallel version of MOEA/D is presented and tested over eight different problems to evolve Pareto fronts with higher quality. Xiao et al. [244] developed new island models specialized for multi-objective optimization.

More recently there have been many parallel hyper-heuristic approaches developed, particularly for optimization algorithms. For example, Bertels et al. [25] explained in their work about the importance of parallel evolutionary algorithms specifically for hyper-heuristic approaches when applied to solving SAT (Boolean Satisfiability) problems. Similarly, [86] develop a parallel Cartesian GP algorithm and show its strength by applying it on a n-bit parity digital circuit problem. [202] develops a parallel hyper-heuristic approach based on island model for frequency assignment problem in a GSM network. They provide more computational resources to islands which are more promising. [102] develop a parallel GPHH approach toward improving the performance of SAT solvers by automatically configuring them for a SAT problem. Similarly [214, 233, 246] are some other works which develop parallel hyper-heuristic approaches. We did not find any existing work which specifically uses GPHH for multi-objective DJSS using parallel MOEAs.

2.9 Summary of Literature Survey

We presented a literature review which covers the introduction to basic concepts followed by a more detailed review of recent works related to the objectives of this thesis. We summarize the key findings from the literature review below.

- Even though uncertainty in scheduling environment is omnipresent, most of the works do not consider its effects. For practical scheduling

problems, it is necessary to take uncertainty into account. The literature review demonstrates that GPHH based techniques to solve DJSS problems have a lot of potential and many researchers have contributed to this area in recent years. Existing research works have shown that the flexible representation of genetic programs have been able to incorporate complex characteristics of shop environment. To summarize, taking these findings into consideration, it is prudent to explore the ability of GPHH to solve DJSS problems under *uncertainty*.

- Multiple dispatching rules are required for different characteristics of the shop in a DJSS problem. Machine learning techniques have been employed for switching the dispatching rules subject to the varying characteristics. For practical scheduling problems in uncertain shop environments, considering the variation in the shop characteristics becomes very important. This requires investigation of machine learning techniques to identify the large number of different shop scenarios. Consequently, the ability of GPHH to automatically design rules could be leveraged to consider the different shop scenarios and design specific rules. Since GPHH is computationally expensive, machine learning techniques should be employed to carefully explore the range of possible shop characteristics, thus extending the scope of current GPHH framework.

- GPHH has been applied with good results for multi-objective DJSS problems even though it is a computationally expensive technique. Furthermore, the literature review shows that recent studies strongly advocate the use of parallel evolutionary techniques for hyper-heuristics approaches. For a practical DJSS problem, it is important to consider multiple objectives under an uncertain environment. These findings show that it is a good research direction to consider parallel evolutionary algorithms for GPHH. Furthermore, the literature

review also shows that the island model approaches defined by specific topologies and migration policies provide an opportunity for investigations under the purview of multi-objective DJSS.

With gaps identified by literature review and the conclusions derived from this survey, the following chapters will address them.

Chapter 3

Genetic Programming based Hyper-Heuristics for Dynamic Job Shop Scheduling under Uncertainty

3.1 Introduction

The literature review emphasized that in practical job shop scheduling problems, particularly dynamic scheduling problems, the shop parameters are uncertain. In spite of this, most of the existing works focus only on deterministic scenarios. It was highlighted that uncertainty in processing times, machine breakdowns, fluctuating arrival rates, modification of due dates, cancellation or modification of orders and uncertain arrival time of raw materials can significantly affect scheduling. In particular, focus is given to uncertainty in processing times. This could be attributed to the fact that variability in processing times has an impact on almost all scheduling objectives, e.g., makespan, mean flowtime, tardiness, etc.

The literature review also emphasized that GPHH approaches have

shown promise in evolving effective dispatching rules for the DJSS problems. Many recent works have successfully leveraged the characteristics of genetic programming like flexible representation and the ability to learn complex features. Therefore, it is a good research direction to further explore the aforementioned qualities of genetic programming to evolve dispatching rules for practical scheduling problems under uncertain shop conditions.

The choice of representation of the genetic programs is an important decision which depends on the problem. The different representations of GP could be achieved in many ways. One of them is through varying its structure e.g. linear tree, multi tree representations, etc. A more problem specific way to design representations of GP is by modifying the function sets and the terminal sets of the genetic programs. This is particularly relevant for DJSS problems which require the genetic programs to capture large number of shop and job features [158]. Our research direction is to explore the ability of GPHH for practical DJSS problems by taking uncertainty into account. Therefore, investigating GP representations by considering terminals which capture the uncertainty information is a good approach.

Apart from the flexible representations of genetic programming, another of its major strengths is its ability to learn complex scenarios without being provided with explicit features. This is aided by many novel evolutionary techniques like evolution strategy, cooperative co-evolution, etc., and [160] is a good recent example of such techniques under the purview of DJSS problems. From the perspective of our research direction, it is wise to investigate GPHH approaches for DJSS problems under uncertainty by exploiting this ability of genetic programming while investigating novel evolutionary approaches suited to our problem. For DJSS problems, the variability and uncertainty in the shop manifests into various complex scenarios. One of these is in the form of continuously varying bottleneck levels of the machines [124].

The idea of considering different bottleneck levels for evolving dispatching rules have been explored for static scheduling problems [108]. But for dynamic scheduling under uncertainty, there are complex interactions between the components of the shop which vary continuously. For example, the nature of jobs arriving at the shop vary due to which the difficulty of machine set-ups changes with time. Since issues with machine set-up, usually involving an operator is a common source of uncertainty [234], it is reflected as the uncertainty in processing times on different machines. This, in effect, manifests as varying bottleneck levels of the machines. The schedules which govern the sequence of the operations on these machines thus define the interactions between them. These interactions are more prominent and complex in dynamic job shop than in a static job shop. For systems with interacting components such as these, cooperative co-evolutionary techniques have been employed because of their strength in capturing the interactions during evolution. Therefore, it is motivating to explore the ability of GPHH helped by evolutionary techniques viz., co-operative co-evolution, to learn from different complex scenarios, like the one involving bottleneck and non-bottleneck machines.

We identified two parallel directions for investigating the ability of GPHH approach to evolve effective dispatching rules for DJSS problems under uncertainty. The first one is a *direct* approach which will try to leverage the strength of GP to consider different useful representations. The second direction hopes to leverage the potential of GP to learn from complex scenarios. Under the purview of DJSS problems, these complex scenarios are basically an effect of the uncertainty present in the shop. By considering such scenarios like varying bottleneck levels while evolution of dispatching rules GPHH could deal with the uncertainty in the shop, *indirectly*.

Furthermore, as mentioned earlier uncertainty in processing times has an effect on most of the scheduling objectives. Also, events like machine break downs and lack of skill in operators mostly impact the processing

times of the operations. To accomplish our research goals effectively it is a good idea to study the DJSS problems under the the scope of uncertainty in processing times.

3.1.1 Chapter Goals

The goal of this chapter is to develop new GPHH methods for DJSS under uncertain processing times. In order to achieve this goal, two parallel directions are considered in the form of two sub-goals. The first sub-goal investigates new representations for GPHH focusing on incorporating uncertainty information into the dispatching rules. Secondly, an approach to classify the bottleneck and non-bottleneck machines is presented which then employs cooperative co-evolution to evolve a pair of dispatching rules for each machine type. More specifically, the sub-goals are:

- To investigate new representations of GP by developing new terminals for incorporating uncertainty information into the evolved dispatching rules using GPHH.

- To develop a cooperative co-evolutionary approach for evolving a pair of dispatching rules for machines with varying bottleneck levels due to uncertainty in processing times.

3.1.2 Chapter Organization

The remaining chapter is organized as follows. In the next two sections, the methods related to the two sub-goals mentioned above are described. The Section 3.2 describes the methods which are developed for integrating uncertainty information into the dispatching rules. The Section 3.3 describes the co-evolutionary approach to evolve a pair rules for bottleneck and non-bottleneck machines. Finally, the chapter summary is presented in Section 3.5.

3.2 New Representations for GP : Toward Integrating Uncertainty Information into DRs

With the aim of developing new representations for GP, for incorporating uncertainty information into the dispatching rules, new terminals are investigated in this section. Since we focus on uncertainty in processing times, the proposed terminals must contain information related to processing time uncertainty. Firstly, we provide the details of the simulation model for uncertain processing times which will then enable us to develop the desired terminals. Then we develop three new methods which exploit the representation of GP while introducing extra changes to the terminals. These three methods are denoted by **EMA**, **ENT** and **EXP**.

3.2.1 Simulation Model with Uncertainty

It is assumed that the uncertainty in the processing time of a particular job is associated with a probability distribution. This probability distribution may change from one job to another. This is similar to many other studies in related work e.g. [95, 191, 231]. Rai et al. [191] consider scheduling in printing industry, where operator skills and job characteristics are main sources of variation in processing times. They state that even jobs with same work flow (route) could have different variation in processing times. So it is reasonable to assume dissimilar uncertainty distributions for different types of jobs. Akker et al. [231] considered processing times with a deterministic component and a random disturbance, which is identically distributed for each job.

We model uncertainty in processing times using the gamma distribution (Figure 3.1), which is widely used to model positive or skewed parameters, and has been used to model uncertainty e.g. [137]. It is a continuous probability distribution with two parameters, $shape$ ($\alpha \in \mathbb{R}^+$) and $scale$ ($\beta \in \mathbb{R}^+$). The deviation from the processing time ($p(o)$) is defined as δ,

such that

$$p'(o) = p(o) + p(o) \times \delta \tag{3.1}$$

where, δ follows a gamma distribution and different jobs are assigned gamma distribution parameters viz. $shape$ (α) and $scale$ (β) from a set. Henceforth, we call the term δ as the delay ratio. In all the experiments, α is set to 1. The choice of $\alpha = 1$ is motivated by the evidence from literature that the information contained in standard deviation of uncertain processing times is useful for scheduling [137, 181].

For a gamma distribution, the standard deviation is equal to $\alpha \times \beta^2$ and the mean is equal to $\alpha \times \beta$. If $\alpha = 1$, then the mean and the standard deviation of this gamma distribution are β and β^2 respectively. Essentially, the mean and standard deviation parameters are the information which could enable a scheduler (a dispatching rule) to generate good schedules for a DJSS problem which considers uncertain processing times. Since in this model, both the parameters are functions of β, this model is advantageous for the GPHH system as estimating the mean is easier than standard deviation. Moreover, note that when $\alpha = 1$, the distribution essentially becomes exponential, which has been frequently used to model uncertain processing times.

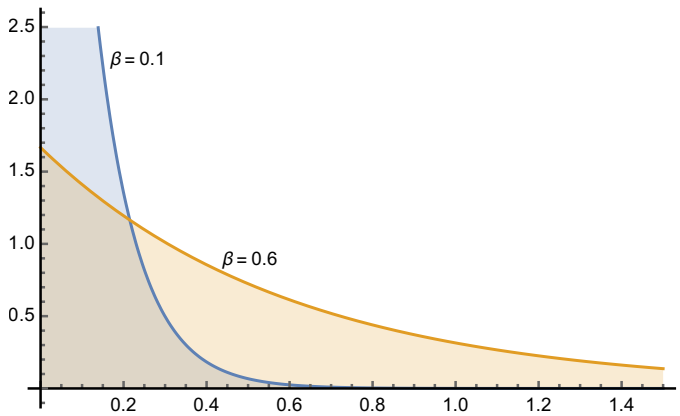


Figure 3.1: Gamma distributions

3.2.2 Exponential Moving Average (EMA) Terminal

The terminals of a genetic program representing a dispatching rule consist of job and shop parameters, like processing time (PT), due date (DD), remaining number of operations (RO), etc. In order to incorporate uncertainty into the genetic program's terminal, we must incorporate the information about uncertainty. According to our simulation model in Section 3.2.1, this information is associated with the value of delay ratio δ, and its distribution over a job is defined by the parameter β. Therefore, toward our goal to designing a terminal, we define the terminal EMA as the expected value of δ which is equal to β.

Since the information about the delay in processing of the operations in a job is known only after it has been completed, and since these operations are ordered, estimating the mean of δ is a repetitive calculation, repeating every time an operation is completed. A related technique in statistics is that of the moving average. The moving average is a rolling mean over different subsets of data. In a related method called the exponential moving average, the data points are weighted such that for the older data the weight keeps decreasing. It is presented in Equation 3.2, where $\bar{\delta}_i$ is the exponential moving average at step $i(> 0)$ and Y_i is the new data. κ is a constant smoothing factor.

$$\bar{\delta}_i = \kappa \times Y_i + (1 - \kappa) \times \bar{\delta}_{i-1} \tag{3.2}$$

Due to this methodology, we find that the exponential moving average is a suitable method for estimating the mean of delay ratio, $\bar{\delta}$. $\bar{\delta}$ is maintained for every job arriving at the shop. For a job j whenever a new operation $o_{j,i}$ is completed, the delay ratio $\delta_{j,i}$ is calculated thereupon using Equation 3.3, which comes from the simulation model (Equation 3.1). $p'(o_{j,i})$ is the realized processing time including the delay.

$$Y_i = \delta_{j,i} = p'(o_{j,i})/p(o_{j,i}) - 1 \tag{3.3}$$

Once the delay ratio $\delta_{j,i}$ is obtained, the mean is updated with this new data using Equation 3.4.

$$\bar{\delta}_{j,i} = \kappa \times \delta_{j,i} + (1 - \kappa) \times \bar{\delta}_{j,i-1} \tag{3.4}$$

According to literature [96], $\kappa \leq 0.3$ is considered good. We used $\kappa = 0.2$ in our experiments, and did not observe any difference in performance. We assigned $\bar{\delta}_{j,0} = \delta_{j,1}$ as the starting value of mean at the beginning as per the common initialization method used for calculating exponential moving average.

The exponential moving average, which is maintained for every job, captures the expected mean of delay ratio from the processing times of operations that are already completed. As earlier explained, due to the choice of our simulation model, estimating the mean of delay ratio is essentially equivalent to determining the standard deviation, which is considered as useful information for scheduling under uncertainty [181]. Therefore, we expect that the GPHH approach will evolve genetic programs which could utilize the information from the EMA terminal to create better schedules for DJSS problems under uncertain processing times.

3.2.3 Ex-post and Ex-ante Optimization

The next two methods are based on the concepts of *ex-ante* and *ex-post* optimization which we introduce now. Ex-ante means 'before the event' and its antonym ex-post means 'after the event'. In ex-post optimization, decision is made after the uncertainty is revealed where as it is the opposite case with the ex-ante optimization. These ideas originate in economic theory, first proposed by Myrdal [155] and subsequently considered in many areas including decision theory, game theory [4], bayesian statistics [123] and stochastic optimization problems [125, 224].

Kouvelis et al. [125] considered these ideas to define *robustness* of solutions to discrete optimization problems under uncertainty. For a decision

obtained through ex-ante the robustness is calculated by comparing the ex-ante decision against the optimal decision obtained through ex-post (perfect information is known). Belien et al. [22] consider ex-post optimization to determine the optimal strategy in fantasy sport games toward team selection and management. Tapiero et al. [224] propose an ex-post inventory control approach for the just-in-time control problem, using the information after the demand uncertainty is realized. They compare it with ex-ante optimization approach which determines the optimal inventory control policy using the demand forecasts.

The intuition behind ex-post optimization is to develop solutions which are optimal in an uncertain environment when the complete information about the uncertainty is revealed. In ex-ante optimization, the solutions are obtained before the unknown information is revealed. In theory, these two concepts are used to measure the robustness of solutions in uncertain environment [125]. In this research, we develop hyper-heuristic approaches and the methodology which they employ in solving a problem is to search in the heuristic space to find optimal (or near optimal) heuristics, which are then used to generate the final solution. Therefore, exploring the intuition of ex-ante and ex-post optimization for developing hyper-heuristic approach is not only novel but also a good research direction to tackle the effect of uncertainty in DJSS problems. Moreover, since we are considering GPHH problems, the flexible representation of GP aids us to develop methods which use cues from the concepts of ex-post and ex-ante optimization. Two GPHH methods inspired by each of these concepts are presented below.

Ex-post Training (EXP) Method

The ex-post optimization develops solutions for problems **after** the uncertainty information is revealed. The EXP method is a GPHH approach which evolved dispatching rules using the DJSS training instances for which the delay in processing times is already known. Consequently, EXP method

Table 3.1: Notation

	Description
ENT	Ex-ante training approach.
EXP	Ex-post training approach.
EMA	Using exponential moving average terminal. [116]
NAT	Standard GP approach (No additional terminals).

evolves dispatching rules which perform well on the instances for which the realized processing times are known. In order to use these dispatching rules on test instances i.e. practical DJSS instances with no information about uncertainty in processing times, we replace the processing time terminal (PT) with its estimate. The method to obtain this estimate is the same as for EMA method. We explain this below.

Suppose, $\mathcal{D}^{EXP}$ is a dispatching rule evolved using ex-post training approach, by using the realized processing times. When $\mathcal{D}^{EXP}$ is used to generate schedules for test instances, each processing time terminal PT^{exp} is replaced by the modified terminal PT^{*} such that:

$$PT^{*} = p_{j,i}(1 + \delta_{j,i}) \tag{3.5}$$

In order to determine the value of $\delta_{j,i}$ we consider the exponential moving average value of the delay ratio which was used with the EMA terminal.

Ex-ante Training (ENT) Method

Now, inspired by the ideas in ex-ante optimization, we propose the ENT method. For ex-ante optimization, the aim is to determine the optimal solution for the problem without actually knowing the complete information about uncertainty, but using a good anticipation about it [45]. In theory, this is expected to provide a solution which does consistently well over the different possible outcomes of the unknown.

Algorithm 4: EMA, EXP and ENT methods for GPHH

Input:

- $\mathcal{G}_\tau$, total number of generations.
- DJSS training instance with uncertain processing times, $\mathcal{P}$.
- DJSS training instance with realized processing times, $\mathcal{P}^r$.
- Algorithm selection parameter, $alg \in \{\textbf{EMA}, \textbf{ENT}, \textbf{EXP}\}$
- $\mathbb{T}$ terminal set for standard GPHH excluding terminal PT (processing time).

Output: Dispatching rule : Δ

Initialize population $\mathcal{S}$
Set $g \leftarrow 0$
if $alg =$ ***EMA*** **then**
    $\mathbb{T} = \{\mathbb{T}, PT, EMA\}$
else
    if $alg =$ ***ENT*** **then**
        $\mathbb{T} = \{\mathbb{T}, (PT + EMA \times PT)\}$
Use $\mathbb{T}$ as the terminal set for GPHH.
while $g \leq \mathcal{G}_\tau$ **do**
    $g \leftarrow g + 1$
    foreach *individual* $\mathcal{I} \in \mathcal{S}_1$ **do**
        if $alg =$ ***EXP*** **then**
            assign fitness to $\mathcal{I}$ using DJSS simulation on $\mathcal{P}^r$.
        else
            assign fitness to $\mathcal{I}$ using DJSS simulation on $\mathcal{P}$.
    end
    Evolve individuals in $\mathcal{S}$ using crossover and mutation.
end
$\Delta \leftarrow$ Best rule from $\mathcal{S}$
if $alg =$ ***EXP*** **then**
    Replace all terminals PT in Δ with $(PT + PT \times EMA)$
end
Return Δ

Table 3.2: **PT** terminals for ex-ante and ex-post approaches.

Dispatching rule	train	test
D^{EXP}	PT^{exp}	PT^{*}
D^{ENT}	PT^{*}	PT^{*}

The ENT method is a GPHH approach to evolving dispatching rules by using DJSS training instances along with an estimate of the delay in processing times. In order to include the information about this estimate, we again exploit the representation of the genetic program. Since, the information about the processing times is stored in the PT terminal, we use the modified terminal PT^{*}, as shown in Equation 3.5. The difference between the EXP and ENT methods is subtle but quite prominent. We describe this in more detail below.

The differences can be easily understood by considering how the two approaches differ in the use of terminals during training and testing. The different terminals used with the training and testing instances for the two approaches are shown in Table 3.2. For the dispatching rules evolved using ENT method, denoted by D^{ENT}, both the training and test runs consider the same terminals. But for the EXP method, in the evolved dispatching rules D^{EXP}, there is a clear distinction in the terminals. In contrast to the EMA method, D^{ENT} the processing time terminal is modified to incorporate the estimate of delay ratio instead of using an additional terminal.

In order to illustrate the difference between the three methods more clearly, we also present the Algorithm 4. Essentially, the three methods just described have been designed with the same objective, that is to integrate $\bar{\delta}_{j,i}$ (estimate) into the terminal of a genetic program but the modus operandi differs. In Algorithm 4, the algorithm selection parameter in the input is used to separate the three different procedures. The set $\mathbb{T}$ is defined as the terminal set used for evolving genetic programs, which exclude the PT terminal. For each of the methods $\mathbb{T}$ is modified. These mod-

ifications are explained through this algorithm below.

For EMA method, the terminal set is changed to include one more terminal EMA. For this method, $\bar{\delta}_{j,i}$ is considered as a new terminal by itself and forms a part of the evolved dispatching rules. So the terminal set becomes $\{\mathbb{T}, PT, EMA\}$ as in line 4. For ENT method, instead of using the processing time terminal PT, the terminal PT+EMA$\times$PT is used (line 7). So the uncertainty information $\bar{\delta}_{j,i}$ is integrated with the PT terminal. Now the EXP method is slightly tricky. Firstly, the DJSS training instances use ex-post information, i.e. the complete information about the delays in processing time is known. These training instances are denoted by $\mathcal{P}^r$ (line 13). Furthermore, after the completion of evolutionary steps in the EXP method, the PT terminal in the evolved rules is replaced by PT+EMA$\times$PT as in line 21.

3.2.4 Experiment Design

The aim of our experiments is two-fold. Firstly we compare our proposed methods with each other and also with standard GP which does not consider incorporation of uncertainty information. Secondly, we aim to analyze the generalization of our methods on DJSS under uncertain processing times. Generalization is the ability of our method to perform well on unseen data. So we test our methods on test configuration which are different to the training configurations.

We describe our experiment starting with the simulation configuration. We use a discrete event simulation system (see Jasima [92]) to generate problem instances of DJSS problems. Each problem instance is made of ten machines and eight operations per job [100]. The processing times of the operations are sampled uniformly from $[1, 49]$. This configuration has been considered in other studies [161]. The arrival of the jobs follows a Poisson process with a rate $\lambda = 0.85$ [116]. In order to evaluate the performance of our methods we need to choose a scheduling objective

which is affected by the uncertainty in processing times. Tardiness, total flowtime, makespan, etc., are some of such objectives. Total flowtime is an objective which is frequently used in literature and is completely based on the processing times [170]. We believe that the methods will be applicable to other objectives as well.

$$total\ flowtime = \sum_{i=1}^{N} (\mathcal{C}_i - \mathcal{R}_{\mathcal{J}_i})$$

where $\mathcal{R}_{\mathcal{J}_i}$ is the release time of job $\mathcal{J}_i$ and $\mathcal{C}_i$ is the completion time.

The jobs from $500 - 2000$ are considered for analysis, out of a 2500 total jobs arriving at the shop [161]. The delay factor $\delta_{j,i}$ follows a Gamma distribution. The different gamma distributions are obtained by varying its parameters, namely scale (β) and shape (α). The specific training and test configurations are described below.

Training and Test Configurations

We considered different settings of parameters for generating the problem instances for testing and training. By associating a particular level of uncertainty to a job, we characterize a job type. Then by varying the ratio of the job types we create problem instances. These problem instances enable us to simulate the variability in the shop. Basically, by using different ratios of jobs pertaining to the different job types, we are able to generate different arrival patterns in the shop. The parameter configurations are described below.

For **training**, we use two configurations by considering different number of job types. We use these two training configurations because they represent problem instances with contrasting variability in job shop. In the first configuration the number of job types is just two which is less but they are very different with respect to their uncertainty levels. In the second configuration, the number of job types is high which cover the range of uncertainty levels from low to high. Our aim is to explore the ability

Table 3.3: Test Configurations

Test-set	scale (β)	#job-type-ratio
I	$\{0.1, 0.6\}$	$1:1$
II	$\{0.1, 0.6\}$	$2:1$
III	$\{0.1, 0.6\}$	$3:1$
IV	$\{0.1, 0.3, 0.6\}$	$1:1:1$
V	$\{0.1, 0.3, 0.6, 0.8\}$	$1:1:1:1$
VI	$\{0.1, 0.2, 0.3, 0.4, 0.5\}$	$1:1:1:1:1$
VII	$\{0.1, 0.3, 0.6, 0.8, 1.2\}$	$1:1:1:1:1$

of GPHH to learn from the DJSS training instances with such contrasting characteristics.

- In the first training configuration, two job types are considered equi-proportion such that delay ratio follows the gamma distribution with $(\alpha = 1, \beta = 0.6)$ and $(\alpha = 1, \beta = 0.1)$ respectively.
- In the second training configuration, five job types are considered equi-proportion and the gamma distribution parameters are $\alpha = 1, \beta \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$

For **testing**, the configurations are give in Table 3.3. The last column #job-type-ratio tells the ratio between the number of jobs in each type. Note that the number of configurations is much higher than training, which is important to study the generalization of our methods. Furthermore, by skewing the ratio of job types in the test configurations I,II and III we would like to study the performance of our methods under varying levels of uncertainty.

The GP System

The genetic programming system uses a population of individuals which are evolved using the crossover and the mutation operators. The fitness

is assigned using a DJSS simulation [92]. After each round of evolution, individuals with high fitness values are selected. After completing all the generations of evolution, the best individual from the population is used as the final solution.

A list of the functions and terminals for GP is given in Table 3.4. The function if has 3 argument; it returns second argument if the first is greater than 0, and the third one otherwise. The protected division returns 1 when divided by 0. The population size is 1024, generation count is 50, maximal tree depth is 8, crossover rate is 0.85, mutation and elitism are 0.1 and 0.05 respectively. These set of terminals, functions and the rest of the parameters have been used earlier in similar experiments e.g. [161].

Table 3.4: Function and Terminal Sets for GP.

Function Set	Meaning
$+$	Addition
$-$	Subtraction
$*$	Multiplication
$/$	Protected Division
Max	Maximum
Min	Minimum
If	Conditional
Terminal Set	**Meaning**
DD	Due date of job
PT	Processing time of operation
RO	Remaining operations for job
RJ	Ready time of job
RT	Remaining processing time of job
RM	Ready time of machine
ERC	Ephemeral Random constant
EMA	Exponential moving average of delay ratio*

*(terminal not used with standard GP, NAT method)

3.2.5 Results and Discussions

The notation used for different methods is given in Table 3.1. Since we have used two training configurations, we suffix the method name with '2' or '5' depending on the #job-types used for training. E.g. ENT2 denotes that the ENT method is using the training instances with 2 job-types .

As seen in Table 3.1, we have 4 methods to compare. For each method, we consider 2 training configurations. This leads to 8 sets of results collected in Tables 3.5- 3.10 for a total of 7 test configurations. Each element of the table is represented as a triplet in the form of $[Win - Draw - Lose]$ which is a result of the Wilcoxon-rank-sum-test under a significance level of 0.05, used to compare the 30 test instances for each test configuration. For example, consider the column corresponding to the test configuration III and the row corresponding to ENT5 in Table 3.5. $[2 - 25 - 3]$ means that ENT2, when compared with ENT5, performs significantly better in 2 test instances, poorly in 3 test instances and shows similar performance in the remaining 25; among a total of 30 test problem instances. Thus we have done an exhaustive comparison across all methods with varied test and training configurations.

Firstly, we compare the proposed methods with standard GP approach (NAT). Secondly, we consider the generalization capabilities by comparing the evolved rules associated with different training configurations.

We present our observations for the comparisons with the existing methods. We use the first two plots in Figure 3.2 for this discussion. In each plot, a group of boxplots pertain to a single problem test instance. For example, in the first plot, with the caption [ENT2-EXP2-EMA2-NAT2] (Test Set-I), the boxplots are grouped 4 at a time. The caption also says that the test problem instances belong to test configuration I. In every group, the order of methods is the same as mentioned in the caption. The method(s) which performs the best are colored in yellow. So in the first group of boxplots, ENT2 outperforms all other methods. To get a complete result on all 30 problems refer the Tables 3.5 - 3.7.

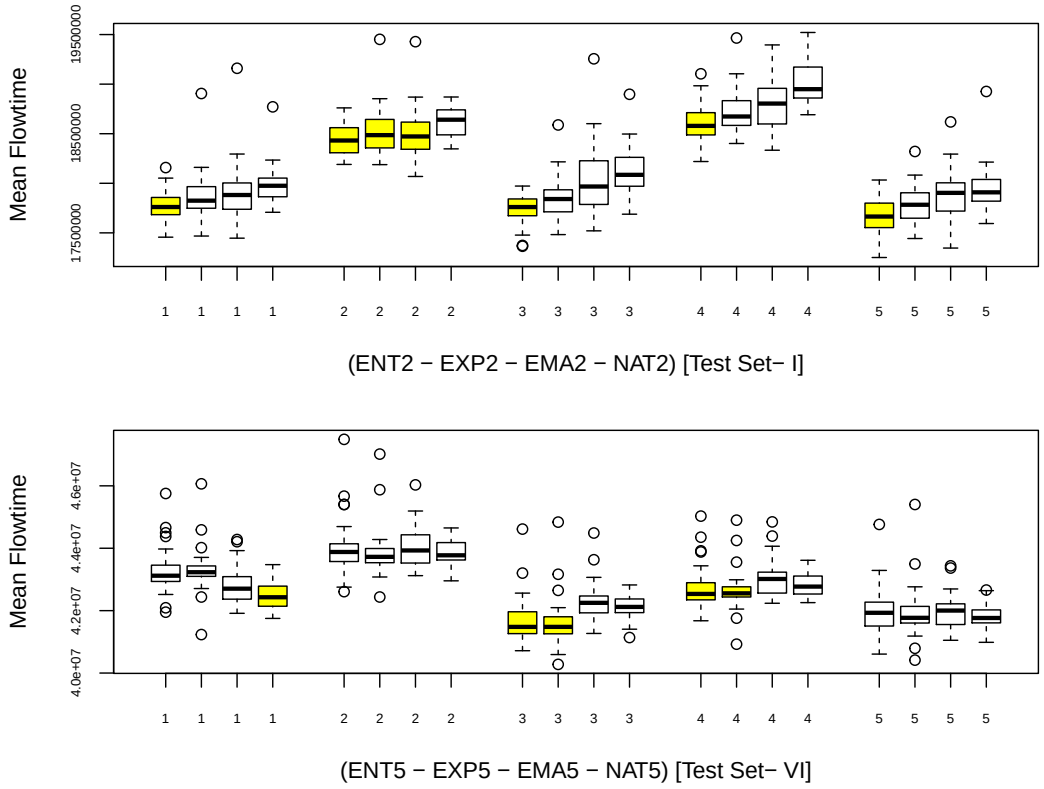


Figure 3.2: Boxplots: The order of boxplots is same as mentioned in the caption. The significantly better result is shown with a colored boxplot.

In particular, if we focus on the two-job-configuration, i.e. ENT2, the results are much better than the standard GP (NAT) and the exponential moving average terminal based approach (EMA2) for test configurations I-V. This can be inferred from the first five columns in the second and third rows of Table 3.5. The corresponding cells are shown in bold. Similarly ex-post training approach (EXP2) outperforms EMA2 and NAT2 for the same test configurations. The corresponding cells are shown in bold in Table 3.9. Therefore, both our proposed methods show significant improvement when compared against the existing methods.

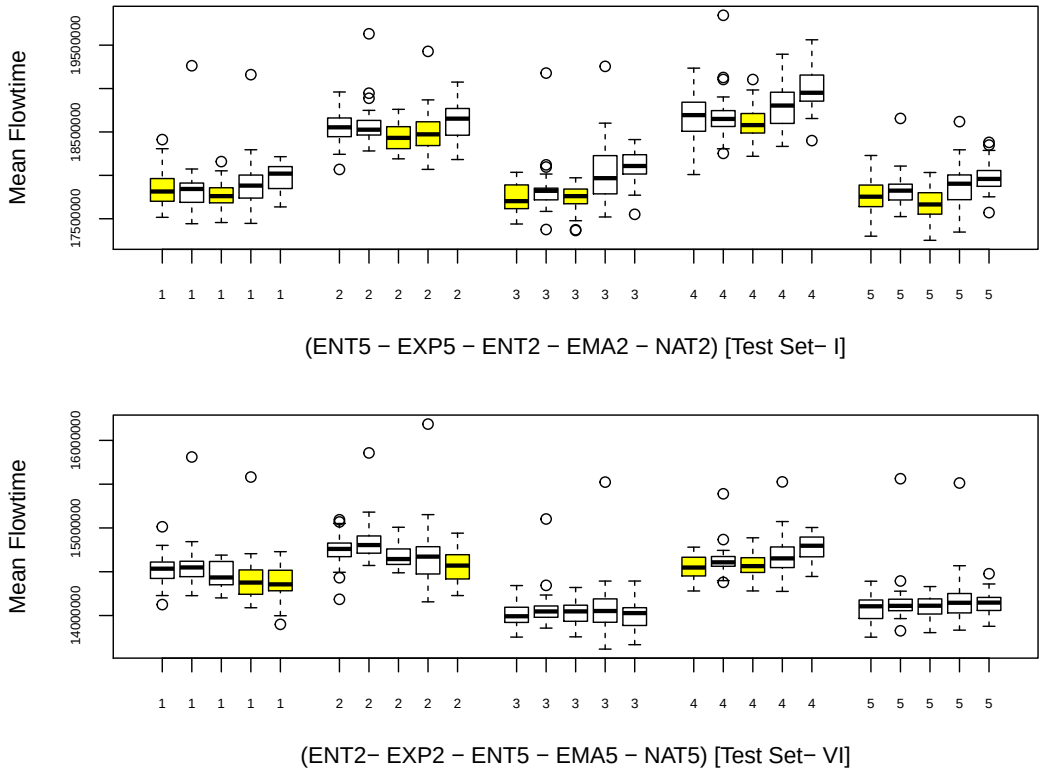


Figure 3.3: Boxplots: The order of boxplots is same as mentioned in the caption. The significantly better result is shown with a colored boxplot.

When the test configurations VI-VII are considered while comparing ENT5/EXP5, none of the methods is a clear winner. In fact, in some cases the proposed method under-performs, e.g., see the cells marked in blue in Table 3.6. This is understandable, as five-job-type configuration is associated with higher variability in the job shop and evolving good rules using the estimation of uncertainty parameter yields poor results, due to difficulty in accurate estimation.

EXP vs ENT

When we compare the EXP and ENT with each other, we observe that ENT2 is slightly better than EXP2 and ENT5 is slightly better than EXP5.

Table 3.5: ENT-2 (Ex-ante method, 2-job types training)

	I	II	III	IV	V	VI	VII
EXP-2	[13-17-0]	[5-25-0]	[2-28-0]	[8-22-0]	[4-26-0]	[5-25-0]	[0-30-0]
EMA-2	**[22-8-0]**	**[29-1-0]**	**[29-1-0]**	**[29-1-0]**	**[24-6-0]**	[0-29-1]	[6-17-7]
NAT-2	**[28-2-0]**	**[30-0-0]**	**[30-0-0]**	**[29-1-0]**	**[29-1-0]**	[3-24-3]	[11-13-6]
ENT-5	[12-18-0]	[2-28-0]	[2-25-3]	[3-27-0]	[8-22-0]	[6-24-0]	[6-24-0]
EXP-5	[19-11-0]	[9-21-0]	[1-29-0]	[10-20-0]	[25-5-0]	[11-19-0]	[6-24-0]
EMA-5	[27-3-0]	[29-1-0]	[30-0-0]	[29-1-0]	[29-1-0]	[2-19-9]	[15-12-3]
NAT-5	[27-3-0]	[30-0-0]	[29-1-0]	[29-1-0]	[30-0-0]	[4-22-4]	[13-12-5]

Table 3.6: ENT-5 (Ex-ante method, 5-job types training)

	I	II	III	IV	V	VI	VII
ENT-2	[0-18-12]	[0-28-2]	[3-25-2]	[0-27-3]	[0-22-8]	[0-24-6]	[0-24-6]
EXP-2	[1-29-0]	[2-28-0]	[7-23-0]	[3-27-0]	[0-30-0]	[4-25-1]	[0-30-0]
EMA-2	[12-17-1]	[28-2-0]	[27-3-0]	[25-5-0]	[13-17-0]	[1-24-5]	[1-12-17]
NAT-2	[25-5-0]	[30-0-0]	[29-1-0]	[27-3-0]	[23-7-0]	[2-21-7]	[7-11-12]
EXP-5	[2-28-0]	[4-26-0]	[6-24-0]	[0-29-1]	[3-27-0]	[1-29-0]	[0-30-0]
EMA-5	[25-5-0]	[29-1-0]	[29-1-0]	[28-2-0]	[26-4-0]	**[2-11-17]**	[11-14-5]
NAT-5	[25-5-0]	[29-1-0]	[29-1-0]	[28-2-0]	[25-5-0]	**[3-18-9]**	[9-12-9]

This can be observed in the first plot of Figure 3.3. And more clearly in #row-1 and #row-5 in Tables 3.5 & 3.6 respectively. The corresponding rows are shown in grey.

Till now we presented the observations of comparison among methods that use identical training configurations. In order to determine the generalization of our methods, we compare dispatching rules evolved using different training configurations.

- **two-job-type-configuration**

  Firstly, we consider the two-job-type-configuration which is associated with less variability in the job shop. The first plot in Figure 3.3

Table 3.7: EXP-5 (Ex-post method, 5-job types training)

	I	II	III	IV	V	VI	VII
ENT-2	[0-11-19]	[0-21-9]	[0-29-1]	[0-20-10]	[0-5-25]	[0-19-11]	[0-24-6]
EXP-2	[0-27-3]	[1-28-1]	[3-27-0]	[0-30-0]	[0-24-6]	[1-27-2]	[0-30-0]
EMA-2	[7-21-2]	[28-2-0]	[24-6-0]	[22-8-0]	[9-21-0]	[0-20-10]	[3-14-13]
NAT-2	[26-4-0]	[30-0-0]	[30-0-0]	[27-3-0]	[19-11-0]	[2-18-10]	[9-13-8]
ENT-5	[0-28-2]	[0-26-4]	[0-24-6]	[1-29-0]	[0-27-3]	[0-29-1]	[0-30-0]
EMA-5	[25-5-0]	[29-1-0]	[28-2-0]	[27-3-0]	[21-9-0]	[1-10-19]	[12-13-5]
NAT-5	[24-6-0]	[29-1-0]	[28-2-0]	[27-3-0]	[22-8-0]	[2-15-13]	[10-14-6]

gives an intuition to the comparison, where we compare the five methods against each other on test configuration I. If we compare EXP5/ENT5 with EXP2/ENT2 on test configurations I to V and consider the comparison of EMA5 with EMA2, we can conclude that the generalization characteristics of the methods ENT and EXP are better than EMA.

We explain this observation using some of the examples. Consider the cells marked in green in Tables 3.6, 3.7 & 3.8. The cells corresponding to test configuration I are [0-18-12], [0-27-3] and [0-10-20] for ENT5, EXP5 and EMA5 respectively. The generalization of EMA5 is the worst among the three methods in this example. The same trend could be observed in other columns too.

- **five-job-type-configuration**

  Refer the second plot in Figure 3.3. Now we try to compare the performance of ENT2/EXP2/EMA2/NAT2 against
  ENT5/EXP5/EMA5/NAT5 on test configurations VI-VII. Referring cells marked in Tables 3.5, 3.9 & 3.10, the generalization characteristics of ENT and EXP methods show high competitiveness.

Table 3.8: EMA-5 (Exponential moving avg. method, 5-job types training)

	I	II	III	IV	V	VI	VII
ENT-2	[0-3-27]	[0-1-29]	[0-0-30]	[0-1-29]	[0-1-29]	[9-19-2]	[3-12-15]
EXP-2	[0-4-26]	[0-2-28]	[0-4-26]	[0-2-28]	[0-3-27]	[18-11-1]	[3-15-12]
EMA-2	[0-10-20]	[0-17-13]	[0-24-6]	[0-15-15]	[0-7-23]	[9-21-0]	[0-4-26]
NAT-2	[0-30-0]	[2-28-0]	[8-22-0]	[0-30-0]	[0-28-2]	[3-27-0]	[0-18-12]
ENT-5	[0-5-25]	[0-1-29]	[0-1-29]	[0-2-28]	[0-4-26]	[17-11-2]	[5-14-11]
EXP-5	[0-5-25]	[0-1-29]	[0-2-28]	[0-3-27]	[0-9-21]	[19-10-1]	[5-13-12]
NAT-5	[0-30-0]	[2-28-0]	[1-29-0]	[1-29-0]	[0-30-0]	[3-27-0]	[0-23-7]

Table 3.9: EXP-2 (Ex-post method, 2-job types training)

	I	II	III	IV	V	VI	VII
ENT-2	[0-17-13]	[0-25-5]	[0-28-2]	[0-22-8]	[0-26-4]	[0-25-5]	[0-30-0]
EMA-2	**[12-17-1]**	**[28-2-0]**	**[20-10-0]**	**[22-8-0]**	**[17-13-0]**	[0-24-6]	[2-21-7]
NAT-2	**[27-3-0]**	**[30-0-0]**	**[28-2-0]**	**[27-3-0]**	**[26-4-0]**	[2-19-9]	[8-16-6]
ENT-5	[0-29-1]	[0-28-2]	[0-23-7]	[0-27-3]	[0-30-0]	[1-25-4]	[0-30-0]
EXP-5	[3-27-0]	[1-28-1]	[0-27-3]	[0-30-0]	[6-24-0]	[2-27-1]	[0-30-0]
EMA-5	[26-4-0]	[28-2-0]	[26-4-0]	[28-2-0]	[27-3-0]	[1-11-18]	[12-15-3]
NAT-5	[25-5-0]	[30-0-0]	[28-2-0]	[28-2-0]	[29-1-0]	[2-16-12]	[8-16-6]

Table 3.10: EMA-2 (Exponential moving avg. method, 2-job types training)

	I	II	III	IV	V	VI	VII
ENT-2	[0-8-22]	[0-1-29]	[0-1-29]	[0-1-29]	[0-6-24]	[1-29-0]	[7-17-6]
EXP-2	[1-17-12]	[0-2-28]	[0-10-20]	[0-8-22]	[0-13-17]	[6-24-0]	[7-21-2]
NAT-2	[24-6-0]	[17-13-0]	[18-12-0]	[16-14-0]	[12-18-0]	[2-27-1]	[11-19-0]
ENT-5	[1-17-12]	[0-2-28]	[0-3-27]	[0-5-25]	[0-17-13]	[5-24-1]	[17-12-1]
EXP-5	[2-21-7]	[0-2-28]	[0-6-24]	[0-8-22]	[0-21-9]	[10-20-0]	[13-14-3]
EMA-5	[20-10-0]	[13-17-0]	[6-24-0]	[15-15-0]	[23-7-0]	[0-21-9]	[26-4-0]
NAT-5	[23-7-0]	[19-11-0]	[13-17-0]	[19-11-0]	[22-8-0]	[0-28-2]	[12-18-0]

3.2.6 Analysis

We developed our methods for DJSS problems under uncertain processing times. Though we considered total flowtime as our scheduling objective in the experiments, it should be expected that other objectives which depend on processing times also show similar results. To summarize, we observed that for the scheduling objective of total flowtime, the performance of the different methods is approximately in the following order.

$$ENT > EXP >> EMA > NAT$$

We analyze our key results and try to gain more insights to our methods. Firstly, we show one of the best dispatching rules evolved using the EXP method and illustrate how the processing time terminals are changed at the time of using the rule to generate schedules on test cases. The Dispatching Rule shown below is the rule which was evolved during training. The terminals PT^{exp}, which are shown in bold are replaced by PT^{*} (refer Equation 3.5)

Dispatching Rule 3.1: EXP method - Training

(If (Max (If (+ PT^{exp} (If RO RM RM))(- (Max RJ RT) (Max 0.002 RM)) (If DD 0.59 0.47))(+ (- (* DD RO) (Max 0.002 RM))(Max RJ RT))) (* (/ (If (* DD RO) (* RO PT^{exp}) (If DD RJ 0.72)) (* (/ 0.78 PT^{exp}) (- 0.24 0.28))) (If (/ (/ DD RJ) (-RM PT^{exp})) (/ (If RM RT RM) (If RJ RJ RO)) (If RM RT RM))) (If (/ RJ RT) (+ PT^{exp} RO) (* DD RO)))

Secondly, we determine the frequency of terminals in 30 best rules evolved using each method. The histogram plot is shown in Figure 3.4. The histogram plot shows that there is a consistent pattern for the frequency of terminals among all methods. Since the maximum depth of the genetic programs is fixed, the introduction of terminal EMA reduces the number of other terminals in the genetic programs. Also it makes the

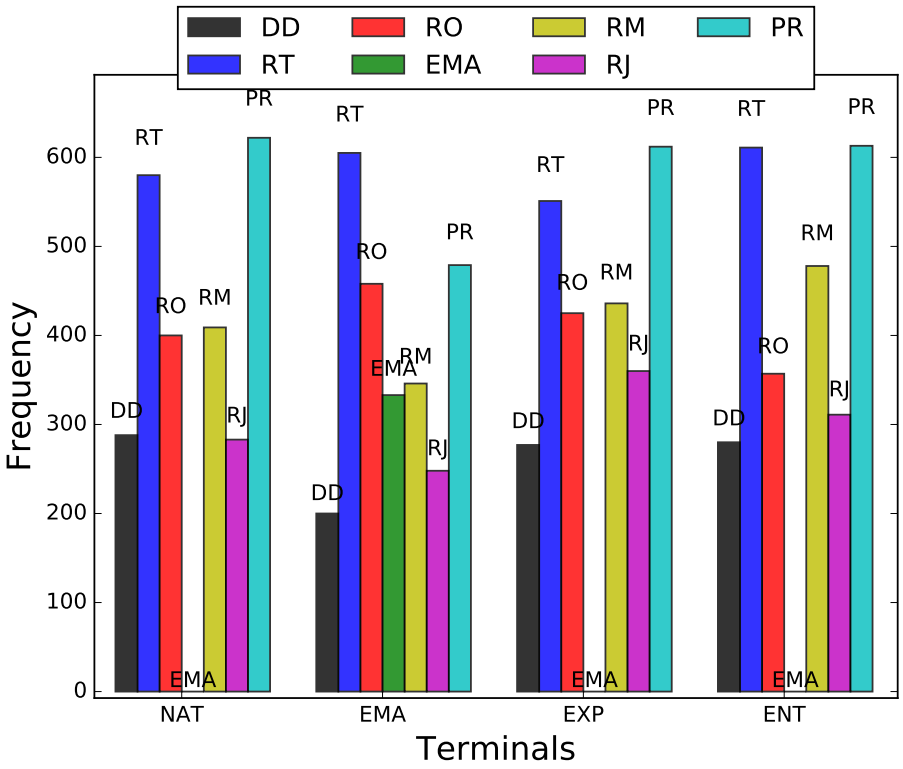


Figure 3.4: Histogram of Frequency of Terminals

search space more complex due to an extra terminal. These observations point to the reason for the ENT and EXP to outperform EMA.

Thirdly, we look at the performance of the evolved rules on the training instance. The box-plots for this comparison is shown in Figure. 3.5. The plot shows the performance of best rules evolved from the 30 independent runs. As expected, dispatching rules evolved using the EXP method perform the best on the training instance as it uses the realized processing time during evolution. We believe that EXP method suffers from high-variance problem and overfit to the *ex-post* DJSS training instances. The ENT method, on the other hand, considers the estimate of the delay ratio during training. Due to this reason the ENT is more competitive than EXP.

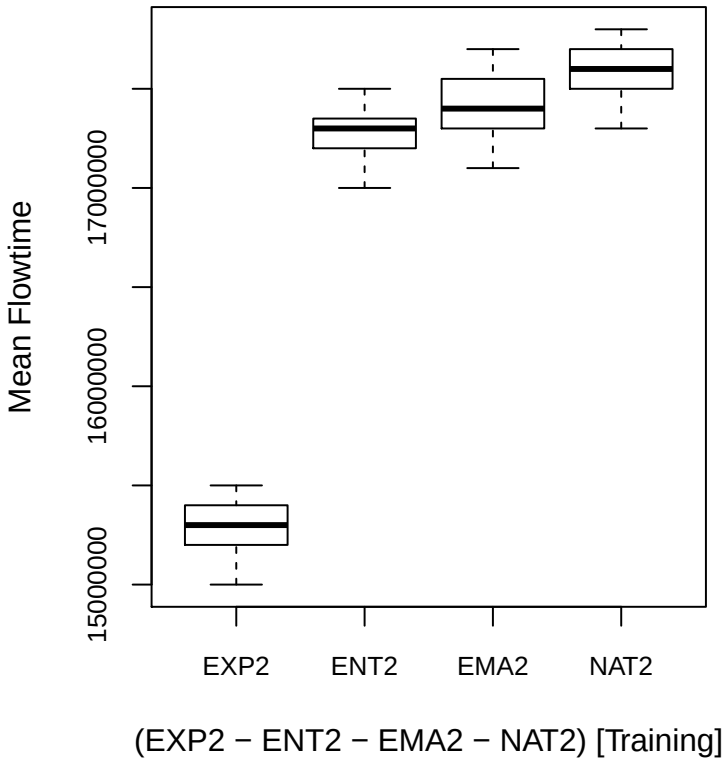


Figure 3.5: Boxplots for training instance

3.2.7 Section Summary

In this section, three methods which exploit the GP representations to incorporate uncertainty information of processing times into dispatching rules were presented. The results show that the proposed methods are successful in evolving effective dispatching rules for DJSS problems under uncertainty. The three methods namely, **EMA**, **ENT** and **EXP**, proposed different approaches to introduce changes to the terminal set of the genetic programs. All the methods performed better than the GP representation which does not consider uncertainty. Furthermore, the experimental results show that the **ENT** performs the best among the three methods. Also, both ENT and EXP methods, which are inspired by the ex-ante and ex-post optimization approaches respectively, perform *much* better than

EMA.

These methods used the assumption that the delays in the processing times of the operations of a job follow a probability distribution, which relates well to many existing scheduling problems. In the next section, we consider the delays in processing times which are a characteristic of the machines, which are detected in the form of varying bottlenecks. This is achieved through exploring the power of GPHH to learn from complex scenarios without the need for explicit features.

3.3 Cooperative Co-evolution of DRs for Bottleneck and Non-Bottleneck Machines

In the previous section, the approaches to integrating uncertainty information into dispatching rules were presented. The goal was to improve the scheduling performance by providing additional information to the dispatching rules. Another potential way of improving the scheduling performance under uncertain processing times is to evolve dispatching rules which are specific to shop scenarios which arise due to uncertainty. As already discussed, the different jobs, due to their inherent characteristics, are composed of operations whose delay in processing times may be pertaining to different probability distributions. Moreover, the machines are associated with different characteristics which vary in the profiles of break downs, complexity in set-ups, etc. In a dynamic shop with continuous arrival of jobs these characteristics change with time resulting in varying shop scenarios. Of particular interest to us, in this section, are the scenarios which manifest as the varying bottleneck characteristics of the machines, which has been an important topic for study [71, 106]. We explain this further with an example below.

Consider the scheduling problem in a print industry [189]. Setting up the required ink cartridges and getting the correct mix for the specific color

is an important and complex set-up requirement. This is specific for each print job. Also the printing is done across many printers to get the final document/product. The characteristic of the previous operation done on the printer has an effect on the time required for set-up and maintenance tasks. This leads to uncertainty in the processing times of the operations. Depending on the characteristics of the arriving jobs, workloads on different printers continuously vary. More specifically, in such job shop scheduling problems, the bottleneck levels of these machines continuously vary. Clearly, the bottlenecks manifest as a result of the uncertainty in processing times.

Coming back to our goal of using GPHH to evolve effecting dispatching rules under uncertainty, we want to exploit the ability of genetic programming to learn from complex scenarios. The two scenarios are associated with bottleneck and non-bottleneck machines. The first step should to identify these bottlenecks or in other words to classify machines into bottleneck and non-bottleneck. But firstly, we look at some of the closely related works in this direction from the literature.

Adams et al. [2] developed a shifting bottleneck procedure for job shop scheduling to minimize makespan. This work has been modified to solve varied classes of problems [48, 180]. Moreover, bottleneck identification has been shown to be a useful step in order to provide additional computational resources to optimize the sequencing at the bottleneck machine [249, 238]. Jakobovič et al. [107] propose a genetic programming based method for static job shop scheduling where they consider evolving separate rules for bottleneck and non-bottleneck machines. The machines are classified using a decision rule which is a genetic program with a different set of terminals.

Now, the work by Jakobović et al. [107] which proposes an adaptive scheduling heuristic for bottleneck and non-bottleneck machines in a static job shop scheduling problem is discussed in more details. Before that, we give a brief outline of the methods presented in this section. We use

some of the ideas for bottleneck identification from Jakobović et al. [107]. These method presented in their work needs some enhancements for it to work on DJSS problems, which we will describe using our proposed methods. Their method, which is named as GP3 and the standard GP approach are also used as benchmarks. We propose a new method (GP2-K) which uses unsupervised clustering of machines' states to classify the bottleneck and non-bottleneck machines are presented. If we look at the previous example, assuming that we are able to identify the bottleneck and non-bottleneck machines, we can try to evolve separate rules for each bottleneck scenario. But in the shop, the sequences generated by these rules interact with each other as the operations of different jobs span across the machines. Therefore, in order to evolve rules which are able to consider this interaction, a cooperative co-evolutionary (CoGP2-K) approach is proposed which aims to evolve dispatching rules for the two types of machines.

Jakobović et al. [107] propose an adaptive scheduling heuristic, where they evolve a pair of dispatching rules, one for the bottleneck machine and the other for non-bottleneck machine. In order to classify a machine into the two types, they use a third rule, decision rule, which uses a different set of terminals. These terminals are shown in Table 3.11.

Table 3.11: Terminal Set: Jakobović-GP3 (Decision rule)

Terminal	**Definition**
MTWK	Total processing time of all operations on a machine
MTWKr	Processing time of all remaining operations on a machine
MTWKav	Average duration of all operations on a machine.
MNOPr	Number of remaining operations on a machine.
MNOPw	Number of waiting operations on a machine.
MUTL	Machine Utilization.

Compared to static job shop scheduling, in DJSS problems under un-

certain processing times, the variation in the bottleneck characteristics of a machine is more prominent. Therefore, in particular, the machine utilization (MUTL) terminal should represent the current state of machine. In order to determine machine utilization, exponentially decreasing weights for older time periods are used, so as to emphasize more on the recent load on the machine.

GP2-K-means (GP2-K)

Although GP3 moves a step towards the scenario-dependent rule learning, it is difficult to evolve both the dispatching rules and the decision rule together. This is because the error made by the decision rule can potentially affect the dispatching rule learning, since the dispatching rule is applied to a wrong scenario. To address this issue, a new GP training process is proposed, which is called GP2-K. GP2-K separates the dispatching rule learning from the decision rule learning. Moreover, since bottleneck machines are associated with higher level of uncertainty and the non-bottleneck machines are associated with lower level of uncertainty, it is imperative that the training sets also capture this trait. Therefore, two DJSS training instances consist of two sets, one with high uncertainty and the other with low uncertainty. This requires two sub-populations, one for DR_l and the other for DR_h. In this way, it could be guaranteed that dispatching rule is consistently applied to the correct scenario, and thus its performance can be evaluated more accurately.

Note that the difference between the uncertainty levels of the two configurations should not be too high, as the goal is to evolve solutions which work under subtle variation in uncertainty levels, which is also more practical. If the difference in uncertainty levels were more, then the dispatching rules will be evolved for one type of machine with high workload and for another with comparatively much lower workload which is a rare occurrence. Normally, in DJSS problems where the bottleneck levels fluctuate, the difference in uncertainty levels of the bottleneck and non-bottleneck

Algorithm 5: GP2-K [Training]

Input:

- $\mathcal{G}_\tau$, total number of generations.
- DJSS training instance ($\mathcal{P}_t$).
  - Simulation parameters
  - $\mathcal{U}_l$ uncertainty configuration (low)
  - $\mathcal{U}_h$ uncertainty configuration (high)

Output: Pair of dispatching rules : $\{DR_l, DR_h\}$

1 Initialize subpopulations $\mathcal{S}_1, \mathcal{S}_2$
2 Set $g \leftarrow 0$
3 **while** $g \leq \mathcal{G}_\tau$ **do**
4 $g \leftarrow g + 1$
5 **foreach** *individual* $\mathcal{I} \in \mathcal{S}_1$ **do**
6 assign fitness to $\mathcal{I}$ using DJSS simulation with $\mathcal{U}_l$ config.
7 **end**
8 **foreach** *individual in* $\mathcal{S}_2$ **do**
9 assign fitness to $\mathcal{I}$ using DJSS simulation with $\mathcal{U}_h$ config.
10 **end**
11 Evolve individuals in $\mathcal{S}_1, \mathcal{S}_2$ using crossover and mutation.
12 **end**

machines is not very high. Therefore, if the variation were stark, both the bottleneck classification and the design of dispatching rules would be easier but not effective, The specific choice of the uncertainty levels is discussed further with our experiment design (Section 3.3.1).

The proposed training process is described in Algorithm 5. The two sub-populations are then evolved independently (lines 3-12). The pair of best evolved rules from sub-populations at the end of last generation is

the final output. Note that the newly proposed training process does not include the decision rule learning. During the test process, it is required to classify the current state to decide which dispatching rule to use. To this end, a clustering approach is proposed to replace the need of the decision rule.

Algorithm 6: K-means-clustering approach: GP2-K & CoGP2-K

Input:

- Pair of dispatching rules : $\{DR_l, DR_h\}$
- DJSS problem instance.
  - uncertainty configuration.
  - set of machines $\mathcal{M}$.

Output: Total flow time :$\mathcal{T}_f$

Set of system state vectors: $\mathcal{H} \leftarrow \emptyset$.
Cluster Centroids: $\{\mathcal{C}_l, \mathcal{C}_h\} \leftarrow \emptyset$.
while *new jobs arrive* **do**
  foreach $m \in \mathcal{M}$ **do**
    $\mathcal{F}(m) =$
    $[MTWK, MTWKr, MTWKav, MNOPr, MNOPw, MUTL]$.
    if $size(\mathcal{H}) > 2$ **then**
      $\{\mathcal{C}_1, \mathcal{C}_2\} \leftarrow KmeansCluster(\mathcal{H})$
      $\{\mathcal{C}_l, \mathcal{C}_h\} \leftarrow associateClusters(\{\mathcal{C}_1, \mathcal{C}_2\})$
      if $distance(\mathcal{C}_l, \mathcal{F}(m)) \leq distance(\mathcal{C}_h, \mathcal{F}(m))$ **then**
        Use DR_l for sequencing on machine m
      else
        Use DR_h for sequencing on machine m
    else
      Use DR_l for sequencing on machine m.
    Add $\mathcal{F}(m)$ to $\mathcal{H}$.
  end
  Update $\mathcal{T}_f$.
end

The K-means clustering component of the method is explained using Algorithm 6. Note that the choice of K-means algorithm is based on its popularity and any other clustering algorithm should also work. A ma-

chine state vector is constructed using the terminal set used in GP3 method (line 5). Initially the set of machine state vectors $\mathcal{H}$ is empty. As the simulation progresses, the machine-state vectors are stored in $\mathcal{H}$.

At the outset, when the simulation is just warming-up and the size of $\mathcal{H}$ is very small the following steps are taken. Initially the algorithm starts simply by using the queue lengths ($MNOPr$, number of remaining operations on a machine) to classify between bottleneck and non-bottleneck machines; higher value of $MNOPr$ implies high level of bottleneck. Once the number of state vectors is greater than 2 ($k = 2$), it is possible to apply clustering method but we continue associating (labeling) the centroids to bottleneck and non-bottleneck machines using the feature $MNOPr$; this is done for a small number (10) of jobs during the warm-up period. Thereafter, in line 7, when $\mathcal{H}$ becomes sufficiently large, for every new pair of centroids their distance is calculated from the pair of centroids obtained in the previous step which are already labeled as bottleneck ($\mathcal{C}_h$) and non-bottleneck ($\mathcal{C}_l$). Based on the distance values, the new centroids are then labeled (line 8).

Once the labeled centroids are obtained, (either using $MNOPr$ initially or using the preceding centroids labels) their distance from the current machine state vector is determined (lines 9-12) and those values are used to decide the dispatching rule from $\{DR_l, DR_h\}$, to be used for sequencing. The preliminary study showed that after the warm-up period of DJSS simulation, during which a fixed number of jobs which have been processed but will be ignored for total flow time computation, there are sufficient number of machine-state vectors for the cluster centroids $\mathcal{C}_l$ and $\mathcal{C}_h$ to be distinct. After the sequencing on the machine m is completed by the chosen dispatching rule (line 10 or 12), the total flow time $\mathcal{T}_f$ is updated (line 17).

Co-evolutionary GP2-K (CoGP2-K)

GP2-K is designed to evolve a pair of DRs independently in separate subpopulations and then they are applied to a DJSS problem instance where they interact with each other through sequencing decisions. GP2-K method does not take into account the effect of this interaction between the two dispatching rules. Cooperative co-evolution is a technique which is applicable to a problem with interacting sub-components. Therefore we propose a cooperative co-evolutionary GP2-K i.e. CoGP2-K.

In CoGP2-K we divide the evolution into two stages as described in Algorithm 7. The notion behind using two separate stages is to allow sufficient generations of evolution for the dispatching rules to capture the required characteristics from the two configurations of training instances used in the first stage. And then further co-evolve to incorporate the interactions ensuing due to the sequencing decisions from each rule. By keeping the two separate stages in evolution, we better control the learning process of our GPHH method.

In the first stage, for some generations the dispatching rules are evolved in separate sub-populations. In this stage, each subpopulation is associated with a specific uncertainty configuration, similar to GP2-K method. In the second co-evolutionary stage, each individual in a sub-population is paired with the best individual from the other subpopulation. In this stage, a single uncertainty configuration is used for both subpopulations which corresponds to lower uncertainty level.

In the lines 3-12 of Algorithm 7, the first stage of the method is presented. The evolution in the subpopulations is performed separately using specific uncertainty configurations without any interaction between the two. The co-evolutionary stage is described in lines 13-24. For each generation, the best individuals from each subpopulation is determined (lines 15-16). For calculating the fitness of an individual, the best individual from the other subpopulation is paired with it, this pair is evaluated using the procedure described in Algorithm 6, which is same as what was

Algorithm 7: Co-evolutionary method - CoGP2-K

Input:

- $\mathcal{G}_c$, the generation after which co-evolutions starts.
- $\mathcal{G}_\tau$, total number of generations.
- DJSS training instance ($\mathcal{P}_t$).
  - $\mathcal{U}_l$ uncertainty configuration (low)
  - $\mathcal{U}_h$ uncertainty configuration (high)

Output: Pair of dispatching rules : $\{DR_l, DR_h\}$

```
Initialize subpopulations S_1, S_2
Set g ← 0
while g ≤ G_c do
    g ← g + 1
    foreach individual I ∈ S_1 do
        assign fitness to I using DJSS simulation with U_l config.
    end
    foreach individual in S_2 do
        assign fitness to I using DJSS simulation with U_h config.
    end
    Evolve individuals in S_1, S_2 using crossover and mutation.
end
while G_c < g ≤ G_τ do
    g ← g + 1
    I_1 ← BestIndividual(S_1)
    I_2 ← BestIndividual(S_2)
    foreach individual I ∈ S_1 do
        assign fitness to the pair {I} using Algorithm 6 with U_l
          configuration and {DR_l, DR_h} ← {I, I_2} for
          problem-instance P_t.
    end
    foreach individual I ∈ S_2 do
        assign fitness to the pair {I} using Algorithm 6 with U_l
          configuration and {DR_l, DR_h} ← {I_1, I} for
          problem-instance P_t.
    end
    Evolve individuals in S_1, S_2 using crossover and mutation.
end
```

used

for GP2-K. So the lines 18 and 21 of Algorithm 7 call the Algorithm 6. The configuration used for fitness evaluation is $\mathcal{U}_l$, which corresponds to the low level of uncertainty. After all the generations are complete, the combination of best individuals from the last generation is returned as output.

3.3.1 Experiment Design

Uncertainty Configurations

In order to simulate a practical shop environment it is important to consider different types of uncertainty configurations due to different job arrival patterns. Also, not all the machines have equal workload, for example, a finishing equipment required to fix small issues in a workshop. Taking into account such considerations, We have considered *six* uncertainty configurations for the machines which are presented in Table 3.12. The columns correspond to the machines. Each machine is associated with one or more θ parameter settings which follow exponential distributions. The scale parameter (β) of the associated exponential distributions are given.

Moreover, in practice, a production environment is characterized by varying defect and rework rates [196] which is reflected through the variation in levels of uncertainty. Therefore, some of the machines are associated with a pair of parameter values, e.g. $m6$ in configuration III is associated with two scale parameters, $\{0.6, 0.3\}$. The duration for which a machine is associated with a specific uncertainty level is uniformly sampled from $[1000, 1700]$. This was chosen based on the requirements imposed by our simulation, so that there is a sufficient duration for the machine to move through transient to a steady bottleneck level.

We require two levels of uncertainty configurations during training for the methods GP2-K and CoGP2-K. These two configurations are shown in bold in Table 3.12. The configurations III and VI correspond to the low and high levels respectively. As explained earlier for GP2-K method, the difference between the uncertainty levels of two configurations is not high.

Table 3.12: Machine uncertainty (scale parameter (β) values of exponential distributions)

	m0	m1	m2	m3	m4	m5	m6	m7	m8	m9
I	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
II	0.1	0.1	0.1	0.1	0.1	0.2	{0.35, 0.1}	{0.35, 0.1}	{0.2, 0.1}	{0.3, 0.1}
III	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.3**	{**0.6, 0.3**}	{**0.6, 0.3**}	{**0.2, 0.1**}	{**0.3, 0.1**}
IV	0.3	0.3	0.3	0.3	0.3	0.5	{0.8, 0.5}	{0.8, 0.5}	{0.4, 0.3}	{0.5, 0.3}
V	0.3	0.3	0.3	0.3	0.3	0.9	{1.2, 0.75}	{1.2, 0.75}	{0.65, 0.4}	{0.75, 0.4}
VI	**0.1**	**0.1**	**0.1**	**0.35**	**0.35**	**0.35**	**0.65**	**0.65**	**0.65**	**0.65**

In the configuration VI, for the machines $m6 - m9$ associated with $\beta = 0.65$ the uncertainty level is marginally higher but consistent. Similarly for machines $m3 - m5$, $\beta = 0.35$ which is marginally higher. If we had chosen the configurations I and V instead, then we would have evolved rules for a scenario where the shop has very high variability or with very low variability. Such extremes are not the normally observed states of a productive shop.

The different test and train configurations used are summarized in the Table 3.13. GP1(l) and GP1(h) are the standard GP methods. For testing all the six configurations are considered. Since our proposed methods and the GP3 method utilizes the two configurations for training, we also train the standard GP separately on these configurations and use them to compare with the other methods.

We compare our proposed approaches with standard GP and GP3 [106]. GP3 considers the bottleneck machine and therefore forms a good benchmark to compare with. It must be noted that GP3 was applied to determin-

istic static JSS problems where as this work focuses on dynamic JSS problems under uncertainty. There is no other work which considers hyper-heuristic approach for this type of problems.

The JSS objective is total flow time.

$$total\ flowtime = \sum_{i=1}^{N} (\mathcal{C}_i - \mathcal{R}_{\mathcal{J}_i})$$

where $\mathcal{R}_{\mathcal{J}_i}$ is the release time of job $\mathcal{J}_i$ and $\mathcal{C}_i$ is the completion time.

Table 3.13: Training and Test Configurations

	Train	Test
GP1(l)	III	I, II, III, IV, V, VI
GP1(h)	VI	I, II, III, IV, V, VI
GP3	III, VI	I, II, III, IV, V, VI
GP2-K	III, VI	I, II, III, IV, V, VI
CoGP2-K	III, VI	I, II, III, IV, V, VI

3.3.2 Results and Discussions

Now we present our set of experiments corresponding to GP3, GP2K and CoGP2K. Referring Table 3.13, we compare the $five$ methods over six test configurations. For each method the solutions are tested over 30 problem instances. As for the previous experiments, the Wilcoxon-rank-sum-test is used to compare the performance of the methods. A significance level of 0.05 is considered.

The results are summarized in Tables 3.14-3.18. Each cell in the tables consists of a triplet detailing the corresponding statistical test result. Consider the first cell of Table 3.14, [25-5-0] which should be read as $[win\text{-}draw\text{-}lose]$. The Table 3.14 compares the GP1(l) method against the other 4 methods. The cell [25-5-0] corresponds to the column of test configuration I and

row of method GP1(h). It means that the GP1(l) performed significantly better in 25 problem instances, is similar in 5 instances and is significantly poor in 0 instances. Furthermore, for those cells which show a significant difference in more than 5 problem instances, color shading is used. The green color is used to show those cases where significant improvement is observed in more than 5 problem instances i.e. $win \geq max(5, loss)$. Similarly orange denotes significantly worse performance, $lose \geq max(5, win)$. For each table, heading mentions the method name and associated training configurations.

In Table 3.14, we compare the standard GP method, GP1(l) which is trained on configuration III. For the test configuration I, which is characterized by low uncertainty level, GP1(l) outperforms all methods. It is better than CoGP2-K by a thin margin. However, as the level of uncertainty increases, its generalization drops rapidly. For test configurations IV to VI, GP1(l) performed noticeably worse than other methods. Refer the cells marked in orange.

Table 3.14: GP1(l) (Configuration-III)

	I	II	III	IV	V	VI
GP1(h)	**[25-5-0]**	**[18-12-0]**	[0-26-4]	[0-5-25]	[0-0-30]	[0-1-29]
GP3	**[29-1-0]**	**[25-5-0]**	[0-27-3]	[0-8-22]	[0-1-29]	[0-4-26]
GP2-K	**[19-11-0]**	**[18-12-0]**	[0-26-4]	[0-7-23]	[0-0-30]	[0-1-29]
CoGP2-K	**[5-25-0]**	[3-27-0]	[0-26-4]	[0-4-26]	[0-0-30]	[0-1-29]

In Table 3.15, the standard GP method trained on higher uncertainty level corresponding to configuration VI is compared to all other methods. Its performance is significantly better than GP1(l) for configurations $IV - VI$, as evidenced by the cells marked in green. These cells correspond to configurations with relatively higher uncertainty levels. GP1(h) performs poorly for configurations $I - II$ for GP1(l) and CoGP2-K. Refer the colored cells in the first two columns. Its is almost an exact draw with

GP2-K across all configurations. For configurations with a higher level of uncertainty it is significantly similar to CoGP2-K for most test problems.

Table 3.15: GP1(h) (Configuration-VI)

	I	II	III	IV	V	VI
GP1(l)	[0-5-25]	[0-12-18]	[4-26-0]	**[25-5-0]**	**[30-0-0]**	**[29-1-0]**
GP3	[4-26-0]	[3-27-0]	[0-30-0]	[0-29-1]	[1-27-2]	[3-26-1]
GP2-K	[0-29-1]	[0-30-0]	[0-30-0]	[0-30-0]	[0-30-0]	[0-30-0]
CoGP2-K	[0-21-9]	[0-22-8]	[0-27-3]	[1-27-2]	[4-26-0]	[2-28-0]

In Table 3.16, the performance of GP3 is presented. It outperforms GP1(l) on configurations $IV - VI$ but is significantly poor for most of the test problems on configuration $I - II$ (refer the orange cells). Apparently, the bottlenecks arising for higher level of uncertainty configuration (configuration VI) have had a dominating influence during training. Consequently, the generalization characteristic of GP3 is poor.

Table 3.16: GP3 (Configuration III & VI)

	I	II	III	IV	V	VI
GP1(l)	[0-1-29]	[0-5-25]	[3-27-0]	**[22-8-0]**	**[29-1-0]**	**[26-4-0]**
GP1(h)	[0-26-4]	[0-27-3]	[0-30-0]	[1-29-0]	[2-27-1]	[1-26-3]
GP2-K	[0-21-9]	[0-28-2]	[1-29-0]	[1-28-1]	[0-30-0]	[0-29-1]
CoGP2-K	[0-3-27]	[0-12-18]	[0-25-5]	[1-29-0]	[1-29-0]	[0-29-1]

In Table 3.17, the results from GP2-K, which uses clustering method during testing, are shown to be similar to GP1(h), as mentioned earlier. It outperforms GP3 for test configuration I, even though they both use same configurations in training. This is because in GP2-K the dispatching rules for bottleneck and non-bottleneck scenarios are learned using specific training configuration and later a clustering method is used to choose

the rules during testing against a non-linear GP classifier in the former.

Table 3.17: GP2-K (Configuration III & VI)

	I	II	III	IV	V	VI
GP1(l)	[0-11-19]	[0-12-18]	[4-26-0]	**[23-7-0]**	**[30-0-0]**	**[29-1-0]**
GP1(h)	[1-29-0]	[0-30-0]	[0-30-0]	[0-30-0]	[0-30-0]	[0-30-0]
GP3	**[9-21-0]**	[2-28-0]	[0-29-1]	[1-28-1]	[0-30-0]	[1-29-0]
CoGP2-K	[0-26-4]	[0-21-9]	[0-27-3]	[0-28-2]	[3-27-0]	[2-28-0]

In Table 3.18, the performance of the co-evolutionary method is shown. Across all the test configurations this method is able to perform well. Though it is marginally poor in test configuration I with a very low uncertainty level. It outperforms GP1(h), GP3 and GP2-K on configurations $I - III$ as evidenced by green cells. In the case of test configurations $IV - VI$, the performance CoGP2-K outperforms GP1(l) and is almost similar to other methods. This shows that the generalization characteristic of the proposed method is superior to all other methods considered in this work. The co-evolution process takes into account the interactions of the dispatching rules through their sequencing decisions in combination with a more effective clustering method to classify the bottleneck and non-bottleneck dispatching rules.

Table 3.18: CoGP2-K (Configuration III & VI)

	I	II	III	IV	V	VI
GP1(l)	[0-25-5]	[0-27-3]	[4-26-0]	**[26-4-0]**	**[30-0-0]**	**[29-1-0]**
GP1(h)	**[9-21-0]**	**[8-22-0]**	[3-27-0]	[2-27-1]	[0-26-4]	[0-28-2]
GP3	**[27-3-0]**	**[18-12-0]**	**[5-25-0]**	[0-29-1]	[0-29-1]	[1-29-0]
GP2-K	[4-26-0]	**[9-21-0]**	[3-27-0]	[2-28-0]	[0-27-3]	[0-28-2]

In Figure 3.6, we present boxplots to compare generalization perfor-

mance of the methods on individual instances. We picked 5 out of the total 30 problem instances under the two test configurations III and VI. The groups of 4 boxplots correspond to one problem instance each. A boxplot is marked in green if its median is lower than the medians corresponding to all other boxplots in the same group. The order of methods is same as mentioned in the caption. In a large number of the cases, the boxplot corresponding to CoGP2-K enjoyed the smallest median value, with respect to these two configurations.

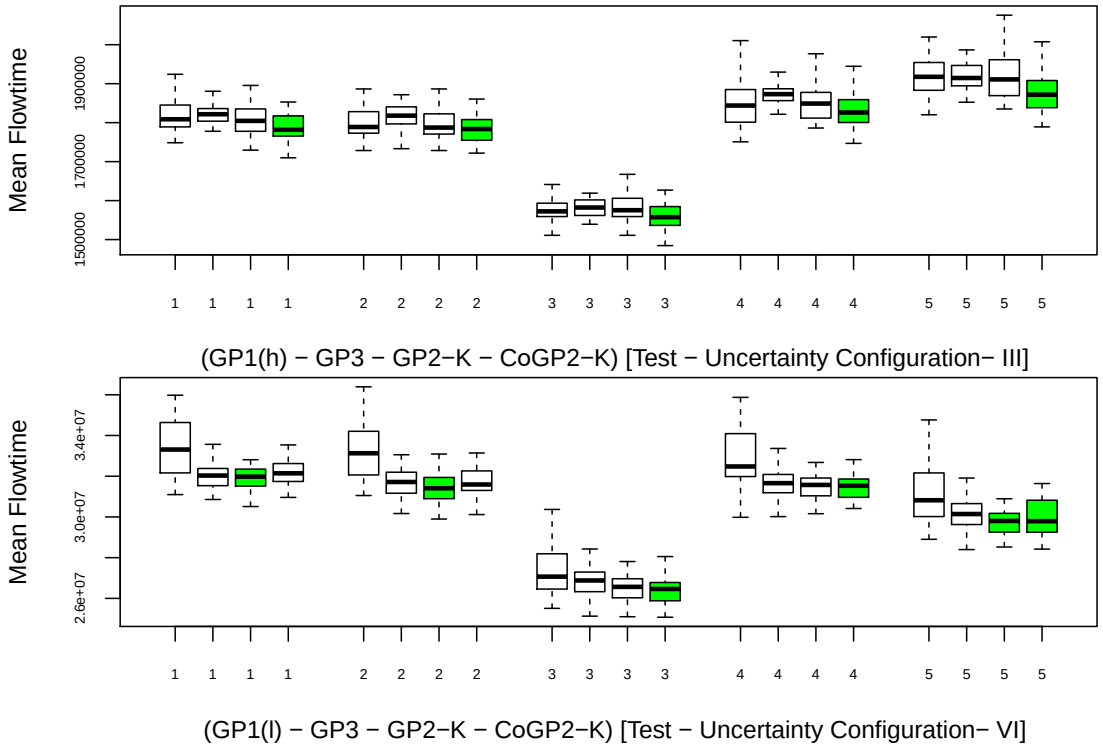


Figure 3.6: Boxplots: The order of boxplots is same as mentioned in the caption. The result with lowest medians are marked in green.

3.3.3 Analysis

We compare the pair of dispatching rules evolved using CoGP2-K. We determine the frequency of the terminals obtained from the 30 runs, which is shown in Figure 3.7. There is a clear difference between the evolved rules with respect to their choice of terminals. The rules which were evolved using a configuration with higher level of uncertainty tend to use the terminals corresponding to the jobs more often. A higher level of uncertainty leads to bottleneck machines, leading to a larger queue length. This makes the problem harder. Therefore, the dispatching rule which is evolved on this configuration tends to use more of the terminals which correspond to the job characteristics as explained below.

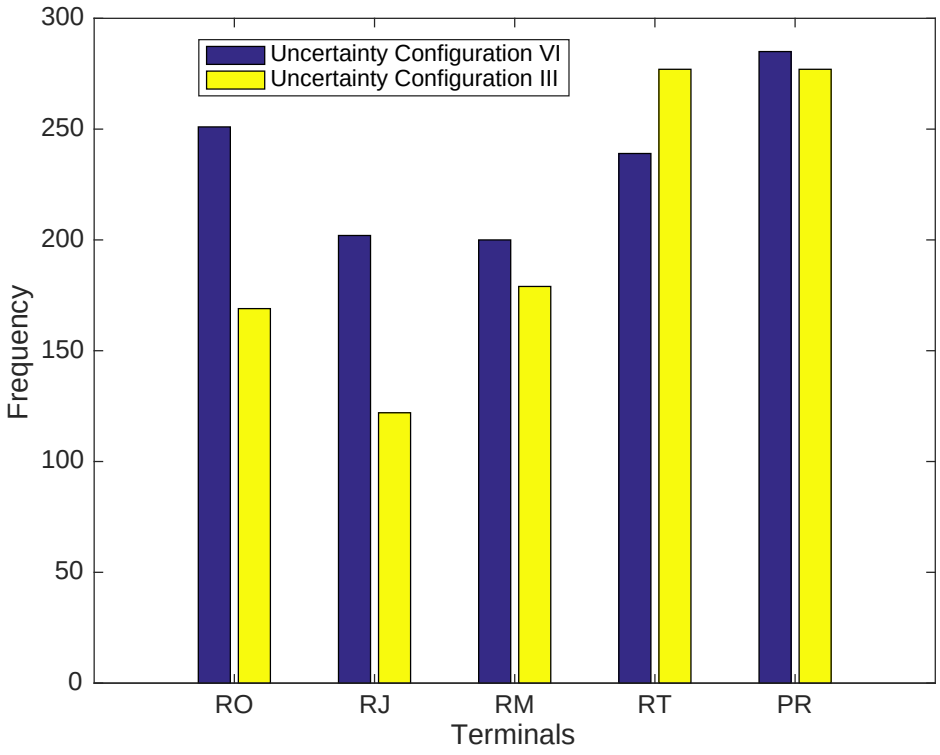


Figure 3.7: Histogram of Frequency of Terminals

Dispatching Rule 3.2: CoGP2-K - (low)

```
(* (If (If (Min RJ RO) PR (If (- RM PR)
0.580 RM)) (+ (* (Min 0.522 RJ) (*RT PR))
(* PR PR)) RT) (- (Max (- (If RO RT PR)
(Min 0.0837 RM)) (Max (/ RJ PR) (Min
0.0597 RT))) RT))
```

Dispatching Rule 3.3: CoGP2-K - (high)

```
(Min (Min (+ RO RO) (+ (- (- (/ RT RM) (+
(Min PR PR) (* RO PR))) (Max 0.250 RT))
(/ RM (* (* RO PR) (* (Min RJ RT) PR)))))
(- (/ (+ RO (Min RJ RT)) (/ PR RO)) (* (Min
PR PR) (* (Min RJ RT) PR))))
```

For a bottleneck machine, the priority value is expected to be very different for a job with many pending operations compared with a job with fewer pending operations, due to their higher impact on scheduling objective when compared with a non-bottleneck machine. Therefore, the two terminals RO and RJ corresponding to remaining operations for job and ready time of job respectively are more prominently used in dispatching rules for bottleneck machines. An example of one of the best pairs of dispatching rules evolved using CoGP2-K is given above. The terminals RO and RJ are shown in bold in both the rules.

3.3.4 Section Summary

In this section, we developed new GPHH approaches which explored the ability of genetic programming to evolve dispatching rules for specific scenarios in DJSS problems under uncertainty. We considered the varying bottleneck and the non-bottleneck machines in the shop as the two scenarios. Firstly, we developed a clustering based method to identify bottleneck machines in DJSS problems. Then using this method two new GPHH methods, CoGP2-K and GP2-K. CoGP2-K were developed for evolving a pair of rules for bottleneck and non-bottleneck machines. The results

show that CoGP2-K which is a cooperative co-evolutionary method outperformed all the other benchmark methods.

3.4 Further Discussion

We would like to highlight the subtle differences between the two approaches developed in each of the sections in this chapter. The ENT, EXP and EMA approaches essentially consider the job characteristics to improve the scheduling performance. The jobs in a print industry are an example where the varying characteristics of jobs [191] lead to uncertainty in processing times e.g. the processing time of a print job which is sensitive to paper quality will generally get affected because the paper is obtained from various sources. On the other hand, when the varying machine characteristics play an important role on the scheduling performance then an approach like CoGP2-K is effective. Consideration of the varying characteristics of machines is very important in production systems where frequent set-ups and maintenance activities are unavoidable. In fact, in many such production environments machine characteristics undergo a cycle [196] of variations between maintenance activities. Therefore, even though these approaches deal with uncertain processing times, their applicability is suitable for different shop environments.

3.5 Chapter Summary

The goal of this chapter was to develop GPHH techniques for evolving dispatching rules for DJSS problems with uncertainty in processing times. This goal was successfully achieved through two major sub-goals.

This chapter presents a first attempt to incorporate uncertainty information into dispatching rules using novel GP representations. Also, this is the first work which tries to combine the ideas of ex-post and ex-ante optimization into hyper-heuristic approaches for problems in uncertain

environments. We developed new representations for GP with the ability to incorporate the uncertainty information into the dispatching rules. Firstly, a new representation was developed by introducing a new terminal **EMA** which incorporated the information about uncertainty into the genetic programs. Then, inspired by ex-ante and ex-post optimization, two novel GPHH approaches were developed namely **ENT** and **EXP**. Our methods using the new representations were successful in evolving effective dispatching rules for DJSS problems under uncertain processing times.

This chapter presents a first method to consider separate dispatching rules for the bottleneck and non-bottleneck machines in a DJSS problem under uncertain processing times while taking into account the interactions between the two rules, through their sequencing decisions. We developed a novel cooperative co-evolutionary method called **CoGP2-K** for evolving a pair of dispatching rules each for the specific machine type. We also developed a new method to identify bottleneck machines in dynamic scheduling problems which is based on clustering and extends the existing work from [106]. Furthermore, by evolving the dispatching rules in two stages, our proposed algorithm enables the GPHH to effectively learn all the desired features into the dispatching rules. Thus, CoGP2-K was successful in evolving effective dispatching rules which outperformed the existing benchmark methods.

By achieving these two sub-goals, this chapter has exploited two major strengths of genetic programming. Firstly, the flexible representation of GP was utilized to *directly* incorporate the uncertainty information of processing times into the terminals of dispatching rules. Secondly, the powerful ability of genetic programming to learn from complex scenarios without being explicitly provided with features was utilized to evolve a pair of rules for bottleneck and non-bottleneck machines. Since the bottlenecks are a result of variability and uncertainty in the dynamic shop, the effect of uncertainty is *indirectly* taken into account by the GPHH approach.

This was benefited by the useful ability of cooperative co-evolutionary approaches to effectively learn from the interactions among the components of a system.

Though both these sub-goals were achieved by developing GPHH approaches for DJSS problems under uncertainty, there some subtle differences which must be noted. For the first sub-goal, the new representations relied on the assumption, which is practical, that uncertainty in the operations comprising a job follow a probability distribution. In the second sub-goal, the machine characteristics are given prominence and the uncertainty in the operations queued on them are considered. This subtle difference is another reason for this chapter to investigate two parallel research directions. Considering the large number of different job shop scheduling problems, both of these considerations and their assumptions are common in the practical job shops.

This chapters considers only two specific scenarios pertaining to bottleneck levels of the machines. In practice, due to uncertainty in processing times, many different scenarios arise in the shop environment. Is it possible to evolve rules for each of these scenarios? How to identify the scenarios which are important? Or in other words, what DJSS training instances must be sampled during GPHH for evolving better dispatching rules? Moreover, when more than one objective is considered, should we consider parallel evolutionary methods? In the next two chapters, we try to explore the methods to address these questions.

Chapter 4

Active Sampling Methods for Dynamic Job Shop Scheduling under Uncertainty

4.1 Introduction

In our previous chapter, we were successful in designing rules for two specific DJSS scenarios, corresponding to bottleneck and non-bottleneck machines, using GPHH approach. In general, designing a single dispatching rule for all the shop scenarios is difficult and will be hard to yield the (near) optimal scheduling performance [88, 219]. As demonstrated in the previous chapter and also highlighted in the literature survey, GPHH is a promising technique for automatically designing dispatching rules. In light of this, we develop the premise of this chapter, which is to consider GPHH approaches for designing more than one rule for the dynamically varying shops scenarios. We will present some examples which show the varying shop scenarios in a dynamic environment.

4.1.1 Shop Scenarios in Dynamic Environment

In print industry, more broadly also known as document management services, using an example we considered earlier in Chapter 1 [189, 190, 191, 192], we discussed about a large number of shop scenarios arising due to the dynamic environment characterized by events like break downs, operator mistakes, spurt in arrival of high priority jobs, etc. The resources in a print shop are printers, cutters, shrink wrapper, binder, etc. In a dynamic print shop, different jobs have different patterns, e.g. short jobs with short deadlines, or recurring jobs with specific characteristics like marketing pamphlets [104, 192]. These get disrupted when jobs with a different pattern arrive at the same time, e.g., a large number of jobs with long patterns when shorter jobs are nearing deadline. This leads to uncertainty in the shop and adversely affects the scheduling objective(s). [192] proposes using dynamic cells which essentially change the structure of print shop by grouping together similar jobs and also grouping together the required equipment (machines) dynamically, with respect to changing (aforementioned) shop patterns. We will refer to these patterns as scenarios. The idea of dynamic cells is born out of the need to recognize and deal with a large number of shop scenarios arising in the dynamic environment. We can clearly see that this example highlights the importance of recognizing the large number of shop scenarios in the dynamic shop and developing methods to deal with them.

Similarly, in the automobile manufacturing industry, modern production units use automated robots to manufacture the components of an automobile. A single production unit has the ability to manufacture vehicles of many designs and specifications. The orders for these products arrive dynamically and many of the set-ups are automated in the robots e.g. [198]. But when new orders with new specifications arrive, the defect rate initially is high which leads to uncertainty in processing times. Thus, the sudden arrival of a new order with complex specifications results in a new scenario in the shop, which has an effect on all the other

jobs which require the same resources. Similarly, if a particular passenger vehicle model becomes popular, new orders for that vehicle with high priority arrive at the production unit, with short deadlines. This scenario could get further complicated when an imminent software upgrade of a system leads to delays in manufacturing of a set of components. Such variability is very common in the manufacturing industries [105]. Designing a universal dispatching rule which can deal with all such scenarios is not practical. Instead, we need dispatching rules which are designed for specific shop scenarios.

4.1.2 Multiple Dispatching Rules

While we discussing the importance of designing scenario specific rules instead of a universal rule, a related idea in machine learning is in the context of global learning versus local learning [98]. A global learner considers the whole data and looks for patterns at the global level. A local learner, on the other hand, considers subsets of data and is able to find useful patterns which could be missed by the global learner. Local learning directly utilizes critical information for sub-tasks rather than obtaining a global perspective. The notion of grouping similar problems based on their characteristics and developing solutions has been considered in other contexts. In particular, evolutionary multitasking [82] attempts to solve multiple optimization problems simultaneously by exploiting the implicit parallelism of population-based search. By utilizing the similarities and differences across the tasks, they are solved simultaneously. Multitasking facilitates the implicit knowledge transfer between diverse tasks and helps achieving optimization across the problem domains. In hindsight, these ideas essentially highlight the research directions which focus on dividing a global problem into multiple tasks and use more specific task oriented information to solve them. In our context, this further reinforces the importance of considering a large number of shop scenarios and designing

scenario-specific rules.

The idea of using more than one dispatching rules has been considered to a limited extent in recent works [88, 89]. Their focus is more on switching the dispatching rules for dynamically varying shop scenarios. Therefore, they do not attempt in designing new rules for the varying shop scenarios. Instead, they experiment with already existing manually designed rules. Secondly and more importantly, the shop scenarios which they simulate only consider variation in utilization and due date factor. In our previous examples, we could see that the combination of jobs of different characteristics which arrive at the shop lead to a large number of distinctive patterns or shop scenarios. Therefore, it is important to consider the effect of the different characteristics of arriving jobs in a complex shop environment. Following that we will need to develop methods for identifying the challenging scenarios (as described in Section 4.1.1) which arise in the shop. Once they are identified, the next step is designing dispatching rules for those challenging scenarios. Finally once the scenario-specific rules are designed, selecting the appropriate dispatching rule when a corresponding scenario for which the rule was designed is encountered in the shop. Since GPHH has shown a lot of potential in automatically designing dispatching rules, it is a good research direction to exploit this approach to designing rules for the specific scenarios. We now discuss the various challenges which we need to address in our endeavor to address these requirements.

In theory, the number of patterns or scenarios which could be observed in a complex shop is infinite. A production system which gets a large number of different kinds of orders is expected to encounter more complex scenarios than a shop with low productivity. Since the productivity varies for a shop (decided by many factors like market supply and demand), the shop scenarios also vary. For improving the overall productivity of the shop, the scheduling effectiveness needs to be improved for those shop scenarios which are 'complex'. Thus, identifying these complex shop sce-

narios is an important step. In this context, the term complex is used to reflect the large number of factors involved in the description of scheduling problem in a dynamic shop. Even though GPHH approach has been successful in evolving good dispatching rules, it is still computationally expensive. Therefore, on the one hand we have a huge number of possible DJSS scenarios and on the other hand we have the computational constraint for GPHH which cannot evolve specific rules for each and every shop scenario. In terms of the GPHH approach, a large number of DJSS shop scenarios are essentially part of the DJSS training instances, implying that we need methods to identify DJSS training instance corresponding to complex shop scenarios. So, essentially the challenge is in efficiently identifying those training instances which capture the complex shop scenarios.

4.1.3 Active Learning methods

These challenges lead us to a sub-field of machine learning called active learning. Active learning [204] is a concept based on the idea that a machine learning algorithm will perform better if it has the choice to select the training instances to learn from. The active learning problems selectively and adaptively samples from the input space to deal with the estimation of unknown parameters. Over the recent years, this idea has inspired many algorithms for semi-supervised learning. The question which an active learning approach tries to address is: "How do we select instances from the underlying data to label, so as to achieve the most effective training for a given level of effort?" [3].

If we desire to consider active learning techniques in the context of GPHH for efficiently identifying those training instances which capture complex shop scenarios, we need to address some specific issues. The active learning techniques, like uncertainty sampling [204] for example, could leverage the underlying distribution of the input space and in general convenienced by the ease of evaluation of the sampled instance's abil-

ity to improve the model. Since GPHH is a hyper-heuristic approach, which means the evolved dispatching rule needs to generate a solution, we cannot directly evaluate the sampled instances just with the evolved rules. We actually need to apply the dispatching rules on a set of DJSS instances and evaluate its performance which could indirectly evaluate the sampled DJSS instances. Essentially, the evolution of the dispatching rules using an identified training instance is actually exploitation while sampling newer training instances and then evaluating their efficacy to evolve rules amounts to exploration. This is actually a multi-armed bandit problem. Since the computational budget is limited, allocating resources toward the sampling of instances from the input data space versus evolution of dispatching rules using already sampled DJSS instances can be considered as a dilemma between exploration and the exploitation. In fact, active learning and the theory of multi-armed bandits (MAB) are closely related [74]. By definition, the multi-armed bandit problem is a problem dealing with allocation of a limited set of resources among competing choices to minimize regret. In essence, the theory of multi-armed bandits and its applications particularly focus on the trade off between exploration and exploitation. Therefore, a research direction which considers these techniques for evolving scenario-specific dispatching rules for DJSS problems in complex environments is quite promising.

Integrating active learning methods into GPHH has some important challenges. Firstly, the current GPHH framework has no facility to actively sample training instances while evolving dispatching rules. Secondly, there is no existing method to evaluate the ability of training instances in promoting the evolution of scenario-specific rule or to evaluate how good a training instance represents a complex shop scenario. Without addressing these two challenges, the techniques from active learning and the theory of multi-armed bandits cannot be employed. Furthermore, since GPHH is already computationally intensive, the integration of these techniques should consider aspects of computational efficiency as well

and should form the main criteria for selection of appropriate methods. In this line of discussion, it is worthwhile to note that Bayesian optimization is an area which deals with optimizing functions that are very expensive to evaluate. Moreover, Gaussian process bandits which falls under the purview of Bayesian optimization, multi-armed bandit is a related technique which could be a key to tackling our issues.

Based on these research challenges and the motivation to employ active learning methods with GPHH, now we construct the goals of this chapter.

4.1.4 Chapter Goals

The goal of this chapter is to develop new GPHH approaches which actively sample DJSS training instances toward evolving scenario specific dispatching rules for DJSS problems.

The following are the two sub goals.

- Develop a new GPHH framework which facilitates the active sampling of DJSS training instances. This also includes developing a method to compare the training instances in their ability to support the evolution of effective dispatching rules.

- Develop two new active sampling methods using the ϵ-greedy strategy and the Gaussian process bandits (GPB) technique as a part of the new GPHH framework.

4.1.5 Chapter Organization

The remaining chapter is organized as follows. The next section describes the proposed algorithms. The Section 4.3 presents the design of experiments. The Section 4.4 presents the results and discussion. The final section provides a summary to the chapter.

4.2 The Proposed Methods

In order to achieve the goals in this chapter, we need to incorporate some new facets into the current GPHH framework to enable the active sampling methods. In particular, the GPHH approach must be enabled to evaluate the DJSS problem instances which were sampled to train a scenario-specific dispatching rule.

Before we describe our new framework, we need a method to associate the DJSS training instances with the shop scenarios which they represent. Since our aim is to enable the GPHH to evolve dispatching rules each specific to different scenarios, we should first be able to map the DJSS instances to the complex shop scenarios. Consequently, for the GPHH system to train, it should be provided with groups or clusters of DJSS problem instances where each cluster of problem instances corresponds to a specific shop scenarios. Therefore, firstly we describe a feature extraction and clustering method for the DJSS problem instances.

4.2.1 Clustering of DJSS Problem Instances

Referring back to our previous examples of dynamic job shops, in particular, the example from print industry, the shop scenarios are defined mainly by the characteristics of the arriving jobs [104, 189]. These characteristics pertain to #operations, processing time of these operations, their due date, etc. The other factors which influence a shop scenario are dynamic events like machine break downs, variability in set-up times, etc., which lead to uncertainty in shop parameters. Since processing time is an important job parameter influencing most of the objectives, we consider the uncertainty in the processing times, which in essence captures the effect of aforementioned dynamic events. Due to the same reason, we had considered uncertainty in processing times in Chapter 3 as well.

Now, with this background, we extract features from the DJSS problem instances. Firstly, the basic features for each job are extracted as described

Table 4.1: Job Features

Feature	Description
#operations	number of operations per job.
p	estimated processing time of the job.
Δ^p	$\frac{p'}{p}$, p' is the actual processing time with uncertainty.
due date factor (ddf)	$\frac{(\delta_{duedate}-\delta_{reldate})}{p'}$; where $\delta_{duedate}$ is the due date and $\delta_{reldate}$ is the release date

in Table 4.1. These features, as explained above, are closely related to the shop scenarios. We would also like to mention that similar feature extraction methods have been employed by [213] though for static scheduling problems and also their work is not related to GPHH or the goals of this chapter. Δ^p,in the Table 4.1 is the delay ratio as defined in Chapter 1. The estimated processing time p is the expected processing time which is used by the dispatching rule to make sequencing decisions. The realized processing time p' is the *actual* processing time obtained after the job is completed on the machines. The number of operations, due date factor and the processing time are the parameters which help in defining the characteristics of a job which in turn influences a shop scenario.

Once the basic features for each of the jobs in a DJSS problem instance are extracted, we need to create a feature vector for the dynamic JSS problem. In order to do that, firstly each of the basic features for each of the jobs is aggregated. And then, the first, second and third quartiles of each aggregate are calculated to form a 12-dimensional feature vector characterizing each problem instance. We illustrate this feature extraction methodology using an example below.

Consider an example of a DJSS problem instance with just 10 jobs

$$\{j_1, j_2, j_3, \ldots j_{10}\}$$

For each job, the features described in Table 4.1 are calculated and aggre-

gated e.g., for processing time the aggregated feature values are:

$$\{p_1, p_2, p_3, \ldots p_{10}\}$$

Then for each feature aggregate, the quartiles are calculated. The feature vector of the DJSS instance is of the form

$$\{\#ops_{Q1}, \#ops_{Q2}, \#ops_{Q3}, p_{Q1}, p_{Q2}, p_{Q3}, \Delta^p_{Q1}, \Delta^p_{Q2}, \Delta^p_{Q3}, ddf_{Q1}, ddf_{Q2}, ddf_{Q3}\}$$

Here, ops and p refers to operations and its processing time respectively while ddf refers to due date factor. The subscripts $Q1, Q2, \ldots$ refer to the qaurtiles.

After extracting the feature vectors from each of the DJSS training instances, they are clustered to form groups of similar problem instances. Since we started extracting features from each job and then aggregated them for the jobs arriving at a shop, we expect that the combination of job characteristics arriving at the shop is reflected in our aggregated features. Therefore, after clustering the DJSS instances, we should expect that the problem instances corresponding to a cluster pertain to similar shop scenarios.

Now we propose our new GPHH framework.

4.2.2 GPHH Framework Using Active Sampling

Before delving into the details of our proposed framework, we briefly outline the commonly used GPHH approach. In the current GPHH approach, the genetic programming evolutionary process uses a set of DJSS training instances. The total number of the DJSS training instances used is quite low. If G is the total number of generations, then usually it is just a small multiple of G. For example [163] considers four DJSS training instances per generation. At the end of the algorithm, it is common for the best dispatching rule to be considered as the final result e.g. [100, 161]. Both these aspects are different from the proposed framework for GPHH.

The proposed GPHH framework is explained using the flowchart in Fig. 4.1. The framework consists of three main components, namely: training, testing and validation. These are shown using blue boxes in the flowchart.

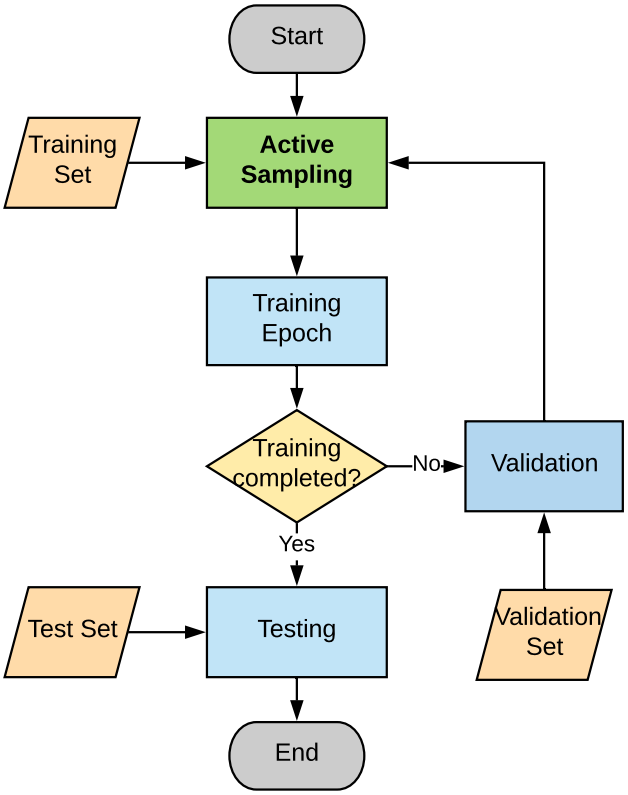


Figure 4.1: Proposed GPHH framework using active sampling.

Before the GPHH procedure starts, we divide the set of problem instances ($\mathcal{X}$) evenly into *three* sets: training set $\mathcal{S}$, validation set $\mathcal{V}$ and test set $\mathcal{T}$. The notation is defined in Table 4.2. We also set aside a smaller set ($\mathcal{H}$) of problems for hyper-parameter optimization, required by one of our algorithms. Since the details about hyper-parameter optimization are algorithm specific, we present them later when we discuss our experiments. In the flowchart, this step should be considered as a part of active sampling. Using the methodology described above, we extract features from each of the problem instances from the three sets and cluster them. These

three sets are used for the training, validation and testing respectively, as shown in the flowchart.

Table 4.2: Notation

Symbol	Definition
$\mathcal{X}$	set of all DJSS problem instances
$\mathcal{S}$,$\mathcal{V}$,$\mathcal{T}$	training, validation and test sets respectively.
$\mathcal{H}$	set of problems for hyper-paramter optimization.
$\mathcal{S}_c$,$\mathcal{V}_c$,$\mathcal{T}_c$,$\mathcal{H}_c$	denote clustered sets.
N_v	number of instances in each cluster of $\mathcal{V}$
$p(V_k, i)$	ith problem instance from the cluster v_k. similarly for $\mathcal{S}$ and $\mathcal{T}$.
G	total number of generations
γ	number of generations for which a subpopulation is evolved
$\mathcal{E}$	number of epochs $\mathcal{E} = G\%\gamma$
N_g	number of gens. for which all subpops. are evolved initially.
$\mathcal{R}_k$	rank order of all dispatching rules for a cluster v_k
$\bar{r}_{\omega_i}$	average rank of a dispatching rule over all clusters

The training step consists of many evolutionary (training) epochs. In the parlance of evolutionary computation, an epoch, γ, consists of a fixed number of generations. Thus if G is the total number of generations, then the number of epochs $\mathcal{E}$ is $G\%\gamma$, where $\%$ is modular division operator. At the end of each epoch, a set of dispatching rules is obtained. This is a common procedure across all our proposed active sampling algorithms under this framework.

As seen in the flowchart, a training epoch is preceded by the active sampling procedure. The goal of active sampling is to effectively sample those DJSS instances which represent the shop scenarios which have the ability to promote evolution of good rules. For the purpose of quantifying this ability of the DJSS instances, the validation step is considered. After the completion of an epoch, which considers a the specific set of sampled

DJSS instances for training, new dispatching rules can be obtained corresponding to specific DJSS training instances. By evaluating these dispatching rules on the validation set, the usefulness (quality of) DJSS instances can be indirectly evaluated. The validation procedure depends on the active sampling algorithm, therefore, we explain it in detail later. For now, consider that the validation step is able to quantify the quality of DJSS instances. The active sampling uses this information to sample better DJSS instances for the next epoch.

After the completion of all the training epochs, the final set of dispatching rules is obtained. Now the question is how to determine which dispatching rule is to be used when a test DJSS instance is presented. The testing component of the framework is designed to address this question.

Having outlined our GPHH framework with a flowchart, we explain in more detail our active sampling methods which are based on multi-armed bandits. We discuss two such methods in the following sections.

4.2.3 GPHH with Active Sampling using ϵ-greedy strategy

The purpose of active sampling methods is to tackle the exploration vs. exploitation dilemma. This dilemma has been extensively studied in the multi-armed bandit (MAB) [16] framework. In a typical multi-armed bandit problem, an agent is modeled which attempts to acquire new knowledge (explores) while simultaneously optimizing the decisions based on existing knowledge (exploits). The agent tries to balance these competing tasks in order to maximize the payoff over the considered period of time. In the MAB framework, the agent chooses an *action* from a discrete set of actions (arms), for which it gets a *reward*. In a sequence of *trials* the agent performs actions and gets rewards. In order to quantify the performance of agent a notion of *regret* is used which is the difference between the collective rewards of the agent and the reward of an optimal strategy.

Algorithm 8: GPHH with Active Sampling based on $\epsilon - greedy$ approach

Input: $\mathcal{S}_c$,$\mathcal{V}_c$

Output: Set of dispatching rules associated with clusters in $\mathcal{S}_c$

Create n subpopulations, where n is the number of clusters in $\mathcal{S}_c$.
for $k \leftarrow 1 : n$ **do**
  for $g \leftarrow 1 : N_G$ **do**
    Sample an instance $\mathcal{I} \in s_k$.
    Run g^{th} iteration of GPHH using $\mathcal{I}$.
  end
end
Collect the best dispatching rule from each subpopulation: $\{\omega_1, \omega_2, \ldots, \omega_n\}$.
for $epoch \leftarrow 1 : \mathcal{E}$ **do**
  Randomly select a number $eps \in [0, 1]$
  if $eps \leq \epsilon$ **then**
    Rank the dispatching rules $\{\omega_1, \omega_2, \ldots, \omega_n\}$ using Algorithm 9, which outputs avg. rank $\{\omega_1^{\bar{r}} \ldots, \omega_n^{\bar{r}}\}$
    Determine subpopulation k_{best} whose dispatching rule has best average rank in $\{\omega_1^{\bar{r}} \ldots, \omega_n^{\bar{r}}\}$.
  end
  else
    k_{best} is selected randomly.
  end
  for $g \leftarrow 1 : \gamma$ **do**
    Sample an instance $\mathcal{I} \in s_{k_{best}}$
    Run g^{th} generation of GPHH using $\mathcal{I}$ for the K_{best}th subpopulation.
  end
  Increment $\epsilon \leftarrow \epsilon + \delta_\epsilon$
end
Collect the best dispatching rule from each subpopulation: $\{\omega_1, \omega_2, \ldots, \omega_n\}$.
Output the final pairs of dispatching rules and associated clusters. $\{(\omega_1, s_1), (\omega_2, s_2), \ldots, (\omega_n, s_n)\}$

The ϵ-greedy [129] is a widely used heuristic in the MAB framework. It is very simple to use and in many cases it outperforms more sophisticated methods [129]. Therefore, we employ this heuristic in our first active sampling method, which also provides us with a benchmark. ϵ-greedy method is used for sequential decision problems.

In each round of decision making, it chooses the arm with the highest empirical mean reward with a probability of ϵ and a random arm with a probability of $1 - \epsilon$. Usually, ϵ increases with every round. With this background, we explain the active sampling method based on the ϵ-greedy method using Algorithm 8.

Training

In lines 1-7 of the Algorithm 8, for each cluster in $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$, a sub-population is created (using the usual initialization methods used for GP) and after GP evolution for N_G generations, n dispatching rules pertaining to each cluster are obtained, $\{\omega_1, \omega_2, \ldots, \omega_n\}$ (line 8).

The rest of the GPHH procedure comprises of epochs. In each epoch, γ generations, the ϵ-greedy strategy either picks the 'best' sub population for the next evolution with a probability of ϵ or chooses a random sub-population with a probability of $1 - \epsilon$ (line 11). By *best* sub population, we actually mean the cluster of DJSS instances which shows the maximum promise in evolution of an effective dispatching rule. This is identified using the validation step in line 12.

Using the $\epsilon - greedy$ approach, the most promising sub population is given a chance for evolution with a higher probability (ϵ). This is shown in lines 10-17. We rely on the validation set to quantify the quality of a sub population. Algorithm 9 is employed for this purpose (line 12) which is explained below. The validation procedure ranks the dispatching rules according to their performance on the validation set. The sub-population k_{best} (line 16) which corresponds to the evolved dispatching rule with the highest rank assigned by validation is used for evolution (line 18-21). This

sub population using DJSS training instances from the associated cluster. The value of ϵ is incremented at the end of every epoch till it reaches the value of 1 in the end (line 22). We can observe that our active sampling method identifies those clusters that serve as better candidates for training the dispatching rules and expends comparatively more computational resources while using them for training.

Validation

The aim of the validation step is to evaluate the efficacy of the training cluster in evolution of effective dispatching rules. But the efficacy can be measured only after a dispatching rule has been evolved using that cluster. Since the requirement is to compare the clusters among each other, we use a separate validation set which also consists of clusters of DJSS problem instances. By ranking the evolved dispatching rules on each of the clusters, and determining the average rank, we are therefore able to quantitatively compare the efficacy of training clusters. The validation step of our GPHH framework is explained in Algorithm 9.

The input to this algorithm is the clustered set $\mathcal{V}_c = \{v_1, v_2, \ldots, v_m\}$ and the set of best dispatching rules $\{\omega_1, \omega_2, \ldots, \omega_n\}$, from each sub population. In order to rank these dispatching rules, each of them is evaluated on a set of problem instances from the clusters. In lines 1-11 of the algorithm, on each cluster, for each dispatching rule the sum of the objective values resulting from the evaluation is determined (line 7). $p(v_k, i)$ denotes a DJSS problem instance in the cluster v_k. Consequently, for every cluster each dispatching rules is associated with a value. Therefore, for each cluster, the dispatching rules can be sorted based on the calculated summation values (line 11). Thus each dispatching rule has a rank on each cluster. In lines 13-19 of the algorithm, the average rank of each dispatching rule across the all the clusters is obtained as $\{\omega_1^{\bar{r}} \ldots, \omega_n^{\bar{r}}\}$.

At the completion of training step of the GPHH procedure, the output comprises of a set of dispatching rules each associated with a cluster

from the training set $\{(\omega_1, s_1), (\omega_2, s_2), \ldots, (\omega_n, s_n)\}$, line 25 of Algorithm 8. Now we explain the testing component of our proposed framework using Algorithm 10.

Algorithm 9: Validation for Active Sampling using ϵ-greedy

Input: $\mathcal{V}_c = \{v_1, v_2, \ldots, v_m\}, \mathcal{D} = \{\omega_1, \omega_2, \ldots, \omega_n\}$
Output: Avg. rank of dispatching rules $\{\omega_1^{\bar{r}} \ldots, \omega_n^{\bar{r}}\}$

1 **for** $k \leftarrow 1 : m$ **do**
2 $\mathcal{R}_k \leftarrow \emptyset$
3 $\Sigma_k \leftarrow \emptyset$
4 **for** *each dispatching rule* $\omega \in \mathcal{D}$ **do**
5 $Sum_w^k \leftarrow 0$
6 **for** *each problem instance* $p(v_k, i) \in v_k$ **do**
7 $Sum_w^k \leftarrow$ Sum + DJSSsimulate($p(v_k, i), \omega$)
8 **end**
9 $\Sigma_k \leftarrow \{\Sigma_k, Sum_w^k\}$
10 **end**
11 $\mathcal{R}_k \leftarrow$ Sorting(Σ_k)
12 **end**
13 **for** $z \leftarrow 1 : n$ **do**
14 $r \leftarrow 0$
15 **for** $k \leftarrow 1 : n$ **do**
16 $r \leftarrow r +$ Rank of ω_k in $\mathcal{R}_z$
17 **end**
18 $\omega_z^{\bar{r}} \leftarrow r/|\mathcal{V}_c|$
19 **end**
20 return $\{\omega_1^{\bar{r}} \ldots, \omega_n^{\bar{r}}\}$

Algorithm 10: Testing

Input: Test Instance $\mathcal{I}_t$ and $\{(\omega_1, s_1), (\omega_2, s_2), \ldots, (\omega_n, s_n)\}$

Output: Objective Value : TWT

1 Schedule the first N_j jobs in $\mathcal{I}_t$ using the dispatching rule SPT
2 Determine the feature vector $\hat{f}_t$ for N_j jobs in $\mathcal{I}_t$
3 $d_{min} \leftarrow \infty$
4 $\omega_t \leftarrow \emptyset$
5 **for** $k \leftarrow 1 : n$ **do**
6 $f_k \leftarrow$ feature vector of centroid of s_k
7 **if** *distance*$(f_k, \hat{f}_t) < d_{min}$ **then**
8 $\omega_t \leftarrow \omega_k$
9 $d_{min} \leftarrow$ distance$(f_k, \hat{f}_t)$
10 **end**
11 **end**
12 Use the dispatching rule ω_t for remaining jobs in $\mathcal{I}_t$.
13 Evaluate TWT
14 Return TWT

Testing

Our evolved solution is a set of dispatching rules associated with specific clusters of DJSS instances. When a new test instance $\mathcal{I}_t$ is presented, we must select the most suitable dispatching rule for scheduling. Essentially, we want to determine the association between the shop scenario and corresponding test instance.

We propose to calculate the feature vector of the test instance. This is described in Algorithm 10. The first few jobs in a problem instance are usually ignored for calculation of the scheduling objective, as it is considered as the warm-up period. So we schedule this set of jobs using a standard rule (SPT i.e., shortest processing time) to determine the feature vector of the test instance. This is shown in the lines 1-2 of the algorithm.

The feature vectors are extracted using the procedure developed in Section 4.2.1.

After the feature extraction, the Euclidean distance of this feature vector from feature vectors of the centroids of the clusters in $\mathcal{S}_c$ is evaluated. Then the dispatching rule corresponding to the cluster with minimum distances is chosen. This is shown in the lines 5-11 of the algorithm.

If we consider the computational complexity of GPHH, then it is determined by the number of simulations of the DJSS environment required to determine the fitness of each individual in the population, which is equivalent to

$$N_S \times G$$

where N_S is the population size and G is the number of generations.

For the proposed approach, the number of simulations are incremented by the those required in the validation stage. Therefore, the computational cost has three components. Firstly, the simulations required as in lines 2-7 of Algorithm 8, secondly the simulations corresponding to the lines 18-21 pertaining to the evaluations within an epoch and the third component is that of validation (line 12). The summation of the three components leads to

$$(N \times N_G) + (\mathcal{E} \times \gamma \times \frac{N}{n}) + (\mathcal{E} \times \sum_k |v_k| + \mathcal{D} \times \sum_k |v_k|)$$

where $|v_k|$ is the number of problem instances in the validation set v_k and N is the population size. Moreover, $(\mathcal{E} \times \sum_k |v_k|)$ is arrived at by considering the fact that only one dispatching rule in $\mathcal{D}$ (see Algorithm 9) is changed at the end of an epoch.

It is important that the computational resource utilized by our proposed approach does not exceed that of the standard GPHH. This can be easily achieved by tuning the values of $\mathcal{E}$ and N.

The ϵ-greedy approach is simple and one of the first methods which is usually applied in the multi-armed bandits context [90]. However, for larger problems this technique is inefficient [228]. This is because for larger

problems, the multi-armed bandit formulation results in large number of arms and ϵ-greedy is not efficient for dealing with, particularly because it remembers on the best arm. In the context of our case, as we mentioned earlier, the number of possible workshop scenarios is actually infinite and therefore we should explore a large number of clusters of DJSS instances to evolve scenario-specific rules. Clearly, the ϵ-greedy method for active sampling is not enough for this task. Therefore, in the next section we present a new active sampling approach based on Bayesian optimization to address the aforementioned limitations.

4.2.4 GPHH with Active Sampling using Gaussian Process Bandits

Bayesian optimization [37] employs a sequential strategy to do black box optimization by building a probabilistic model of the function to be optimized and then exploiting this model to decide on which point in the input space the next evaluation of the function should be performed.

$$\max_{x \in \mathcal{A}} f(x) \tag{4.1}$$

In Eq. 4.1 we show a function $f(x)$ which needs to be maximized on the feasible set $\mathcal{A}$. f is a function which is very expensive to evaluate. In other words, it is an optimization problem with an expensive objective function.

For the prior distribution required by the probabilistic model, Gaussian processes [195] are used due to their tractability and flexibility. The strength of Gaussian processes in expressing rich distribution of functions is dependent on the choice of covariance functions (or kernel). Essentially the covariance functions define the Gaussian process' behavior. A frequently used kernel function is the exponential kernel function in Eq. 4.2.

$$\mathcal{K}(\mathbf{x_1}, \mathbf{x_2}) = exp\left(\frac{||\mathbf{x_1} - \mathbf{x_2}||^2}{2\sigma^2}\right) \tag{4.2}$$

The kernel function assigns the covariance between the function evaluations on the points $\mathbf{x_1}$ and $\mathbf{x_2}$.

Methodologically, Bayesian optimization closely relates to active learning and multi-armed bandits [37]. In order to solve the problem of multi-armed bandit, the tools of Gaussian processes have been shown to be very useful [217], particularly when the optimization involves a noisy environment and an unknown function which is expensive to evaluate, thus requiring minimization of sampling from the input space. The objective is to find the maximum (or minimum) of the function or in other words find the point where the function evaluation gives maximum reward. After a set of trials, a Gaussian process [195] is fit on the data points obtained. More formally, say for n_0 points the function f was observed. The posterior probability distribution is updated on f using this data. This gives a model of the function with the posterior mean and confidence intervals.

Now a mechanism is required to identify the next point which should be explored; this is determined by an *acquisition* function (also called a utility function). The acquisition functions use the Gaussian process model to acquire the next point to be explored. For example, the points for which the confidence interval is high are associated with most uncertainty, and such a point when sampled will yield maximum information gain (maximum exploration). If a point is sampled from a region where the mean value is maximum, the immediate reward is expected to be high (maximum exploitation). The acquisition function takes both the mean and the variance into consideration. Thus, the acquisition function is basically addressing the exploration vs exploitation dilemma.

More formally, let x_n be the point returned by the acquisition function, then observe $f(x_n)$ and update the posterior distribution after including the new observation. This process is iterated till the budget of function evaluations is exhausted. The final solution is the point for which $f(x)$ was maximum or the the point where the posterior mean was maximum.

Recently, Gaussian process-Upper conference bound (GP-UCB) [217]

has been considered as a useful acquisition function. They use the upper confidence bounds in terms of information gain from the sampling to minimize regret over the course of optimization. The GP-UCB rule is given in Eq. 4.3.

$$\mathbf{x_t} = \underset{\mathbf{x}\in D}{\operatorname{argmax}}\ \mu_{t-1}(\mathbf{x}) + \sqrt{\beta_t}\sigma_{t-1}(\mathbf{x}) \tag{4.3}$$

where μ and σ are mean and variance of the posterior GP distribution over the optimization function. This method has been supported by sound theoretical and empirical analyses [217].

The methods described above is expected to be more efficient than ϵ-greedy especially when the input space is larger. This is because ϵ-greedy does not have a memory of the exploration already done where as the methods described above exploit the information already gained through sampling.

With this background, we now present an active sampling algorithm based on Gaussian process bandits (GPB) for our proposed GPHH framework. When we considered active sampling using the ϵ-greedy heuristic, the number of sub populations used for evolving dispatching rules was equal to the number of clusters in the training set, which is also same as the number of dispatching rules evolved after a run. Now we consider a much higher number of clusters and consider evolving a larger number of rules, but it is not feasible to increase the number of sub populations due to computational issues. Consequently, for this multi-armed bandit problem, the number of arms is much higher. The GPB active sampling approach is presented in Algorithm 11 which we describe below.

In line 1 of Algorithm 11, we randomly select p of clusters from $\mathcal{S}_c$, the training set and assign it p subpopulations each. In lines 2-7, we obtain a set of p dispatching rules through GP evolution. In line 8, the initial solution is obtained which is a set of pairs of the dispatching rules and the feature vector of DJSS instance corresponding to the centroid of each of the p clusters. These steps are similar to the previous active sampling

algorithm. Furthermore, in line 9, $\mathcal{C}_*$ is assigned the set of feature vectors of DJSS instances corresponding to each of the centroids of $\mathcal{S}_c$. The total number of clusters in $\mathcal{S}_c$ is much higher than p. For any cluster $s_a \in \mathcal{S}_c$, $\mathcal{C}(s_a)$ stands for the feature vector of the centroid of s_a.

Now, after exploring a limited set of clusters initially and obtaining the initial solution Δ, the question is which cluster to explore next. This is addressed by our **validation** procedure which we describe using Algorithm 12. The input to this algorithm is the current solution Δ, the training $\mathcal{S}_c$ and the validation $\mathcal{V}_c$ clusters. For each of the evolved dispatching rule in the current solution Δ, its rank for each of the clusters in $\mathcal{V}_c$ is assigned. This is done in the lines 1-12. After this, the average rank is calculated for each of the dispatching rule. These steps are exactly the same as in the validation procedure with ϵ-greedy. But ϵ-greedy explored only a limited set of clusters, and its main goal was to identify the cluster which was best for training among them. But now, the number of clusters which we consider is much higher, and a dispatching rule is not associated with each of them. Therefore, we need a method which can approximate the ranking of the dispatching rule pertaining to a cluster without the expensive evolution.

Using the concepts of Bayesian optimization described earlier, we model the approximation of the average rank of a dispatching rule on a cluster using Gaussian processes. In line 20 of Algorithm 12, we build a dataset D by collecting the pairs of the average rank of the dispatching rules ($\omega_z^{\bar{r}}$ and the feature vector $\mathcal{C}(s_z)$), of clusters (centroid).

Using this dataset, in line 22, the Gaussian process regression outputs the mean and variance of the posterior distribution of function evaluations on each cluster. In other words, without actually evolving the dispatching rule for a cluster s_x the approximate average rank of the rule is determined using the features of the cluster. After this step, the acquisition function based on Eq. 4.3 is used to determine the next cluster which should be explored.

Algorithm 11: GPHH with Active Sampling using Gaussian Process Bandits approach

Input: $\mathcal{S}_c$,$\mathcal{V}_c$

Output: Set of dispatching rules associated with clusters in $\mathcal{S}_c$

Create p subpopulations. Randomly choose p clusters from $\mathcal{S}_c$.
for $k \leftarrow 1 : p$ **do**
  for $g \leftarrow 1 : N_G$ **do**
    Sample an instance $\mathcal{I} \in s_p$.
    Run g^{th} iteration of GPHH using $\mathcal{I}$.
  end
end
Collect the best dispatching rule from each subpopulation:
$\Delta \leftarrow \{(\omega_1, \mathcal{C}(s_1)), (\omega_2, \mathcal{C}(s_2)), \ldots, (\omega_n, \mathcal{C}(s_p))\}$.
$\mathcal{C}_* \leftarrow \{\mathcal{C}(s_1), \ldots, \mathcal{C}(s_p)\}$, where $\mathcal{C}(s_a)$ is the feature vector of centroid of s_a
$s_x \leftarrow$ GP-UCB-Validation($\Delta, \mathcal{V}_c, \mathcal{C}_\mathcal{S}$) using Algorithm 12.
for $epoch \leftarrow 1 : \mathcal{E}$ **do**
  distance $\leftarrow \infty$
  $\mathcal{C}(s_x) \leftarrow$ feature vector of centroid of s_x
  for $c \in \mathcal{C}_*$ **do**
    dist $\leftarrow$ euclidean distance($\mathcal{C}(s_x), c$)
    if *distance* $<$ *dist* **then**
      distance = dist
      $K \leftarrow$ index of subpop. corresponding to c
    end
  end
  for $g \leftarrow 1 : \gamma$ **do**
    Sample an instance $\mathcal{I} \in s_x$
    Run g^{th} generation of GPHH using $\mathcal{I}$ for the Kth subpopulation.
  end
  $s_x \leftarrow \omega_x \leftarrow$ best dispathching rule from subpop. K
  $\Delta \leftarrow \{\Delta, (\omega_x, \mathcal{C}(s_x))\}$
  GP-UCB-Validation($\Delta, \mathcal{V}_c, \mathcal{C}_\mathcal{S}$) using Algorithm 12.
end
return Δ

Algorithm 12: Validation using GPB approach

Input: $\mathcal{S}_c$,$\mathcal{V}_c$,Δ

Output: s_x, next selected cluster from $\mathcal{S}_c$

1 **for** $k \leftarrow 1 : m$ **do**
2 $\mathcal{R}_k \leftarrow \emptyset$
3 $\Sigma_k \leftarrow \emptyset$
4 **for** *each dispatching rule* $\omega \in \Delta$ **do**
5 $Sum_w^k \leftarrow 0$
6 **for** *each problem instance* $p(v_k, i) \in v_k$ **do**
7 $Sum_w^k \leftarrow$ Sum + DJSSsimulate($p(v_k, i), \omega$)
8 **end**
9 $\Sigma_k \leftarrow \{\Sigma_k, Sum_w^k\}$
10 **end**
11 $\mathcal{R}_k \leftarrow$ Sorting(Σ_k)
12 **end**
13 $D \leftarrow \emptyset$
14 **for** $z \leftarrow 1 : n$ **do**
15 $r \leftarrow 0$
16 **for** $k \leftarrow 1 : n$ **do**
17 $r \leftarrow r$ + Rank of ω_k in $\mathcal{R}_z$
18 **end**
19 $\omega_z^{\bar{r}} \leftarrow r/|\mathcal{V}_c|$
20 $D \leftarrow \{D, (\omega_z^{\bar{r}}, \mathcal{C}(s_z))\}$
21 **end**
22 $\{(\mu_1, \sigma_1), \ldots, (\mu_p, \sigma_p), \ldots, (\mu_{p+epoch}, \sigma_{p+epoch})\} \leftarrow GaussianProcess(D)$
23 Choose $\mathbf{s_x} = \underset{\mathbf{s_y} \in D}{\operatorname{argmax}} \mu_i(\mathbf{s_y}) + \sqrt{\beta_{epoch}}\sigma_i(\mathbf{s_y})$
24 return s_x

Once the next cluster to be explored is identified, the next epoch, line 11 in the Algorithm 11 is started. Now, we need to determine which sub-

population to use for this evolutionary epoch. We do this by finding the Euclidean distance between the feature vector of the centroid of the chosen cluster s_x with the feature vectors in $\{\mathcal{C}(s_1), \ldots, \mathcal{C}(s_1)\}$ and the subpopulation corresponding to shortest distance is chosen (lines 12-20). After the GP evolution, lines 21-24, the best dispatching rule from the subpopulation is identified and the solution Δ is updated (lines 25-26). After the completion of each epoch, the next round of validation steps commences. Thus the number of dispatching rules in the solution Δ is continuously incremented as the newer clusters are actively sampled. Also the value of β_{epoch} is decremented in every epoch [217]. This essentially means that the algorithm will focus more and more on exploitation than exploration.

The testing component of this approach is exactly same as for the previous GPHH approach with active sampling using ϵ-greedy, described in Algorithm 10.

4.3 Experiment Design

Since the definition of dynamic job shop scheduling problems which we considered is the same throughout the thesis, we do not repeat the definition here. We considered weighted total tardiness as the objective for DJSS in this chapter. Since tardiness is evaluated using due date, the additional parameter makes the problem more complex. Thus offering better scope for evaluating the proposed methods. This objective is frequently considered in literature [163], as the scheduling objective. Moreover, since our feature extraction considers due date factor, it makes sense to focus on tardiness as the scheduling objective.

$$TWT = \sum_{j=1}^{N} w_j \times \max(C_j - d_j, 0),$$

where C_j is the completion time, d_j is the due date and w_j are the weights of a job j.

In order to simulate DJSS we use a discrete event simulation system (Jasima) [93]. For each simulation, 500 jobs in the beginning are considered as warm-up and the objective value is determined for the following 2000 jobs. Referring Algorithm 10, in line 1, we use these warm up jobs for extracting test instance features. Here, the number of warm-up jobs $N_j = 500$. This is adopted in many of the existing works e.g. [156].

The estimated processing time of an operation, i.e. $p_{j,i}$, is usually different from the actual processing time ($p'_{j,i}$) due to uncertainty. $p'_{j,i}$ is known only at the time of realization on the machine. Also $p'_{j,i} > p_{j,i}$ is a practical assumption which is supported by the relationship below [117]:

$$p'_{j,i} = (1 + \theta_{j,i})p_{j,i}, \theta_{j,i} \geq 0.$$

θ follows exponential distribution.

Table 4.3: DJSS simulation parameters

Simulation parameter	Values
Processing time range	[0,49],[20,69]
Uncertainty scale (β)	$\{0.2, 0.4\}$
Due date tightness	$\{1.5, 2.5\}$
# operations per job	$\{8, 10\}$

DJSS problem instances are generated using the parameters specified in Table 4.3. We had explained the importance of these parameters in Section 4.2. Two levels are considered for each of the parameter towards defining different shop scenarios. In particular, we chose the number of operations per job as 8 and 10, which are close, as we would like our approach to be sensitive to subtle differences in the shop scenarios. These four pairs of parameters can be used to simulate 16 types of jobs. A DJSS problem instance is composed of 3 types of jobs at a time. The ratio of job types arriving at shop is $1 : 1 : 1$. The unique combinations *with* repetition results in 816 possible configurations. For each of these configurations, 60

DJSS problem instances are created which are then evenly distributed into training, validation and test sets. Our preliminary study showed that a larger set of instances do not show any advantage. Because if we increase the number of instances then the size of a particular cluster will increase, which does not provide any edge to the algorithm.

Table 4.4: Parameter values

		GPHH	ϵ-greedy	GPB
1	# sub population	-	4	4
2	population size	1500	600×4	600×4
3	#generations	200	100	100
4	#clusters validation	-	20	20
5	#instances used per validation cluster	-	20	20
6	#clusters training	-	4	500
7	#generations per epoch	-	10	10

We present our choice of parameters for the different approaches in Table 4.4. For active sampling using GPB, the number of training clusters is 500 which is much higher than the 4 clusters considered with ϵ-greedy. For just two levels of each parameter value, a size of 500 is appropriate considering there are 816 shop configurations. It is enough to challenge our algorithm while each cluster essentially maps to more than one of the 816 configurations. The number of sub-populations for both the algorithms is 4. The size of populations have been chosen to ensure that the computational budget of the different approaches are such that none of them are not in any advantage with respect to computational resources.

The number of instances in the validation clusters is 20, which is arrived at by doing some preliminary investigations. For this preliminary investigation, we evolved a set of 5 dispatching rules by randomly choosing a 5 clusters from $\mathcal{H}$, the set for hyper-parameter optimization. Then

we tried to rank them using a different number of clusters for validation. We found that for a lower number of clusters, the ranks were not consistent, because when the number is low more shop scenarios correspond to same cluster. When the number of clusters was closer to 20, we observed consistency in the ranks.

The number of generations per epoch is set to 10, again using preliminary investigations. We observed that usually within just 20 generations the evolved dispatching rule becomes effective. Since the subpopulation which we use already has undergone evolution, 10 generations in an epoch are sufficient to learn the characteristics of a newly sampled cluster of DJSS instances. The performance of proposed approaches is compared with standard GP (SGP) which uses a population size of 1500 and 200 generations for evolution. This ensures similar computation budget for all algorithms for a fair comparison.

The GP System

Now , we describe the genetic programming system considered in our experiments. The populations are randomly initialized using ramp half-and-half method. The set of terminals and functions which we considered are listed in Tables 4.5 & 4.6 respectively. The protected division in Table 4.6 outputs 1 when divided by 0. The mutation, crossover and tree depth are 0.1, 0.85 and 6 respectively [161].

Note that the terminal set is different from the experiments in Chapter 3, because the JSS objective considered is different. Similarly, the parameter settings for GP system are different to ensure that the computational cost of our proposed approaches is same for fair comparison.

Hyper-parameter Optimization

Hyper-parameter optimization is an important step for machine learning algorithms to do well. When more than one parameters are involved, it is

Table 4.5: Terminal Sets for GP.

Terminal Set	Meaning
PT	Processing time of operation
RO	Remaining operations for job
RJ	Ready time of job
RT	Remaining processing time of job
RM	Ready time of machine
DD	Due date
W	Job weight
ERC	Ephemeral Random constant

Table 4.6: Function Set for GP.

Function Set	Meaning
$+$	Addition
$-$	Subtraction
$*$	Multiplication
$/$	Protected Division
Max	Maximum
Min	Minimum

not a trivial job to identify the optimal parameters for an algorithm [23, 24]. The choice of parameters is dependent on the problem. In this work we use RBF kernel [195] for the Gaussian processes. Two parameters *sigma* (σ) and *noise* (η) are required for this algorithm, Equation 4.4. *noise* (η) is not a part of the kernel but is used by the Gaussian process model to accommodate noise in training data.

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = exp\left(\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\sigma^2}\right) + noise(\eta^2) \tag{4.4}$$

In order to find a good set of parameters we design an experiment

which tries to emulate the optimization problem. We present our approach below to optimize these parameters.

We utilize the set of DJSS problems $\mathcal{H}$ which has much smaller number of instances compared to the training set $\mathcal{S}$. We apply the feature extraction and clustering approach, explained in Section 4.2.1, to $\mathcal{H}$ and obtain $\mathcal{H}_c$. Say the number of clusters in $\mathcal{H}_c$ is h. We chose $h = 30$ for this experiment. This choice was done to keep the computational cost to minimum while ensuring that the hyper-parameter optimization is effective. For each of these DJSS problem instances cluster, we evolve dispatching rules which uses the training instances drawn from the cluster. We use the same terminal set and the function set, and other set of parameters for genetic programming, and the total number of generations is 50 and population size is 500. Choosing a larger cluster size would make the hyper-parameter optimization quite expensive as we would need to evolve more rules. We denote the evolved set of rules by $\mathcal{D}_{\mathcal{H}}$, which comprise of the best rule from each population.

We assign ranks to the dispatching rules in $\mathcal{D}_{\mathcal{H}}$ using Algorithm 9. The average rank of the dispatching rules is essentially the rank of the corresponding clusters in $\mathcal{H}_c$. Consequently, each centroid of a cluster in $\mathcal{H}_c$ can be mapped to a value. We construct this dataset, denoted by $\{\mathcal{H}_c, \mathcal{R}_{\mathcal{H}}\}$.

We divide $\{\mathcal{H}_c, \mathcal{R}_{\mathcal{H}}\}$ into training and test sets and use the Random grid search [23] method for hyper-parameter optimization. In order to determine the range of the parameters, we initially apply manual search and once we decide on a good range, we apply random search with cross validation to find a near optimal set of parameters. The manual search for optimal parameters narrowed it down to the following ranges; $\sigma \in [-4, 4]$ and $\eta \in [-2, 2]$. We chose a random search method because on the one hand it is computationally efficient and also is able to determine near optimal parameters [23].

The hyper-paramter optimization yielded 0.8 and 0.2 for σ and η respectively.

4.4 Results and Discussions

In this section, we describe our results. In order to perform comparisons, 30 independent runs of a method are used to produce 30 sets of solutions. For each method, the evolved solutions are compared over each of the 30 problem instances in a test subset which are denoted using Roman numerals. With a significance level of 0.05 Wilcoxon-rank-sum test is used to compare the performance. The results are summarized in Tables 4.7-4.9.

A cell in the table consists of a triplet which must be read as $[win\text{-}draw\text{-}lose]$. As an example, the first cell in Table 4.7 is [7-23-0]. This means that for the training set I, out of the 30 problem instances, the method ϵ-greedy significantly outperformed standard GP in 7 instances (win) and there is no significant difference in 23 (draw). On none of the test instances standard GP outperforms ϵ-greedy approach.

We obtained the testing sets from $\mathcal{T}_c$ which is obtained using feature extraction and clustering explained in Section 4.2.1. We obtained 30 such testing sets using that methodology.

Firstly, we discuss the comparison results between our GPHH approach with active sampling based on ϵ-greedy, which we denote as ϵ-greedy in short. This is presented in Table 4.7. The performance of ϵ-greedy is mixed. In many of the test sets like I, II, III, IV, VI, XIII, XVI, XVII, XXII, etc. this active sampling method has succeeded in giving improved results. These cells are marked in green. We color a cell in green if at least on 5 instances the proposed method outperforms and also does not under perform on a large number of instances (allowed up to 5). On the other hand, if it under performs on more than 5 instances, the cell is marked in orange.

The reason for the poor performance of ϵ-greedy in some of the test sets could be attributed to the fact that in order to evolve more specific rules, some of the characteristics on other clusters are not taken into account. This variation in performance is indicative of the fact that scenario-specific rules, which are trained using specific clusters of DJSS training instances,

do work well on those scenarios but are outperformed by a general rule which is trained over all the scenarios. Clearly, the evolved rules suffer from overfitting on DJSS instances corresponding to some shop scenarios.

Table 4.7: ϵ-greedy vs. GPHH

I	II	III	IV	V	VI	VII	VIII
[7-23-0]	[4-25-1]	[4-26-0]	[1-26-3]	[4-26-0]	[5-25-0]	[9-14-7]	[3-25-2]

IX	X	XI	XII	XIII	XIV	XV	XVI
[3-26-1]	[2-10-18]	[4-18-8]	[1-3-26]	[5-24-1]	[5-23-2]	[0-8-22]	[6-24-0]

XVII	XVIII	XIX	XX	XXI	XXII	XXIII	XXIV
[4-26-0]	[1-5-24]	[3-26-1]	[5-25-0]	[5-24-1]	[14-5-11]	[5-25-0]	[4-25-1]

	XXV	XXVI	XXVII	XXVIII	XXIX	XXX	
	[0-10-20]	[2-2-26]	[1-1-28]	[15-15-0]	[6-17-7]	[4-25-1]	

By incorporating a larger number of clusters we expect to alleviate this problem in the GPHH approach with active sampling using Gaussian Process Bandits which we call as GPB method, in short. The comparison results between GPB method and the standard GP are presented in Table 4.8. Our proposed method has significantly outperformed the standard GP on almost all the test sets. The improvement is quite consistent. Once again we have marked the cells in green if the number of instances on which our algorithm outperforms is at least five.

Therefore, by developing an algorithm which can actively sample large number of clusters of DJSS training instances and address the limitation of ϵ-greedy significant improvement in performance can be achieved. Furthermore, considering the fact that we were successful in developing the active sampling techniques as part of our objective, it is worthwhile to do a comparative run-time analysis of the two algorithms, by considering a

Table 4.8: GPB vs. GPHH

I	II	III	IV	V	VI	VII	VIII
[5-25-0]	[4-26-0]	[5-25-0]	[3-26-1]	[5-24-1]	[5-25-0]	[7-21-2]	[5-25-0]

IX	X	XI	XII	XIII	XIV	XV	XVI
[4-23-3]	[5-24-1]	[5-25-0]	[7-23-0]	[7-22-1]	[6-21-3]	[5-25-0]	[5-25-0]

XVII	XVIII	XIX	XX	XXI	XXII	XXIII	XXIV
[4-26-0]	[8-21-1]	[3-25-2]	[6-24-0]	[6-24-0]	[6-24-0]	[2-26-2]	[3-24-3]

	XXV	XXVI	XXVII	XXVIII	XXIX	XXX	
	[3-25-2]	[4-24-2]	[4-24-2]	[6-24-0]	[5-24-1]	[4-26-0]	

Table 4.9: GPB vs. ϵ-greedy

I	II	III	IV	V	VI	VII	VIII
[4-26-0]	[4-26-0]	[6-24-0]	**[14-14-2]**	[1-25-4]	[4-26-0]	[9-16-5]	[4-22-4]

IX	X	XI	XII	XIII	XIV	XV	XVI
[11-19-0]	**[18-12-0]**	[2-28-0]	**[18-12-0]**	[6-24-0]	[4-25-1]	**[14-16-0]**	[5-24-1]

XVII	XVIII	XIX	XX	XXI	XXII	XXIII	XXIV
[10-18-2]	**[17-12-1]**	[4-26-0]	[4-26-0]	[3-27-0]	**[15-14-1]**	[3-27-0]	[2-28-0]

	XXV	XXVI	XXVII	XXVIII	XXIV	XXX	
	[15-15-0]	**[16-13-1]**	**[14-14-2]**	**[11-15-4]**	[4-21-5]	[4-26-0]	

combination of different parameters. It might lead to some insights and improve our understanding of these methodologies further. We will consider this in future work.

Finally, we also compare the performance of ϵ-greedy with GPB and as expected we observe significant improvement particularly on those test sets where ϵ-greedy fared poorly against the standard GP. Such test sets are marked in **bold**.

We have included more detailed results in Appendix A.

Further Analysis

In this section, we try to determine the characteristics of the different clusters of DJSS problem instances and map them to the possible shop scenarios and then reflect upon the reasons for the observed results.

We represent the characteristics of a DJSS instance by using the matrix representation shown below. The matrix $\mathcal{G}$ has three rows such that each row corresponds to a job type. A job type is characterized by the parameters (Table 4.3) like due date tightness, number of operations in a job etc. as explained in detail in Section 4.2.1. Since we used two levels for each parameter, they are represented by $+$ and $-$ for high and low respectively. So, for the first row in the matrix, the job type corresponds to a job with a less tight due date, more number of operations per job, higher processing times and with higher uncertainty in the processing times. The δ column denotes delay ratio. Similarly, the other two rows characterize the other two job types.

When dispatching rules are evolved using GPHH by training in DJSS instances with such characteristics as in $\mathcal{G}$ we denote these dispatching rules by $D^{\mathcal{G}}$.

$$\mathcal{G} = \begin{bmatrix} \mathbf{d.d.} & \mathbf{\#op} & \mathbf{p.t.} & \boldsymbol{\delta} \\ - & + & + & + \\ + & + & + & - \\ + & + & - & - \end{bmatrix}$$

By analyzing the different runs of GPHH approach with active sampling using Gaussian process bandits, we found that for 22 out of 30 runs,

the dispatching rules evolved using DJSS instances with characteristics similar to $\mathcal{G}$ ranked highly on a lot of clusters on the validation set. Note that the clusters in validation set represent a wide range of shop scenarios. This is not surprising, because this configuration with tighter due dates and a higher number of operations is harder. Therefore, by training the rules on such instances we expect good performance. Even though obvious, this outcome of the analysis adds credibility to our approaches.

$$\mathcal{B} = \begin{bmatrix} \mathbf{d.d.} & \mathbf{\#op} & \mathbf{p.t.} & \boldsymbol{\delta} \\ - & - & - & + \\ - & + & - & - \\ - & - & - & - \end{bmatrix}$$

Similarly, the matrix $\mathcal{B}$ represents a class of DJSS instances which have usually evolved dispatching rules with poor ranks on the validation clusters. Again, the reasons are similar, these instances are not hard when compared to $\mathcal{B}$. We use the notation

$$D^{\mathcal{G}} \gg D^{\mathcal{B}}$$

to denote this observation.

As already mentioned, these observations are not surprising and we are more interested in identifying those DJSS training instances which resulted in scenario-specific rules. In order to identify such instances, we tried to find those dispatching rules which do very well on some validation clusters but not so good on the others. One such training cluster which was consistently responsible for evolution of dispatching rules with such characteristics is $\mathcal{Q}$.

$$\mathcal{Q} = \begin{bmatrix} \mathbf{d.d.} & \mathbf{\#op} & \mathbf{p.t.} & \boldsymbol{\delta} \\ - & + & - & + \\ - & + & - & + \\ + & - & + & - \end{bmatrix}$$

$\mathcal{Q}$ represents a class of DJSS instances in which the characteristics of the job types is quite dissimilar. It represents a shop scenario which deals with arrival of jobs with higher number of operations but lower due date tightness and shorter processing times and of jobs with low number of operations, longer processing times but tighter due dates. This is a complex shop scenario. We compare the dispatching rules evolved using $\mathcal{Q}$, $D^{\mathcal{Q}}$ on two different validation clusters represented by $\mathcal{X}$ and $\mathcal{Z}$.

We observed that the rank of $D^{\mathcal{Q}}$ on $\mathcal{X}$ was generally poor compared with $D^{\mathcal{G}}$. An example of the dispatching rules which show such behaviour is discussed later. We denote this observation as

$$D^{\mathcal{G}}(\mathcal{X}) > D^{\mathcal{Q}}(\mathcal{X})$$

The reason for this observation is that the DJSS training instances with characteristics like those of $\mathcal{X}$ are not very complex, e.g. all the job types have similar due date tightness. Therefore, a dispatching rule evolved over $\mathcal{G}$, can clearly perform better.

$$\mathcal{X} = \begin{bmatrix} \mathbf{d.d.} & \mathbf{\#op} & \mathbf{p.t.} & \boldsymbol{\delta} \\ - & + & - & - \\ - & - & - & + \\ - & - & - & + \end{bmatrix}$$

Now the interesting observation is that $D^{\mathcal{Q}}$ does better than $D^{\mathcal{G}}$ on many of the clusters whose DJSS instances are similar to $\mathcal{Z}$. The matrix $\mathcal{Z}$ shows a class of DJSS problem instance which has again got dissimilar job types.

$$\mathcal{Z} = \begin{bmatrix} \mathbf{d.d.} & \mathbf{\#op} & \mathbf{p.t.} & \boldsymbol{\delta} \\ - & + & + & + \\ + & - & - & + \\ + & - & - & + \end{bmatrix}$$

$\mathcal{Z}$ and $\mathcal{Q}$ are thus similar in this manner which leads to

$$D^{\mathcal{G}}(\mathcal{Z}) > D^{\mathcal{Q}}(\mathcal{Z})$$

We present an example of $D^{\mathcal{Q}}$ and $D^{\mathcal{G}}$ below. an interesting observation is that the number of terminals RO, i.e., the number of remain operations for job is quite. This is understandable, because if you look at the matrix $\mathcal{Q}$, one of the differences between the job types is in their number of operations. The higher number of the terminal RO in dispatching rules in $D^{\mathcal{Q}}$ indicate that it has been able to capture the characteristic of the shop scenario. This is supported by our observation that for the 30 runs, the total count of terminal RO for $D^{\mathcal{Q}}$ was 182 and for $D^{\mathcal{G}}$ it was 153 across the dispatching rules.

Dispatching Rule 4.1: $D^{\mathcal{Q}}$

```
(* (+ (* (* (Max RJ RO) (Max RJ RT)) (If
(/ RO DD) (If RJ W PR) (/ RJ W))) (Min (Min
W (+ 0.93 W)) (- (* RT RO)
(Min W W)))) (- (/ (* (Max RJ RO) (Max RJ
RT)) (- PR (+ 0.65 0.24)))  (/ (+ (+ RJ RM)
(Max RO RT)) (* (+ DD W) (If RO RT DD))))) RO
```

Dispatching Rule 4.2: $D^{\mathcal{G}}$

```
(Max (+ (/ (+ (- RO RT) (Max RT RJ)) (/ (/
W DD) (- RT DD))) (+ (/ W RT) (* RJ DD)))
(/ (Min (Max (/ 0.17 DD) (Max
0.43 0.88)) (Min (/ W PR) (If PR RO 0.52)))
(Min DD (* (Max RM 0.65) (If RJ RJ DD)))))
```

4.5 Chapter Summary

The goal of this chapter was to incorporate active learning techniques into GPHH approach toward evolving scenario-specific dispatching rules. To

achieve this goal, firstly, we developed a feature extraction and clustering methodology to map DJSS training instances with the dynamic shop characteristics. Then we identified the exploration versus exploitation dilemma in sampling good training instances, representative of complex shop scenarios, while evolving the dispatching rules using GPHH.

We developed a new GPHH framework, which enables developing active sampling techniques to tackle the exploration versus exploitation dilemma. Using this framework, we developed an active sampling method for GPHH using ϵ-greedy heuristic. In order to alleviate the limitations of ϵ-greedy method we developed another active sampling method for GPHH using Gaussian process bandits.

Through our experiments, we showed that our proposed active sampling methods for GPHH significantly outperforms the existing approach. Furthermore, the active sampling approach based on GPB is more robust and outperforms ϵ-greedy method. We also analyzed the characteristics of shop scenarios which led to our observations and our findings provide more understanding of our results.

In this chapter, we considered only one objective for DJSS. But in practical production systems, more than one objective are considered while scheduling. When we consider more than one objective, the computational challenges scale up. In order to deal with this, parallel evolutionary algorithms are frequently employed. Moreover, the exploration versus exploitation dilemma which we discussed in this chapter, become more tricky. In our next chapter, we will try to address these issues and develop active sampling approach for multi-objective DJSS problems.

Chapter 5

Active Sampling Heuristics for Multi-objective DJSS Problems Using Island Based Parallel Genetic Programming

5.1 Introduction

A study of the literature shows that many existing research works feature the use of sequential scheduling methods and focus primarily on optimizing a single performance objective, such as the makespan or the total tardiness. In practice, however, it is frequently shown [57, 65] that multi-objective optimization is essential for successful job shop scheduling especially when useful schedules must meet multiple performance criteria. Moreover the objectives to be optimized are usually conflicting in nature. As a result, not a single optimal solution but a collection of *Pareto optimal solutions* will need to be identified in order to properly schedule jobs in a job shop. Pareto ordering [57] is a mathematical concept used to define the optimal solutions of a multi-objective optimization problem. This

uses the concept of domination which compares two solutions for a multi-objective problem. Considering a bi-objective optimization problem, a solution will dominate another solution if it is better in at least one of the objective functions and not worse in the other. For a multi-objective optimization problem Pareto optimal solutions are the set of solutions which are not dominated by any other solution [57].

In Chapter 4, we had shown that for GPHH to be effective for a single objective DJSS problem, it is important to use large training instances which are representative of complex shop scenarios. Furthermore, when we consider multiple objectives the importance of using diverse and large training set representing the complex scenarios is further compounded. For example, makespan and total tardiness are two frequently considered conflicting objectives. Minimizing the makespan results in high throughput where as minimizing tardiness requires jobs to be not very late. A conflicting scenario arises when a set of jobs with long processing times but shorter deadline compete with a set of jobs with shorter processing times and longer deadlines. For higher throughput, the shorter jobs must be completed first as against the longer jobs which adversely affects the tardiness. For evolving good dispatching rules, it is important to present the evolutionary system with training instances which capture scenarios highlighting all such conflicts amongst the objectives, under different shop scenarios.

We highlight the importance of considering large number of shop scenarios for multi-objective scheduling problems arising due to variability and uncertainty in a shop environment with a more practical example [104] of printing industry, where planning and scheduling of print jobs is an important problem. The printing industry considered here is a part of the more broader documentation management services. In general, the printing jobs show patterns, for example, monthly credit card statements, marketing materials, etc. have similar print characteristics and recurring nature. The nature of these patterns must be considered in the schedul-

ing routine. Prior to applying the scheduling routine, a simulation is run to determine the size of the print jobs which essentially gives the value of the estimated processing times. For this print shop, machine down-time and operator's breaks are a major source of uncertainty in processing times. This uncertainty has the ability to disrupt the characteristics of the patterns mentioned above. Consequently, it is equally important to also consider the impact of this **uncertainty** in the multi-objective DJSS problems. For example, due to change in weather conditions, an older printer might require more time for printing of a specific set of medium sized jobs due to heating but having little impact on relatively smaller jobs. For a scheduling problem considering both total tardiness and makespan, this pattern, which arises due to uncertainty in processing times, could be a potentially conflicting scenario if the due dates of the effected jobs is tight. But considering the added effect of uncertainty along with other patterns the difficulty in identifying potentially good training instances increases.

Therefore, in order to evolve effective dispatching rules to solve multi-objective DJSS problems under uncertain shop environments, it is even more important for the GPHH approach to consider active sampling of DJSS instances. In the previous chapter, we had successfully employed the active learning methods, in particular, the Gaussian process bandits method to address this problem. We had used a mathematically sound algorithm with good theoretical foundation to develop active sampling techniques for GPHH for a single objective scheduling problem. However, when we consider a larger number of objectives, our solution is a Pareto set of dispatching rules rather than a single rule. Recalling the GPHH framework which we had proposed in our previous chapter, validation step was an important component of the framework. The validation step required multiple comparisons of the performance of evolved rules. If we consider a similar framework for the multi-objective DJSS problems, the validation step would require comparing Pareto fronts of solutions. This is very expensive, particularly when the validation step is repetitive. More-

over, for effectively comparing the Pareto fronts we require more than one metric [122] demanding a large computational cost. For this reason, even though the proposed active sampling algorithms in the previous chapter were very effective for single objective DJSS problems, their applicability to multi-objective problems is problematic.

One possible solution to address this computational issue is to use surrogate models [94] as an alternative; but they suffer from poor accuracy among many other drawbacks [158]. In our literature survey, we had highlighted the importance of parallel evolutionary algorithms. We had discussed many existing parallelization models to speed up the evolutionary algorithms including MOEAs. In particular, island models stood out not only because they are efficient but also because they have the ability to produce more effective solutions. It was also highlighted that island models have this ability to capture the dynamics of exploration and exploitation due to its migration policies. Recalling the active learning concepts from Chapters 2 and 4, it was mentioned that exploration versus exploitation is a key issue which is addressed by the active learning methods toward sampling of DJSS instances. Taking this into consideration and the inherent dynamics of island models mentioned above, it is encouraging to employ island models to not only speed up the multi-objective optimization algorithms but also leverage it to develop methods for active sampling of DJSS instances for the multi-objective DJSS problem.

In the literature survey, we had also discussed the importance of the design choices we make for an island model and how it has a huge impact on its performance. Migration policies, number of islands, island topology, migration frequency etc. are some of the important design parameters for an island model. Even though island models have been applied to many problems and has some theoretical foundation, our understanding about them is still not complete [115]. In fact for every new problem, the appropriate design parameters of the island model must be identified carefully, preferably supported by empirical evidence. The application of

parallel EAs to hyper-heuristics in general and of island models to GPHH in particular, is still nascent. Moreover, with respect to our aim to develop active sampling methods based on island models it is very important that we identify the design parameters mentioned above through careful experimentation. It is important to determine migration policies and appropriate topologies for the island model which are able promote evolution of effective Pareto set of dispatching rules.

One of the major factors responsible for the high computational cost in the experiments involving GPHH is the cost of function evaluation which in essence owing to the JSS simulations preformed by the discrete event simulator. Considering our requirement to develop empirical support for the choice of appropriate design parameters for island model, which will require large number of experiments, it is a good idea to actually start our experiments using static JSS problem, because essentially the mechanisms of GPHH for both the static and dynamic problems are largely similar. In fact the only major difference, is that for dynamic JSS problems, the arrival of jobs needs be additionally simulated where as in the case of static JSS it is known at the outset. Once the appropriate design parameters are identified, we can switch back to DJSS problems for developing active sampling methods. If the static JSS problem is used in lieu of DJSS for studying the island models, it is also sensible to consider a less complex job shop by not taking into account the uncertainty and variability.

Having demonstrated the ability of GPHH to develop effective dispatching rules for DJSS problems under uncertainty by considering different machine specific scenarios in Chapter 3 and successfully combining active learning with the GPHH framework to evolve rules for many shop scenarios in Chapter 4, now we consider multi-objective DJSS problems with the aim of developing active sampling methods using the island model parallel EA framework. To this end, we firstly need to identify the design parameters for the island model which is suitable for GPHH for DJSS problems and secondly we need to develop methods which can

leverage the exploration and exploitation dynamics of island models toward active sampling. We present the more specific chapter goals below.

5.1.1 Chapter Goals

The goal of this chapter is to develop active sampling heuristics for multi-objective DJSS problems using GPHH approach leveraging the dynamics of island model parallelization framework.

We aim to achieve this through the following sub-goals.

- Investigate different island models defined by the parameters such as migration topology and identify the ones which are capable of evolving an effective Pareto set of dispatching rules using MO-GPHH approach.

  In order to ease the computational burden of these experiments, *static* JSS problems are considered by recognizing the similarity of GPHH mechanisms for both problems.

- Develop an active sampling heuristic for MO-GPHH, namely *successive reject heuristic* (SRH) based on the island model which iteratively rejects training instances in favour of those which have the potential to improve the Pareto front.

  This also requires determining the migration policies of the island model which can promote the efficacy of the proposed sampling heuristic. This will be accomplished by utilizing the empirical results from the previous sub-goal.

5.1.2 Chapter Organization

The remaining chapter is organized as follows. In the Section 5.2 we present our investigations on island model for static JSS problems using GPHH. The Section 5.3 describes the active sample heuristic based on the island model. The final section provides a summary to the chapter.

5.2 Investigating Island Model for JSS problems using GPHH

We briefly recall from Chapter 2, the description of the island model approach to parallelization. Evolutionary algorithms run over many iterations and generally have high convergence time. Parallel evolutionary algorithms are known to converge faster and also provide better performance [79], particularly when parallel models which consider exchange of individuals are employed. Different variants of island model have been proposed [229]. In general, an island model consists of different subpopulations each running a parallel evolutionary algorithm. The subpopulations have a communication topology for migration of individuals among each other. The migration policies could be synchronous or asynchronous. The policy involves selection of the individuals to be migrated to the different subpopulations.

Parallelization models for evolutionary algorithms have been studied for quite some time [79]. Investigations with island model [211] for non-panmictic population [215] have been conducted. One of the directions of the research has been in using different topological structures [229] and their analysis. The other major investigation is related to the population and its migration policies [13, 212].

Island model have been proposed for applications in multi-objective optimization problems [139, 148, 244]. Different topological structures have been used by Xiao et al. [244] toward using island model for multi-objective optimization. In general, determining the island model by defining its migration policies, topology, etc is not straightforward. For example, a higher migration frequency will have a detrimental effect on the ability of the island to explore the promising regions of search space as the currently most successful island will dominate the others while a lower frequency will result in local optima. This implies that the characteristics of search space which are difficult to study and vary with the considered

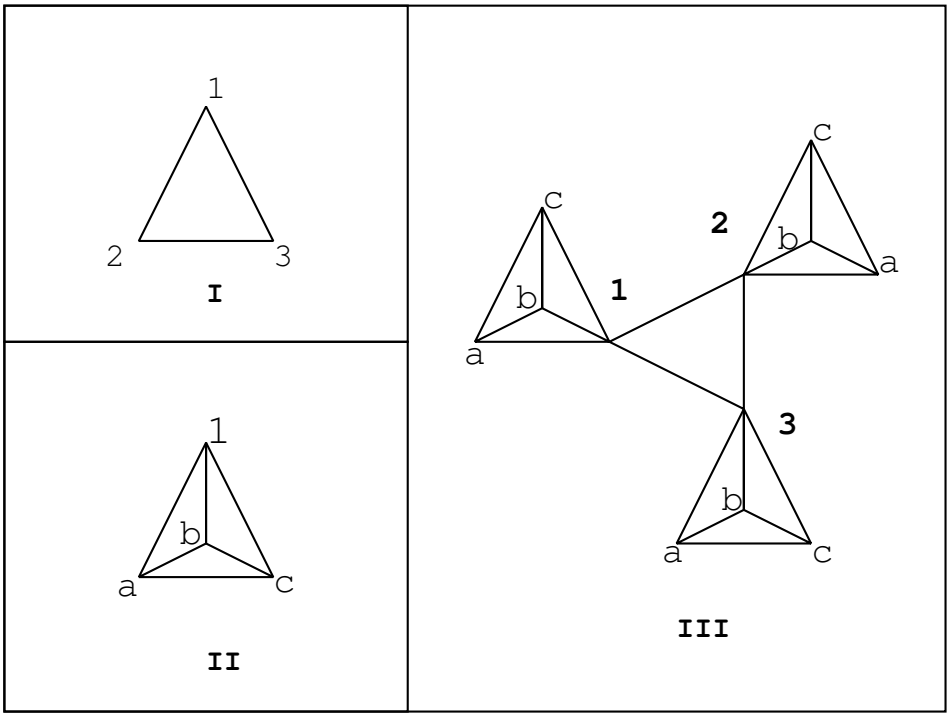


Figure 5.1: Island topologies

problem have an effect on the performance of the island model. Similarly, a topology which makes it difficult for the migrants to reach islands which are further away in the network might result in some of the islands to be stuck in a local optima.

Considering these difficulties, it is sensible to adapt some of the existing models from works which have considered multi-objective optimization with island models. Xiao, et al. [244] is one such work which has inspired us to use some of their island model topologies and empirically analyze the performance on a job shop scheduling problem with multi-objectives.

For this sub-goal, we have considered up to *three* objectives for the scheduling problems. This choice direct influences the design of topologies which we consider. Based on these considerations, we investigate

three different island topologies in our work. We refer them as *TopologyI*,*TopologyII* and *TopologyIII* respectively. They are shown in Figure 5.1.

The communication between the nodes is bi-directional and the individuals can migrate in and out of each island. Also the communication is synchronous in our experiments. In a synchronous communication, the islands 'wait' for all the migrations to complete across all islands for that epoch. An epoch is the number of generations considered for evolution before the migration begins. Asynchronous communication does not involve waiting and is thus usually faster because the islands do not wait for each other to complete the evolution or the exchange of individuals. But it results in staggered completion of evolution across different islands with some of them finishing early while the others still sending their best individuals across.

Different islands in a particular topology optimize the different subsets of the objectives. We refer to the islands which run EA for a subset of objectives which is less than maximum number of objectives with an alphabet $\{a, b, c\}$. The islands represented by $\{a, b, c\}$ in Figure 5.1 find solutions using EA for a subset of objectives. As an example, consider an optimization problem with 3 objectives namely $\{o_1, o_2, o_3\}$. In our experiments, these objectives represent $makespan$, $total tardiness$ and $energy$ respectively. Let us consider the TopologyII, shown in Figure 5.1. The islands a, b, c will optimize the subsets $\{o_1, o_2\}$,$\{o_1, o_3\}$ and $\{o_2, o_3\}$ respectively and the island 1 will optimize all the objectives, $\{o_1, o_2, o_3\}$. TopologyIII is basically using the three units of TopologyII, communicating among each other where island 1,2 and 3 transfer individuals on the one hand and islands a,b,c each communicate within themselves on the other. The islands 1, 2 and 3 optimize all the three objectives.

The motive behind such an assignment of the objectives to the different islands is to study the effect of migration on the final solution (Pareto set of dispatching rules) when different islands focus on different subsets of objectives. Essentially, the characteristics of the search space in each of these

islands is different but since the objectives overlap, the migration of the individuals from these islands could actually promote the improvement of the Pareto front.

Algorithm 13: JSS using Island Model

Input: $\mathcal{T}_o$, $\mathcal{D}ataset(train)$
Output: $\{\Omega_1, \Omega_2, \ldots, \Omega_p\}$

```
for s ← 1 : F do
    for k ← 1 : |T_o| do
        Run s^th iteration of A for island I_k.
        for < I_j^i, h_j^i, u_j^m > ∈ M_k do
            Transfer top-u_j^k fit individuals to I_j^i from I_k.
        end
    end
    Wait for all the communication to complete.
end
Evaluate the pareto front.
Collect the corresponding genetic programs : {Ω_1, Ω_2, ..., Ω_p}.
```

We explain our island model based GPHH algorithm using Algorithm 13. Let $\mathcal{A}$ be the multi-objective optimization algorithm and F is the total number of iterations. In our case we use $\mathcal{A}$ = NSGA-II with island models. NSGA-II is one of the most popular MOEA algorithms preferred when objectives are *three* or less. Note that the choice of MOEA does not effect the outcome of our experiments because the proposed methods will work for any MOEA, as we will show in our experiments. The proposed methods are not aimed at improving MOEAs, instead the goal is to improve the solutions to multi-objective JSS problem.

We define an island *topology* $\mathcal{T}_o$ as set of tuples $< \mathcal{I}_i, \mathcal{M}_i >$ where $\mathcal{I}_i$ is an island and $\mathcal{M}_i$ is its migration policy. $\mathcal{M}_i$ of an island $\mathcal{I}_i$ defines how many top-m individuals should be transferred to which other islands at what interval of generations. Thus $\mathcal{M}_i$ is the set of triplets $< \mathcal{I}_j^i, h_j^i, u_j^m >$

where h_j^i is the interval after which u_j^m individuals are transferred. The migration policies are static i.e. u_j^m and h_j^i do not vary.

In line 3 of Algorithm 13, for each island in the topology, an iteration of the algorithm $\mathcal{A}$ is completed. The top individuals are sent for migration to all other islands (line 5). In line 8, the islands wait to get synchronized. Finally after all iterations are completed, the Algorithm 13 gives the output as the genetic programs $\{\Omega_1, \Omega_2, \ldots, \Omega_p\}$

5.2.1 Experiment Design

The dispatching rules in the experiments are represented by genetic programs constructed from a list of function and terminal sets, as summarized in Table 5.1. Function If-then-else includes three arguments and if the value from the first argument is greater than or equal to zero, the second argument is returned else the third argument is returned. The protected division returns 1 if the second argument is 0. With a tree depth of 6, the crossover and mutation are 0.85 and 0.1 respectively [100]. We conduct 51 iterations for all runs.

Migration policies

In this experiment, static migration policies are considered, shown in Fig. 5.1. Every island will exchange 40 individuals with each adjacent island after every 5 generations. We arrived at these numbers by observing the size of the first non-dominating front in a single run of the NSGA-II algorithm. We also observed that it usually takes 5 generations (especially during the initial stages) to show considerable improvement. The population sizes for different algorithms have been presented in Table 5.2. The values enclosed in brackets refer to population sizes of every island (or subpopulation) when island models and Algorithm 13 are used.

We use the dataset generated by Taillard et al. [223] for our experiments. This dataset consists of 8 subsets that together cover JSS prob-

Table 5.1: Functional and Terminal Sets for genetic programs.

Function Set	Meaning
$+$	Addition
$-$	Subtraction
$*$	Multiplication
$/$	Division
Max	Maximum
Min	Minimum
$If - then - else$	Conditional
Terminal Set	**Meaning**
DueDate	Due date of job (DD)
MachineIdlePower	Power consumed by idle machine(MWP)
MachineWorkPower	Power consumed by working machine(MIP)
ProcessingTime	Processing time of each operation(PR)
RemainingOperations	Remaining operations for each job(RO)
RemainingTime	Remaining processing time of each job(RT)
ERC	Ephemeral Random constant

lems with varied number of jobs and number of machines. The maximum number of jobs considered in any subset is 100 jobs, which will have to be scheduled on 20 separate machines. The JSS problem instances within each subset will be further divided into $60 : 40$ train and test set. This division is completely random and all instances will have an equal probability of being used either for training or testing.

Since a maximum of 20 machines will be included in any problem instance, for all the 20 machines, their idle power rates and working power rates are further determined randomly under a general restriction that the working power rate of any machine must be greater than its idle power rate. Generally, in industries it is not possible to power down the machine when it is idle and power it up when it is required as set-up times are crucial. Moreover, powering up frequently might result in an overall higher

Table 5.2: Population size per island in braces

.

	NSGA-II	SPEA-2	Top-I	Top-II	Top -III
Bi objective	4096	4096	3072 {1024}	–	–
3-objective	4096	4096	12288 {4096}	16384 {4096}	12288 {1024}

consumption of power. Therefore, two rates for energy consumption are considered. The obtained power rates, organized in order according to the 20 machines, are given in Table 5.3. No specific models have been utilized in our experiments to determine these power rates. We believe that this enables us to evaluate the power consumption of job shops without being restricted to specific type of machines and application domains. However, to further evaluate the usefulness of evolved dispatching rules in practical applications, realistic power consumption settings will need to be adopted. We are interested in addressing this issue in our future work. Meanwhile, in our experiments, we assume that the machines are always on. Even though temporary turn-off of machines is proposed to save energy [144], in general many machines used in real job shops cannot be powered down. For example, the printers in a print industry are not shut down when idle as powering them up requires considerable energy and also complex set-ups in some cases.

Table 5.3: Idle power and working power of machines.

Idle power	0.93	0.34	0.77	0.40	0.09	0.25	0.58	0.70	0.23	0.95
	0.66	0.51	0.48	0.22	0.48	0.88	0.13	0.78	0.19	0.28
Working power	0.94	0.74	0.95	0.87	0.61	0.56	0.77	0.97	0.55	0.99
	0.88	1.0	0.72	0.47	0.8	0.97	0.39	0.8	0.85	0.44

In order to determine the due dates, we use a job specific assignment procedure [49]. This follows the procedure of endogenously finding due date by using total work content (TWK) [49]. The due date is assigned with a tightness of 1.25 with all jobs released at the outset. Basically, the

due date is the product of due date tightness and the total processing time of the job.

JSS objectives

We considered makespan, total tardiness and total energy as the three objectives for this experiment. These are described below.

It is generally desirable for a schedule $\mathcal{S}$ to minimize its *makespan*. For any job $\mathcal{J}_i$, let's use $\mathcal{C}_i$ to refer to its completion time according to schedule $\mathcal{S}$. The makespan of schedule $\mathcal{S}$ can hence be determined as the maximum completion time over all jobs, i.e.

$$\mathcal{C}_{max} = \max_{\mathcal{J}_i} \mathcal{C}_i. \tag{5.1}$$

Besides the makespan, the *tardiness* of any job $\mathcal{J}_i$ in a schedule $\mathcal{S}$, i.e. $\mathcal{T}_i$, is defined as $\max\{0, \mathcal{C}_i - \mathcal{D}_i\}$, where $\mathcal{D}_i$ is the *due-date* of job $\mathcal{J}_i$. Thus, the *total tardiness* becomes

$$total\ tardiness = \sum_{\mathcal{J}_i} \mathcal{T}_i \tag{5.2}$$

Energy-aware scheduling is considered strictly harder than constructing schedules that minimize merely the makespan [5]. We will adopt an energy consumption model that is fundamentally identical to the one presented in [144]. Specifically, it is assumed in the model that total energy consumption (equivalent to energy cost if we assume constant power tariff) of any working machine is completely independent from the schedule to be used in a job shop. In other words, for a set of jobs $\mathcal{J}$, two different schedules $\mathcal{S}_1$ and $\mathcal{S}_2$ will result in the same consumption of total working power. Moreover, the machines have constant working and idle power consumption rates. Thus the total energy could be considered as the sum of the idle energy and the working energy.

$$\mathcal{E}^{total}_{price} = \mathcal{E}^{idle}_{price} + \mathcal{E}^{work}_{price} \tag{5.3}$$

The total idle energy cost across all machines is defined in (5.4), where S_k^r and C_k^r stand for the start and completion time of an operation m_k^r performed on machine M_k respectively. P_k^{idle} indicates the machine's idle power rate.

$$\mathcal{E}_{price}^{idle} = \sum_{M_k} \left\{ P_k^{idle} \times \left(\max_{m_k^r}(C_k^r) - \min_{m_k^r}(S_k^r) - \sum_{m_k^r}(C_k^r - S_k^r) \right) \right\} \tag{5.4}$$

The total working energy cost across all machines is defined in (5.5), where P_k^{work} indicates machine's working power rate.

$$\mathcal{E}_{price}^{work} = \sum_{M_k} \left\{ P_k^{work} \times \left(\sum_{m_k^r}(C_k^r - S_k^r) \right) \right\} \tag{5.5}$$

5.2.2 Results and Discussions

In order to compare the Pareto fronts obtained for each run we use hypervolume indicator [250], generational distance [250] and generalized spread [59] as the three metrics. These metrics need a true Pareto front for evaluation which is not known to us. Because of that, following a simple strategy demonstrated in [59], we combine the individual Pareto fronts obtained by NSGA-II, SPEA-2 and our Algorithm 13 together and jointly determine an approximated Pareto front. Separate approximated fronts are created with respect to the train and the test sets. A higher value of hypervolume indicator means better performance while for generational distance and generalized spread a lower value is better. We use the Wilcoxon signed-rank test [242] to verify the significance of our results.

In order to understand whether Algorithm 13 can outperform NSGA-II and SPEA-2 even with commonly used optimization objectives, including both the makespan and total tardiness, a series of experiments have been conducted and the results obtained have been presented in Subsection 5.2.2. Inspired by these encouraging results, further experiments that

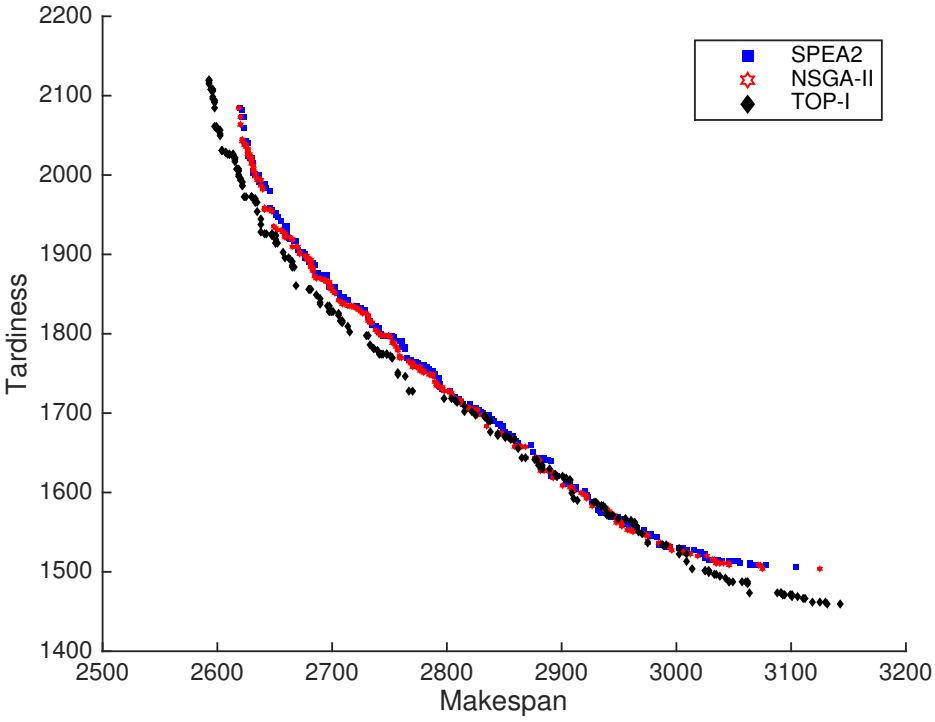
include energy as the third optimization objective have been conducted and reported in Subsection 5.2.2.

Experiments on bi-objective JSS problems

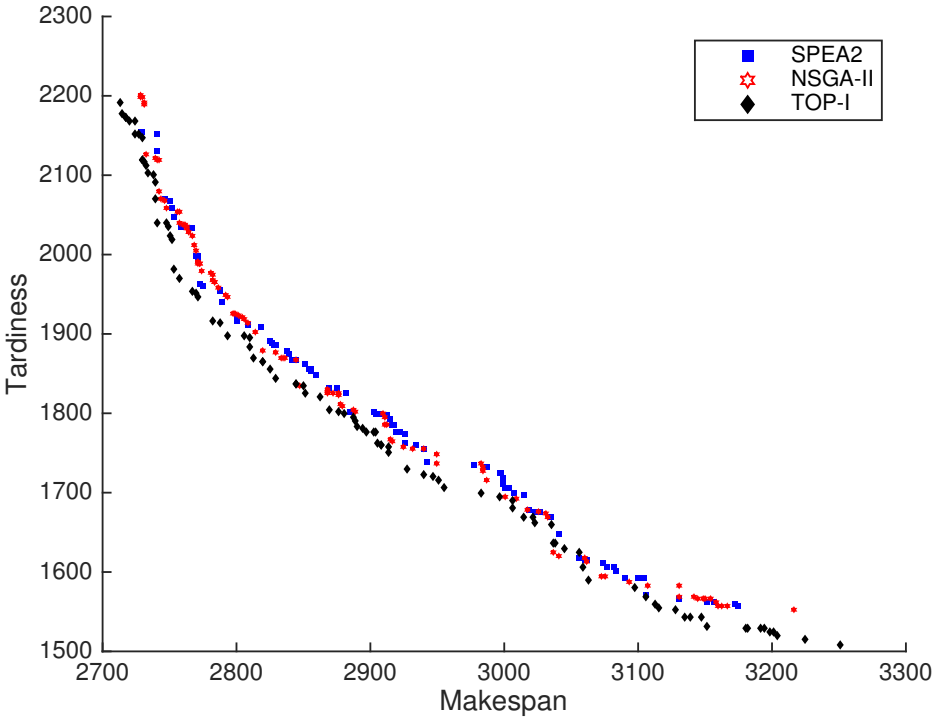
In this experiment, we consider only the optimization of makespan and total tardiness. Since Topologies II and III in Fig. 5.1 involve the use of multiple types of islands, all of which are not necessary for bi-objective optimization, we conduct the experiment using only Topology I. In Topology I, each island will consider both the makespan and total tardiness.

We compare our work against the standard implementation of SPEA-2 and NSGA-II. We combine Pareto fronts from all the runs and generate a single Pareto front from the combined solutions for each algorithm. The combined Pareto fronts are shown in Fig. 5.2. The Pareto front of Topology I dominates the fronts from single population runs of SPEA-2 and NSGA-II. One the one hand the computational resource required by the Topology-I is less than the single population runs and yet the Topology-I could produce a Pareto front which is better. We show the box-plot comparisons from the runs in Fig. 5.3. For the Wilcoxon test to be significant we need the p-value to be lower than 0.05. The hypervolume indicator shows Topology I to outperform NSGA-II and SPEA-2 for the train set. For the hypervolume indicator and generalized spread we obtained the p-values of $7e-15$ and 0.09 (not significant) respectively against NSGA-II. For the generational distance, Algorithm 13 shows no improvement. For the test set we observe similar performance, with the p-values of $7e-15$ and 0.02 for hypervolume indicator and generalized spread respectively against NSGA-II.

More importantly the total population size used for Topology I is less than for the other two methods. As shown in Table 5.2, the population used for Topology I is 1024 per island which sums to 3072 individuals for the topology. In our experiments we observed that NSGA-II took close to 3.8 hours for completion against approximately 0.76 hours for Topology I.

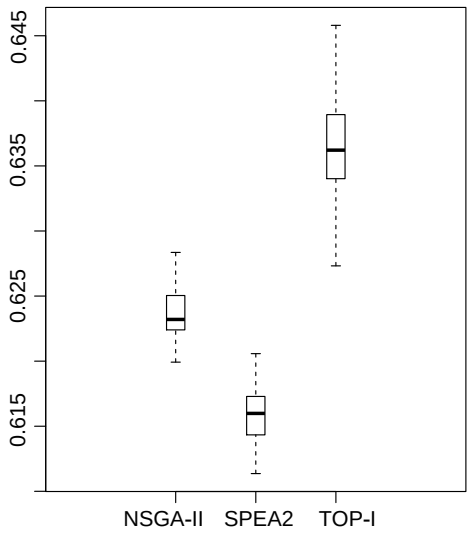


(a) Train

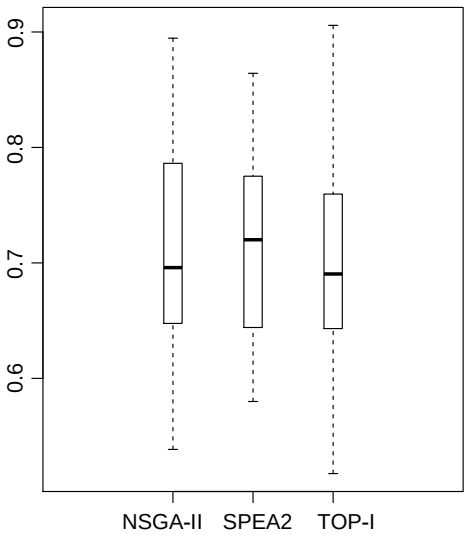


(b) Test

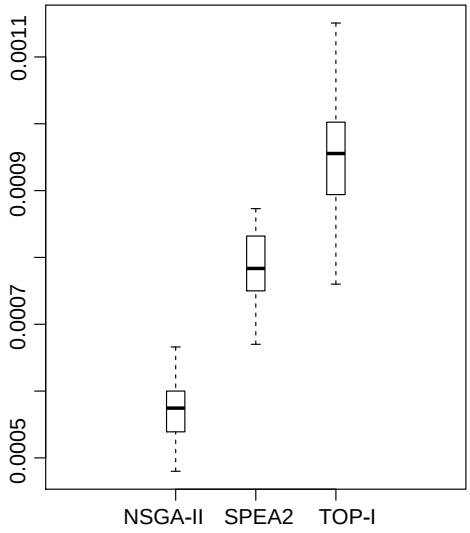
Figure 5.2: Bi objective : combined Pareto fronts.



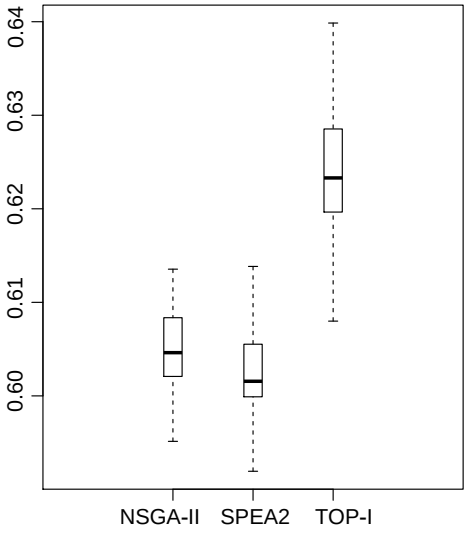
(a) Hypervolume indicator (train) ↑



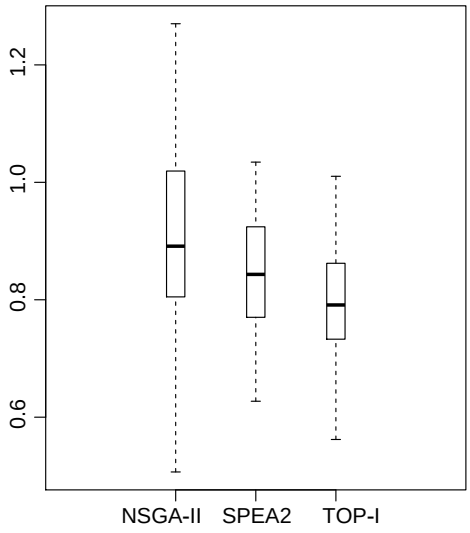
(b) Generalized spread (train) ↓



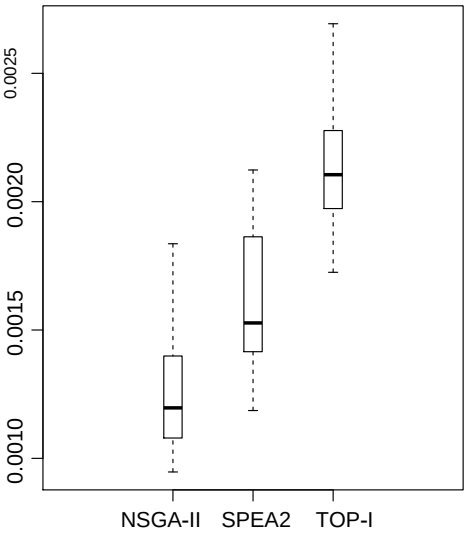
(c) Generational distance(train) ↓



(d) Hypervolume indicator(test) ↑



(e) Generalized spread(test) ↓



(f) Generational distance(test) ↓

Figure 5.3: Bi-objective optimization (up arrow indicates higher the better)

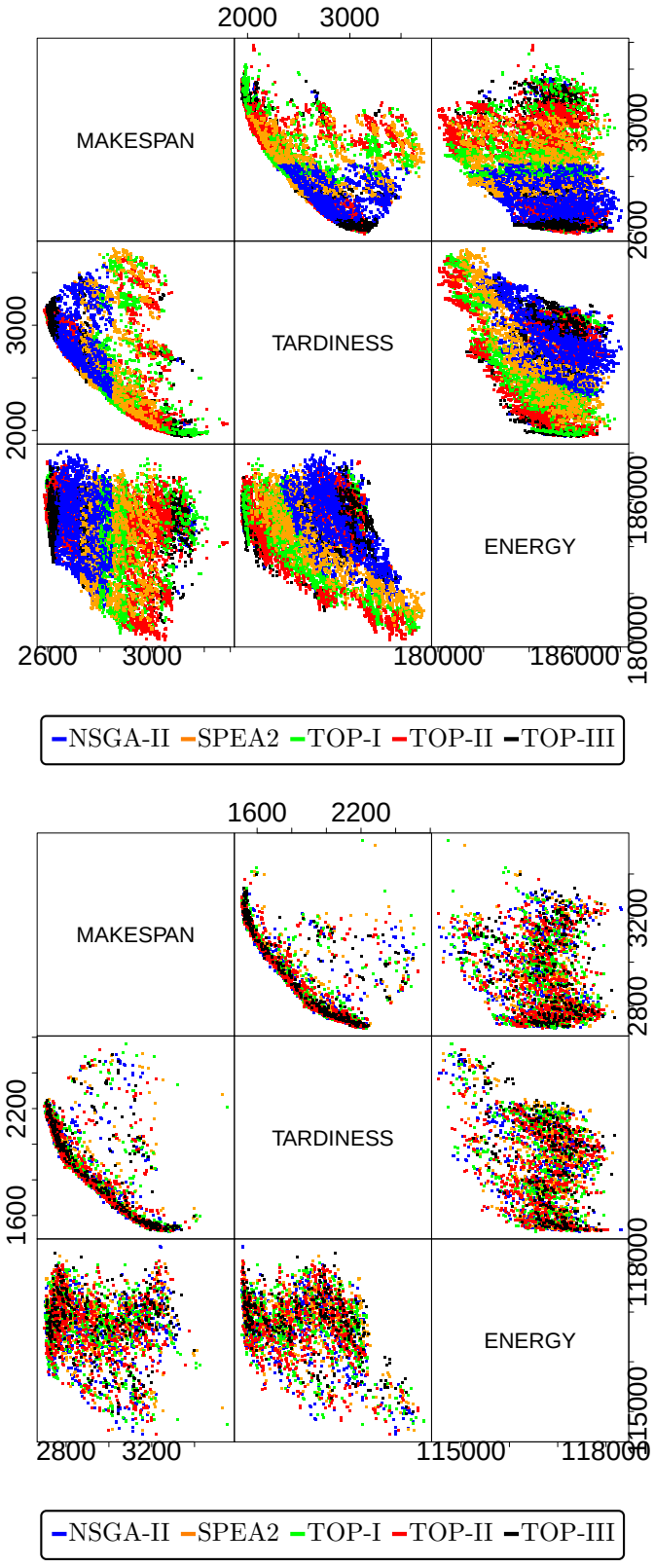
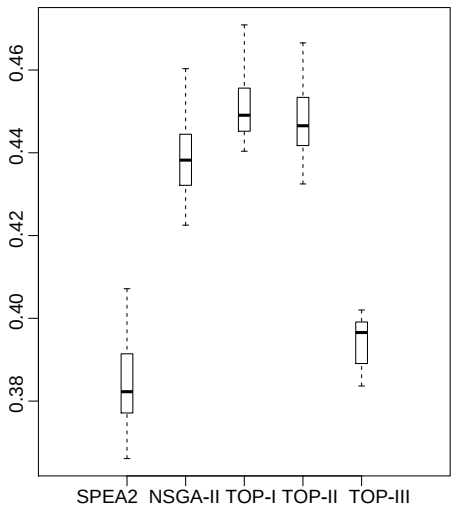
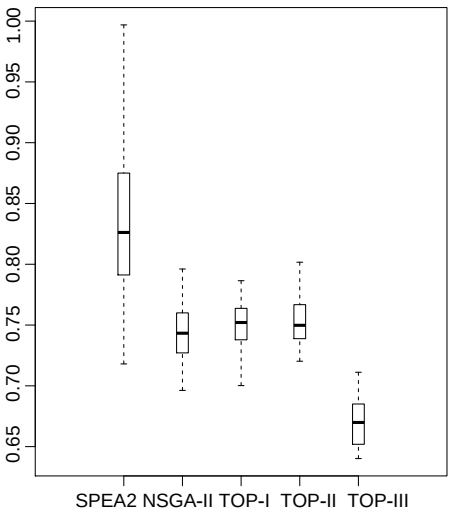


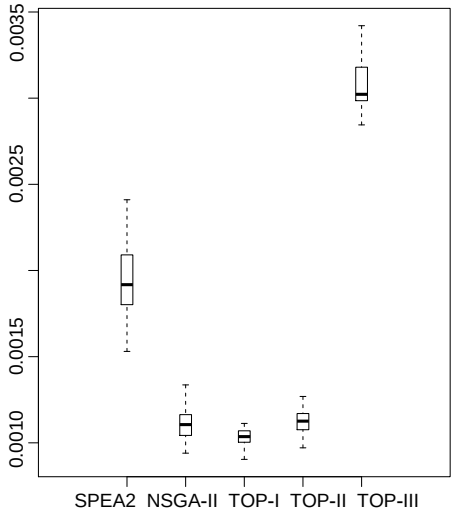
Figure 5.4: Multi-objective pareto fronts. (top: train, bottom: test)



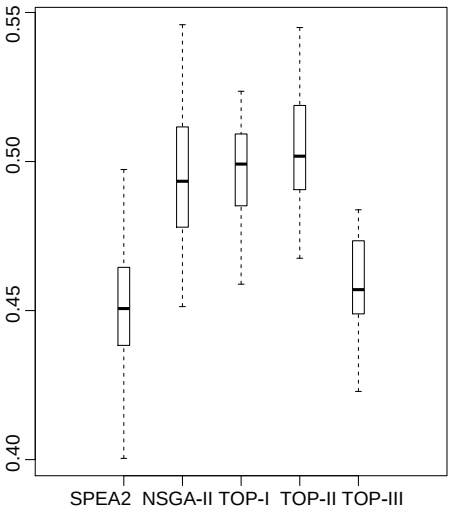
(a) Hypervolume indicator(train) ↑



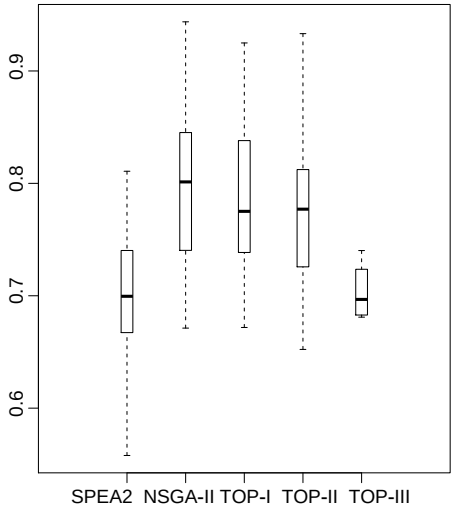
(b) Generalized spread(train) ↓



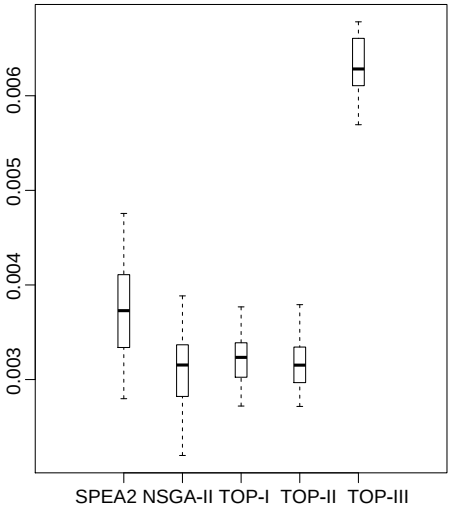
(c) Generational distance(train) ↓



(d) Hypervolume indicator(test) ↑



(e) Generalized spread(test) ↓



(f) Generational distance(test) ↓

Figure 5.5: Multi-objective optimization (up arrow indicates higher the better)

The increased performance and saving in time comes at the price of some communication complexity as the exchange of individuals based on the migration policy requires communication.

Experiments on 3-objective JSS problems

In this set of experiments, we evolve solutions to the multi-objective optimization problem (three objectives) and use all the proposed topologies in Fig. 5.1.

Algorithm 13 again shows improvement over NSGA-II. The pairs plot of the Pareto front is shown in Fig. 5.4. The training set shows that there is significant difference in the quality of solutions generated from the different methods. This is more clear when we show the results using the box plots (Fig. 5.5). But we do not observe a significant difference in the test, as the solutions represented by different methods are not visually distinguishable in the plot. Therefore, we need to rely on statistical tests to determine the effectiveness of our algorithm.

We again use the metrics of hypervolume indicator, generalized spread and generational distance to compare the different methods and the Wilcoxon signed-rank test to determine the significance (0.05) of our results. The box plot of the results is shown in Fig. 5.5. For the train set, Topology I outperformed NSGA-II with a p-value of $2.4e-06$ and 0.0002 for the hypervolume indicator and generational distance respectively. For the test set Topology I and Topology II outperformed NSGA-II, with respect to the hypervolume indicator, showing p-values of $3e-8$ and $1e-14$ respectively. The Topology II also outperformed Topology I with a p-value of 0.02 for the same metric. There is no significant difference for performance based on generational distance in the test set.

Though Topology III outperformed SPEA-2 but not NSGA-II, it must be noted that the computation time needed in Topology III is much lower than that of NSGA-II. On average, the processing times of 5.2 hrs, 5.5 hrs, 6.4 hrs and 2.7 hours were needed for NSGA-II and Topologies I, II

and III respectively per run. For the population sizes as indicated in Table 5.2, although Topology I and II required longer processing time than NSGA-II due to the communication and synchronization overhead, they can achieve significantly better performance.

To summarize, of the three island topologies used with Algorithm 13, Topology I generally performed better than both NSGA-II and SPEA-2, as confirmed particularly by the hypervolume indicator. Though Topology II also performed well in Subsection 5.2.2, it did not outperform Topology I significantly. Because only simple and static migration policies have been utilized in our island models, useful individuals cannot be effectively exchanged among multiple islands in Topology III. As a result Topology III failed to perform as we hoped. However, considering the fact that Topology III could potentially reduce the total time required for evolving useful dispatching rules, its practical usefulness should be further investigated in the future.

5.2.3 Section Summary

In this section, we investigated the island models for static multi-objective JSS problems using GPHH. We tried to identify the parameters which could be used to design island models which are capable of promoting the evolution of effective Pareto set of dispatching rules. In general, we found the island model to perform significantly better than other MOEAs. In particular the Topology-I performed significantly better than single population MOEAs like NSGA-II and SPEA-2 for the bi-objective optimization. Topology-I is not only efficient in using the computational resources but is also very good in evolving effective Pareto set of dispatching rules. This finding is very important for our next sub-goal related to the development of active sampling heuristics.

We also conducted out experiments for 3-objective JSS with three objectives which included minimization of energy cost. Generally, JSS prob-

lems do not consider such objectives and this work is novel with respect applying GPHH approach toward energy-aware scheduling.

The other topologies, even though were computationally efficient, showed mixed results when compared with single populations MOEAs. Considering the higher complexity of their topologies, a more complex migration policy, possibly dynamic in nature could be considered in our future work.

5.3 Active Sampling Heuristic for DJSS problems using Island Model

In this section, we present our active sampling heuristic for multi-objective DJSS problems. We recall the feature extraction and clustering methodology from Chapter 4 and then present our proposed methods. Following that, we present the experiment design, results and analysis.

We discussed in the Section 5.1 about the importance of active sampling of DJSS instances for multi-objective DJSS problems. Essentially, the DJSS instances represent shop scenarios some of which are more important (as explained through our examples in Section 5.1) in promoting the evolution of better dispatching rules (Pareto set of dispatching rules in multi-objective case). Since it is our goal to develop active learning techniques which can identify such instances, we need a methodology which can group similar instances which correspond to shop scenarios with similar characteristics.

In Chapter 4, we had described a feature extraction and clustering methodology in order to facilitate the application of our active learning methods. We use a similar methodology in this chapter. Table 5.4 is reproduced from chapter 4 and described the features which are extracted from a DJSS instance.

Feature Extraction and Clustering

We extract the job features for each job arriving at the shop using Table 5.4. The features are related to the number of operations in a job, the processing time, the uncertainty in processing time and the due date. Once these features are obtained, the quartiles are obtained for each feature and a 12 dimensional feature vector is constructed. This is used as the feature vector of a DJSS instance.

Table 5.4: Job Features

Feature	Description
#operations	number of operations per job.
p	estimated processing time of the job.
Δ^p	$\frac{p'}{p}$, p' is the actual processing time with uncertainty.
due date factor (ddf)	$\frac{(\delta_{duedate}-\delta_{reldate})}{p'}$; where $\delta_{duedate}$ is the due date and $\delta_{reldate}$ is the release date

5.3.1 Proposed Method

The active sampling heuristic which we are about to present is called successive reject heuristic (SRH). The intuition behind this heuristic is to successively remove those DJSS instances (cluster of DJSS instances) which are comparatively less useful toward promoting the evolution of an effective Pareto front. This heuristic has an iterative nature and therefore, we need the clustering to be hierarchical so that the SRH could exploit the hierarchy to actively sample (or reject) cluster of DJSS instances.

More formally, $\mathcal{T}$ is the training set containing n DJSS problem instances. We extract features for all these problems and cluster them into $\mathcal{C}_1$ and $\mathcal{C}_2$ using K-means clustering. We apply K-means clustering again on each of these clusters to yield $\{\mathcal{C}_{11}, \mathcal{C}_{12}\}$ and $\{\mathcal{C}_{21}, \mathcal{C}_{22}\}$ respectively. This

process can be repeated to obtain more sub-clusters $\{\{\mathcal{C}_{111}, \mathcal{C}_{112}\}, \{\mathcal{C}_{121}, \mathcal{C}_{122}\}\}$ and $\{\{\mathcal{C}_{211}, \mathcal{C}_{212}\}, \{\mathcal{C}_{221}, \mathcal{C}_{222}\}\}$ and so on.

Island Model

In Section 5.2, we conducted many experiments with different topologies for multi-objective DJSS problems. We observed significant improvement when using Topology-I for bi-objective optimization. Therefore, in this section, for developing our active sampling method we employ the Topology-I of island models.

As mentioned earlier, the SRH iteratively rejects the cluster of DJSS instances. In order to facilitate this process, the SRH exploits the island model. We use two classes of islands in our evolutionary system. The first class of islands, represented as G in Figures 5.6(a) & 5.6(b), sample training instances from the set $\mathcal{T}$ throughout the evolutionary process. The second class of islands represented as A and B in Figure 5.6(b) sample problem instances from the different clusters the choice of which varies with generations. The appropriate choice of the cluster is controlled by the *successive reject heuristic*. The heterogeneous island model is exploited by the SRH in its iterative process of rejecting the clusters.

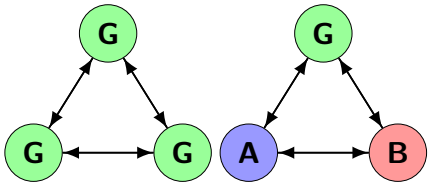


Figure 5.6: (a)Standard island model, (b) Island model for successive reject heuristic

We first describe the evolutionary process of island G in Algorithm 14. At every generation, a new training instance is sampled from $\mathcal{T}$. Unless otherwise mentioned, we use sample to denote a *simple random sam-*

Algorithm 14: Island G

Input: $\mathcal{T}$

Output: $\{\omega_1, \omega_2, \ldots, \omega_p\}$

1 **for** $g \leftarrow 1 : N_G$ **do**
2 Sample an instance $\mathcal{I} \in \mathcal{T}$.
3 Run g^{th} iteration of NSGA-II using $\mathcal{I}$.
4 Receive/Send individuals using migration policies.
5 **end**
6 Collect the genetic programs corresponding to the Pareto front : $\{\omega_1, \omega_2, \ldots, \omega_p\}$.

Algorithm 15: Island Z ($Z \in \{A, B\}$)

Input: $\mathcal{C}_Z$,$\mathcal{N}_{SRH}$

Output: $\{\omega_1^z, \omega_2^z, \ldots, \omega_p^z\}$

1 **for** $g \leftarrow 1 : N_G$ **do**
2 Sample an instance $\mathcal{I} \in \mathcal{C}_Z$.
3 Run g^{th} iteration of NSGA-II using $\mathcal{I}$.
4 Receive/Send individuals using migration policies.
5 **if** $g \in \mathcal{N}_{SRH}$ **then**
6 $\mathcal{C}_Z \leftarrow SRH(P_k^A, P_k^B, \mathcal{C}_A, \mathcal{C}_B)$
7 **end**
8 Collect the genetic programs corresponding to the Pareto front : $\{\omega_1^z, \omega_2^z, \ldots, \omega_p^z\}$.

ple [218]. Due to our familiarity with NSGA-II [59] and the fact that we consider only two objectives in this work, we chose NSGA-II as our underlying evolutionary algorithm. In line 3, an iteration of NSGA-II is performed. After each generation, the migration policy determines (line 4) if there will be an exchange of individuals among the islands. The output is a set of dispatching rules which jointly form a Pareto front. Note that the final output of the parallel evolutionary system is the combination of

outputs from all individual islands.

Table 5.5: Notation

Notation	Description
$\mathcal{T}$	set of all DJSS problem instances for training.
N_G	total number of generations for evolutionary process.
$\mathcal{C}_Z$	cluster corresponding to island $Z \in \{A, B\}$.
P_k^Z	top k individuals from island $Z \in \{A, B\}$.
$M_{\overrightarrow{X,Y}}$	migration policy from island X to Y.
P_{2k}	combined list of top k individuals from islands A and B.
TOP_k	list of top k individuals across island A and B.
$TOP^{\mathcal{Z}_k}$	#individuals which are present both in $P_k^{\mathcal{Z}}$ and TOP_k, $\mathcal{Z} \in \{A, B\}$
$\mathcal{N}_{SRH}$	set of generations at which SRH is invoked.

In Algorithm 15, we describe the evolutionary process of the islands A and B (Figure 5.6(b)). Both islands are similar, except that they sample their training instances from different clusters. Their migration policies, which we will define in detail later, are the same. The cluster $\mathcal{C}_Z$ is changed at discrete stages during the evolutionary process. The set $\mathcal{N}_{SRH}$ contains the generations at which the successive reject hypothesis is invoked to change $\mathcal{C}_Z$ (line 6). The rest of the procedure is the same as in Algorithm 14.

Successive Reject Heuristic

The successive reject heuristic (SRH) is described in Algorithm 16. When the SRH is invoked by islands A and B at generation $g \in \mathcal{N}_{SRH}$, the top-k individuals from each island, P_k^A and P_k^B respectively, are sent to the island G on which the SRH algorithm is run. The two sets of individuals are then combined to get P_{2k} (line 3). A new set of DJSS problem instances,$\mathcal{I}$ is sampled from $\mathcal{T}$ and utilized to evaluate the new fitness values of each individual in P_{2k} (lines 4–11).

Algorithm 16: Successive Reject Heuristic

Input: $P_k^A, P_k^B, \mathcal{C}_A, \mathcal{C}_B$

Output: $\{\mathcal{C}_A^{new}\}$,$\{\mathcal{C}_B^{new}\}$ to respective islands.

$\mathcal{T} \leftarrow$ set of all DJSS training instances.
$\mathbb{I} \leftarrow$sample from $\mathcal{T}$
$P_{2K} \leftarrow \{P_k^A, P_k^B\}$
foreach $p \in P_{2k}$ **do**
    $tot.fit. \leftarrow \overrightarrow{0}$
    foreach $\mathcal{I} \in \mathbb{I}$ **do**
        $obj.values \leftarrow$ Simulation for $(p, \mathcal{I})$.
        $tot.fit. \leftarrow tot.fit. + obj.values$
    end
    $fit(p) \leftarrow tot.fit.$
end
Sort P_{2k} using NSGA-II fitness startegies.
$TOP_k \leftarrow$ Extract top-k individuals from P_{2k}.
$TOP_k^{\mathcal{A}} \leftarrow |P_k^A \cap TOP_k|$
$TOP_k^{\mathcal{B}} \leftarrow |P_k^B \cap TOP_k|$
if $TOP_k^{\mathcal{A}} \geq TOP_k^{\mathcal{B}}$ **then**
    Reject $\mathcal{C}_B$.
    $\{\mathcal{C}_A^{new}\}$,$\{\mathcal{C}_B^{new}\} \leftarrow$ K-means cluster($\mathcal{C}_A$)
else
    Reject $\mathcal{C}_A$.
    $\{\mathcal{C}_A^{new}\}$,$\{\mathcal{C}_B^{new}\} \leftarrow$ K-means cluster($\mathcal{C}_B$)
end

After the fitness assignment, we use the NSGA-II fitness strategies to sort the individuals. NSGA-II first ranks the individuals based on dominance relation and then the individuals with same rank are ordered based on crowding distance [59]. We use the same approach to sort the indi-

viduals in P_{2k} (line 12). After sorting, the best k individuals are extracted from P_{2k} into the list TOP_k (line 13). Then we count the number of individuals corresponding to each island in the list TOP_k (lines 14–15). The cluster corresponding to the island with the lower number of individuals in TOP_k is rejected (line 17 or 20). The $\mathcal{C}_Z$ of the winning island is further clustered into two sub-clusters (lines 18 or 21). The new clusters which are the output of SRH algorithm are then randomly assigned to the islands. Till the next invocation of SRH, the evolution in the islands is continued using DJSS problem instances sampled from the new clusters.

Now we describe the migration policies which are key to the success of the proposed heuristic.

Migration Policies

Migration policies play a major role in the performance of the island models [168]. A migration policy states the number of individuals to be sent to the destination island, frequency of migration and the generation from which the migration starts. For the standard island model shown in Figure 5.6(a) designing a policy is straightforward. Due to homogeneity, a single policy for all the islands will suffice. Since we consider two classes of islands, different migration policies must be designed for islands of different classes.

Formally, a policy $\mathcal{M}_{\overrightarrow{I_1,I_2}}$ from island I_1 to I_2 is defined by a triplet <start generation, frequency, #individuals to send>. We consider the migration policy $\mathcal{M}_{\overrightarrow{I_1,I_2}}$ to be different from $\mathcal{M}_{\overrightarrow{I_2,I_1}}$. The selection of individuals for migration is based on elitism, i.e, a proportion of fittest individual(s) are chosen from the population for migration.

For island G, it is more productive to receive individuals from A and B frequently as this will improve solution diversity. This is because the evolved rules in A and B are exposed to training instances which are different from G. On the other hand, a high frequency of migration between A and B will homogenize the islands, making SRH less effective. More-

over, the frequency of migration in $M_{\overrightarrow{AG}}$ is much higher than $M_{\overrightarrow{GA}}$ (similar for island B) for the same reasons. The same analysis applies to determining the number of individuals to be migrated between the two. Furthermore, the migration policy $M_{\overrightarrow{AB}}$ is restricted to exchanging individuals only and immediately after invocation of SRH.

5.3.2 Experiment Design

The simulation model used for this experiment is consistent with our other experiments in this thesis. We repeat some of the important details for clarity. The job arrival follows a Poisson process with $\lambda = 0.85$ [117]. This assumption has been used in large number of works [34, 158, 162]. For every run of the simulation, the first 500 jobs are considered as warm-up and the objective values are calculated for the next 2000 jobs.

The uncertainty in processing times is simulated using the model considered earlier in Chapters 3 and 4. Basically for an operation $o_{j,i}$ the relationship between the processing time with uncertainty $p'_{j,i}$ and processing time without uncertainty $p_{j,i}$ is:

$$p'_{j,i} = (1 + \theta_{j,i})p_{j,i}, \theta_{j,i} \geq 0.$$

θ follows exponential distribution [117]. In Table 5.6, the parameter β corresponds to the scale parameter of the exponential distribution.

In order to create problem instances with varying characteristics, DJSS problem instances are generated with many combinations of the simulation parameters shown in Table 5.6. The combination of these four pairs of parameters can simulate 16 types of jobs. When building a training DJSS problem instance, 3 job types are considered at a time. On counting the unique combinations of 3 job types we find a total of 816 possible configurations (combinations with repetitions $\binom{n+k-1}{k}$). Since we extract features from problem instances in order to perform clustering we create 20 DJSS problems for each configuration to build the training set $\mathcal{T}$. Our preliminary study showed that a large training set would show no advantage but

Table 5.6: DJSS simulation parameters

Simulation paramter	Values
Processing time range	[0,49],[20,69]
Uncertainty scale parameter (β)	$\{0.2, 0.4\}$
Due date tightness	$\{1.5, 2.5\}$
# operations per job	$\{8, 10\}$

require more computational effort because by increasing the number of DJSS instances per configuration will only increase the cluster size which adds no additional benefit to the algorithms.

For testing, we create a new set (say $\mathcal{Y}$) of DJSS problems using the 816 possible configurations mentioned above. We sample 30 DJSS problem instances from $\mathcal{Y}$ to obtain our first test set. Due to large number of problem configurations it is not possible to test on each of them separately. Therefore, we create $four$ more test sets by clustering $\mathcal{Y}$ and sampling 30 problem instances from each. These test sets are denoted by 3-$\mathcal{Y}$, 3-I, 3-II, 3-III and 3-IV , where 3 stands for number of job types.

We also want to observe the generalization ability of our methods over more complex configurations. Therefore, DJSS instances comprising of 4 job types are created. On counting, the total number of unique configurations in this case are as reaches 3876 (combinations with repetitions). Performing the same procedure described above generates the following test sets: 4-$\mathcal{Y}$, 4-I, 4-II, 4-III and 4-IV.

The GP System

The terminal set for genetic programming is listed in Table 5.7 and the function set in Table 5.8. The protected division returns 1 when the second argument is 0. For all our islands we use a population size of 800 each. We also compare the performance of our method with the standard NSGA-II for which the population size is set at 2500. With a tree depth

of 6, the crossover and mutation are 0.85 and 0.1 respectively [158]. Each evolutionary algorithm is run for 150 generations.

Table 5.7: Terminal Sets for GP.

Terminal Set	Meaning
PT	Processing time of operation
RO	Remaining operations for job
RJ	Ready time of job
RT	Remaining processing time of job
RM	Ready time of machine
DD	Due date
W	Job weight
ERC	Ephemeral Random constant

Table 5.8: Function Set for GP.

Function Set	Meaning
$+$	Addition
$-$	Subtraction
$*$	Multiplication
$/$	Protected Division
Max	Maximum
Min	Minimum

Island model

The SRH algorithm also requires the simulator to assign fitness to individuals. Furthermore, for GPHH to utilize the problem instances from a cluster, considerable number of generations are required. So frequently invoking SRH will not yield the desired outcome but only incur additional

computational cost. Therefore the size of $\mathcal{N}_{SRH}$ is small and generations selected are far apart. Therefore, we use the SRH algorithm at generations 49 and 99, i.e., $\mathcal{N}_{SRH} = \{49, 99\}$. We have used ≈ 50 as the number of generations in many of our works for GPHH, including our experiments on island model investigations in Section 5.2. Therefore, these many generations should be enough for the evolutionary algorithms to evolve effectively.

The migration policies are presented in Table 5.9. While deciding the frequency parameter of the migration policies involving islands A and B, $\mathcal{N}_{SRH}$ has been taken into account. The exchange of individuals starts after a delay as the evolved rules in the early generations are not good. For the TOP_k individuals the value $k = 30$ was chosen. We arrived at this value after observing the size of non-dominated Pareto fronts in the islands.

Table 5.9: Migration Policies

Island-pairs	Policies
$\overrightarrow{GG}$	$< 20, 20, 30 >$
$\overrightarrow{AB}$	$< 50, 50, 60 >$
$\overrightarrow{BA}$	$< 50, 50, 60 >$
$\overrightarrow{AG}$	$< 20, 20, 30 >$
$\overrightarrow{BG}$	$< 20, 20, 30 >$
$\overrightarrow{GB}$	$< 50, 25, 10 >$
$\overrightarrow{GA}$	$< 50, 25, 10 >$

5.3.3 Results and Discussions

In this section, we present the results from our experiments with SRH using island model. We compare the performance of our method with the standard NSGA-II algorithm and the standard island model approach. The hypervolume ratio (HV), inverted generational distance (IGD) and

Table 5.10: Island-Model versus NSGA-II

	3-$\mathcal{Y}$	3-I	3-II	3-III	3-IV	4-$\mathcal{Y}$	4-I	4-II	4-III	4-IV
HV	[18-12-0]	[11-19-0]	[18-12-0]	[19-11-0]	[17-13-0]	[14-16-0]	[15-15-0]	[18-12-0]	[16-13-0]	[12-18-0]
IGD	[24-6-0]	[21-9-0]	[25-5-0]	[29-1-0]	[27-3-0]	[24-6-0]	[21-9-0]	[22-8-0]	[22-8-0]	[19-11-0]
SPREAD	[3-22-5]	[0-18-12]	[4-23-0]	[5-25-0]	[2-28-0]	[3-21-6]	[6-22-2]	[4-23-3]	[5-20-5]	[2-24-2]

Table 5.11: SRH-Island Model versus NSGA-II

	3-$\mathcal{Y}$	3-I	3-II	3-III	3-IV	4-$\mathcal{Y}$	4-I	4-II	4-III	4-IV
HV	[23-7-0]	[17-3-0]	[23-7-0]	[25-5-0]	[24-6-0]	[20-10-0]	[24-6-0]	[22-8-0]	[24-6-0]	[21-9-0]
IGD	[30-0-0]	[30-0-0]	[27-3-0]	[29-1-0]	[30-0-0]	[30-0-0]	[27-3-0]	[30-0-0]	[28-2-0]	[29-1-0]
SPREAD	[1-23-6]	[0-25-5]	[1-27-2]	[3-26-1]	[0-28-2]	[4-21-5]	[1-27-2]	[1-27-2]	[2-22-6]	[0-25-5]

spread (SPREAD) indicators are again considered for comparison. In order to approximate the true Pareto front as required by performance indicators, the individuals from all the methods across all runs are combined. For each method, the solutions are compared over 30 problem instances from a test set. 30 independent runs produce 30 sets of dispatching rules for each method. The Wilcoxon-rank-sum test is used to compare the performance. We consider a significance level of 0.05.

The results are summarized in Tables 5.10-5.12. Each cell in the tables consists of a triplet which represents $[win\text{-}draw\text{-}lose]$. For example, in Table 5.10 the comparison between standard island model and NSGA-II approach is summarized. For the training set 3-$\mathcal{Y}$, if we consider hypervolume indicator, then island model has significantly outperformed NSGA-II in 18 problem instances and there is no significant difference observed for 12 problem instances.

In Table 5.10, we compare NSGA-II with standard island model. As expected, the performance of island model is much better, which is line with the observations made in Section 5.2. For HV and IGD performance indicators, the performance is very good, but for SPREAD indicator there is no clear winner. This significant difference in performance is consistent across all the test sets including 4-job type configurations.

In Table 5.11, we compare the performance of NSGA-II and SRH-based

Table 5.12: SRH-Island Model versus Island model

	3-$\mathcal{Y}$	3-I	3-II	3-III	3-IV	4-$\mathcal{Y}$	4-I	4-II	4-III	4-IV
HV	[10-20-0]	[5-24-1]	[6-22-2]	[9-21-0]	[14-16-0]	[10-20-0]	[10-20-0]	[8-21-1]	[9-21-0]	[11-19-0]
IGD	[18-12-0]	[20-10-0]	[19-11-0]	[19-11-0]	[20-10-0]	[15-15-0]	[14-15-1]	[13-17-0]	[15-15-0]	[18-20-0]
SPREAD	[5-18-7]	[10-20-0]	[3-24-3]	[1-25-4]	[0-23-7]	[5-20-5]	[1-22-7]	[2-20-8]	[4-17-9]	[3-24-3]

island model (SRH). Across all the test sets the proposed method performed well. Particularly for HV indicator, the SRH method has significantly done better than NSGA-II in more than 20 problem instances for almost every test set. Similar performance is observed for IGD as well. Once gain, however, with respect to SPREAD, there is no verifiable difference. This is because the obtained Pareto fronts are sparse for all the algorithms.

Finally we compare, the SRH approach with the standard island model. Once again the SRH approach performs significantly better on an average of 10 problem instances from each test set and no significant difference on the others. This confirms that SRH approach was able to associate useful training instances through the successive rejection of clusters of training instances.

We have included more detailed results in the Appendix B.

5.3.4 Analysis

A frequently observed path taken by the successive reject heuristic is represented below.

$$\mathcal{T} \rightarrow \{\cancel{\mathcal{C}_1}, \mathcal{C}_2\} \rightarrow \{\mathcal{C}_{21}, \cancel{\mathcal{C}_{22}}\} \rightarrow \{\mathcal{C}_{211}, \mathcal{C}_{212}\}$$

In **retrospect**, we analyze the clusters which showed potential to guide the GPHH toward evolving better rules. In order to further validate the ability of SRH, we took the clusters represented by $\mathcal{C}_{21}$ and $\mathcal{C}_{22}$ as training sets. We performed 30 independent runs of NSGA-II algorithm on each. We observed that the cluster rejected by SRH ($\mathcal{C}_{22}$) performed significantly

poor on both HV and IGD indicators. Figure 5.7 shows a box plot for HV indicator on a test problem instance from the set 3-$\mathcal{Y}$.

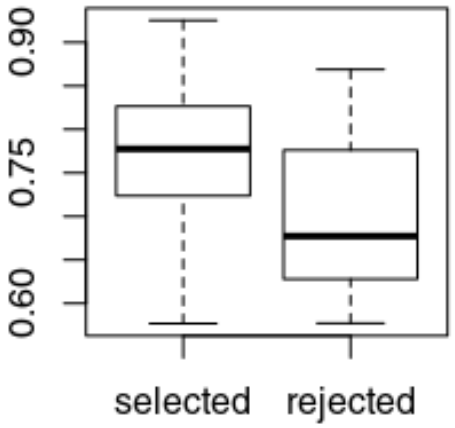


Figure 5.7: Comparing $\mathcal{C}_{21}$(selected) and $\mathcal{C}_{22}$ (rejected) using HV.

Furthermore, we also analyzed the problem configurations associated with cluster $\mathcal{C}_{21}$. One of the reasons for analyzing $\mathcal{C}_{21}$ rather than $\mathcal{C}_2$ is its smaller size and also the fact that, out of 30 independent runs, this path was chosen by SRH for 20 of the runs. We observed that the DJSS instances whose job types were pertaining to equal proportion of high and low level of uncertainty were in high numbers. Also, DJSS instances comprising jobs with low and high number of operations per job were found in large numbers. In other words, SRH is biased towards instances with high variability in their jobs. A high variability in the training instances has more potential to present the GPHH with difficult and conflicting scenarios, as explained in a previous example in Section 5.1.

5.3.5 Section Summary

In this Section, we developed an active sampling heuristic for multi-objective DJSS problems. The proposed successive reject heuristic employed the Topology-I (Section 5.2) to evolve Pareto set of dispatching rules which are significantly better than the NSGA-II and also better than the standard island model based on Topology-I. We also developed migration policies specifically for SRH to assist the method in identifying useful clusters of DJSS instances.

Further analysis revealed that the active sampling heuristic was able to identify those instances which have the potential to highlight the conflict between the objectives.

5.4 Chapter Summary

The goal of this chapter was to investigate the island model approach for GPHH toward evolving dispatching rules for multi-objective JSS problems and then extend the active sampling techniques to multi-objective DJSS problems.

To achieve this goal, different topologies of island model were explored for static JSS problems. The empirical results were promising and the key findings were used to develop an active sampling technique called successive reject heuristic (SRH). The proposed heuristic leveraged the island model topologies and migration policies to show significant improvement in evolving Pareto fronts for DJSS problems. Further analysis was conducted to reveal the characteristics of frequently selected training instances. The findings were consisted with the assumptions made earlier about the importance of selecting potentially more useful training instances.

Having explored the ideas for evolution of multiple dispatching rules for different scenarios for DJSS problems using GPHH in the previous two

chapters, these ideas were brought to a conclusion by extending them successfully to the multi-objective problems. The first two chapters mainly focused on improving the effectiveness of GPHH while considering the effect uncertainty in processing times in DJSS problems. This chapter was successful in developing algorithms which are both effective as well as efficient while dealing with more complex problems involving multiple-objectives.

Chapter 6

Conclusions

The overall goal of this thesis is to develop effective and efficient genetic programming based hyper-heuristic approaches using active learning techniques for dynamic job shop scheduling problems for one or more objectives.

We successfully achieved this goal in this thesis. In order to achieve this goal, we developed new GPHH approaches and new active sampling methods to evolve effective dispatching rules for dynamic job shop scheduling problems. More specifically we developed new GPHH methods to evolve dispatching rules considering uncertain processing times, a new GPHH framework which incorporates active learning methods to evolve scenario-specific dispatching rules, and an efficient active sampling heuristic for parallel GPHH toward multi-objective DJSS.

The rest of this chapter presents the conclusions and highlights from each of the research objectives. The summary of research is presented where main conclusions are described with discussions on key issues and findings. Finally, potential research directions for future work are suggested.

6.1 Achieved Objectives

In this thesis, the following research objectives have been fulfilled:

- The dynamic job shop scheduling environment is characterized by uncertainty and this thesis specifically focuses on uncertain processing times. Through this thesis, two new GPHH approaches were developed to solve DJSS problems under uncertain processing times (Chapter 3).

  The first approach consists of three methods for incorporating the uncertainty information into the dispatching rules. This thesis presents a first study which considers incorporation of uncertainty in processing times into GPHH. In the first method (EMA), a new representation for genetic programming is developed by proposing a new terminal. The other two methods, ENT and EXP propose new training methodologies to incorporate the uncertainty information. These methods are compared with standard GPHH approach. All the three methods outperformed the standard GPHH approach. In particular, ENT method was the best and it also showed better generalization.

  In the second approach, a new method to identify the bottleneck and non-bottleneck machines for the dynamic scheduling environment is developed. Then a new cooperative co-evolutionary method is developed to evolve dispatching rules for each type of machine. This method is compared with standard GPHH approach and an existing method from literature called GP3. GP3 also considers bottleneck and non-bottleneck machines but for static scheduling problems. The proposed method significantly outperforms the existing approaches.

  These two approaches address the issue of uncertainty in processing times which manifests due to the dynamic nature of the shop environment. In particular, two varying *scenarios* which arise in this dynamic environment, the bottleneck machine scenario and the non-bottleneck machine scenario are addressed. In order to co-evolve a pair of rules for each of these scenarios, we considered two differ-

ent types of training instances which highlighted the corresponding machine states. Essentially, we used the idea of using different samples of training instances to evolve rules for different shop scenarios. The next research objective explores this idea by developing *active sampling* methods for GPHH to evolve multiple scenario-specific dispatching rules.

- Through this thesis, a new GPHH framework was developed which introduces new elements into GPHH in order to facilitate the proposed active sampling mechanism. In particular, a validation stage was introduced into GPHH which is a unique characteristic of the proposed framework compared with the existing ones. A new method to extract features from instances of DJSS problem under uncertainty was developed to *cluster* the training instances. This is also one of the preliminary requirements for the active sampling methods. Two new active sampling methods, respectively based on ϵ-greedy and Gaussian process bandits (GPB) approaches were developed.

  This thesis presents the first work which considers use of GPB as an active learning method for GPHH approaches. These active sampling methods were integrated into the new GPHH framework to identify potentially useful training instances and evolving multiple dispatching rules corresponding to different shop scenarios identified using the clustering approach. Essentially, the two active sampling methods have been leveraged to tackle the exploration versus exploitation dilemma which arises when it is required that we explore the space of DJSS training instances while exploiting the already identified good training instances to evolve dispatching rules.

  Finally, we also developed a method to associate the scenario-specific evolved rules with unseen DJSS problem instances.The GPHH approach based on the GPB method outperformed the standard GPHH approach as well as the one using ϵ-greedy method.

This research objective focused on evolving scenario-specific dispatching rules using GPHH with the help of active learning methods for a *single* scheduling objective. The next research objective develops new active sampling methods for DJSS problems which consider *multiple* objectives.

- This thesis developed a new active sampling heuristic for GPHH toward evolving an effective Pareto set of dispatching rules for multi-objective dynamic job shop scheduling problems. The first step in achieving this objective was to investigate the island model approach for parallel evolutionary algorithms under the purview of GPHH for static and multi-objective JSS problems. Different topologies for the island model were evaluated. It was empirically verified that with lower computational cost than standard GP, the performance of island model approach was significantly better. We achieved this by identifying the appropriate design parameters for the island model, particularly its topology and migration policy, which is known to promote the evolutionary algorithms to get out of local optima [220].

  Furthermore, leveraging the migration policies of the island models a successive reject heuristic for active sampling was developed with the aim of identifying potentially better training instances for GPHH. The proposed *successive reject heuristic* successfully achieved this aim by taking advantage of the inherent ability of island models to tackle exploration versus exploitation through its topology and migration policies. The proposed GPHH method using the new sampling heuristic outperformed both the standard GPHH and the island model based parallelized approach to GPHH.

6.2 Major Conclusions

We present the summary of research highlighting the main conclusions and present some discussions.

6.2.1 DJSS under Uncertain Processing Times

This thesis is the first work which explores the ability of GPHH to evolve dispatching rules for DJSS problems under uncertainty. In chapter 3, we develop methods which are successful in taking into consideration the effect of uncertainty in processing times in the DJSS problems. In particular, we developed simple yet effective new training approaches for GPHH leveraging the flexible representation of genetic programs. Moreover, even though the interpretability of the evolved dispatching rules is still hard, the genetic programs still give more insight when compared to say a neural network. Due to this reason it was possible to incorporate the uncertainty information directly into the terminals.

The ability of genetic programs to capture complex characteristics of a system was utilized to co-evolve dispatching rules for varying characteristics of machines, which is a manifestation of varying uncertainty levels of the dynamic job shop. Furthermore, cooperative co-evolution was successfully used to co-evolve a pair of dispatching rules which closely interact with each other during the sequencing process.

Scenario specific dispatching rules

Furthermore, in the dynamic job shops under uncertain processing times, two different types of machines, namely bottleneck and non-bottleneck machines are considered. Using appropriate machine features a clustering method was used to identify these machines with varying characteristics in a dynamic environment. Having successfully evolved a pair of dispatching rules for each of these machines, it was further validated that

using a single universal dispatching rules for all scenarios is not effective. In a complex shop environment, there are many other dynamic factors viz. varying job characteristics which give rise to different shop scenarios. Therefore, after evolving the machine specific rules, the next step was to consider the different complex shop scenarios arising in the shop and exploring the ability of GPHH to evolve scenario-specific rules. But when more of these scenarios need to be considered, there is an important problem of effectively sampling useful training instances which are addressed through the newly proposed active sampling techniques.

6.2.2 Toward Evolving Dispatching Rules for Multiple Shop Scenarios

A complex shop environment is defined by varying characteristics of jobs, different job arrival patterns, uncertainty in shop parameters and dynamic events like breakdown etc. The combinations of these factors lead to a very high number of shop scenarios. In Chapter 4, methods were developed to effectively identifying those scenarios which demand a more specific dispatching rule and evolve them using GPHH. A feature extraction and clustering methodology to associate the DJSS problem instances to the shop scenarios was developed, similar to the clustering approach employed in the context of the two scenarios arising due to the bottleneck levels of machines.

Active sampling in GPHH for DJSS problems

The very high number of shop scenarios leads to a large input space of DJSS problem instances and therefore active learning methods were required for effective sampling of the instances. Active sampling requires a method to measure the potential of a DJSS problem instance to promote evolution of effective dispatching rules. To this end, a new GPHH framework was developed by introducing a validation stage which facilitates

the quantification of the ability of DJSS problem instances to support evolution of good rules. Essentially, the validation stage of the GPHH is responsible for *exploration* of the input space of DJSS problem instances and evolution of dispatching rules using the identified DJSS instances is *exploitation*. The dilemma between exploration and exploitation arises when the two tasks need to be performed under a limited computational budget.

This dilemma of exploration versus exploitation has been well studied in the multi-armed bandit theory [16]. We developed new GPHH approaches which employ the active learning techniques based on the multi-armed bandit approaches and tackled the exploration versus exploitation dilemma in the context of active sampling of DJSS instances.

The first of the active sampling methods, ϵ-greedy heuristic, considered only *four* possible dispatching rules. For considering larger number (hundreds) of scenarios we integrated Gaussian process bandits as the active sampling method with GPHH. The new GPHH approaches using these active sampling methods could evolve effective scenario-specific dispatching rules. To associate the dispatching rules with unseen DJSS instances and solve them, the feature extraction methodology used earlier was extended.

We empirically evaluated the performance of the GPHH approaches using the two active sampling methods and the existing GPHH framework. Significant improvement was observed over most of the test sets. The main conclusion derived from Chapter 4 is that the new GPHH approaches developed using the active sampling methods utilizing the clustering methodology were successful in evolving effective scenario-specific dispatching rules.

6.2.3 Active Sampling Heuristics for GPHH toward Multi-Objective JSS

After considering the complex shop scenarios for DJSS problems for a single objective in Chapter 4, the advantages of using active learning for GPHH were explored for DJSS problems when multiple scheduling objectives are involved. GPHH for evolving dispatching rules for multi-objective scheduling problems is more difficult due to more complex search space [57]. Moreover, since for multi-objective scheduling problems a Pareto set of dispatching rules is evolved the GPHH framework consisting of validation stage becomes very expensive. This is because for evaluating a Pareto front requires comparing many metrics such as hypervolume and inverted generational distance.

For dealing with these computational issues in multi-objective scheduling a parallel GPHH framework was considered in Chapter 5. In particular, island model for parallelization was considered due to its qualities like ability to tackle problem of local optima and the inherent dynamics of exploration and exploitation in its subpopulations.

Since the efficacy of island model relies heavily on its design parameters, an empirical investigation of the different topologies of island model was conducted. Another salient point of this investigation is that we considered energy-aware scheduling by using minimization of energy cost as one of the scheduling objectives. This is a first study which considers energy as an objective while evolving dispatching rules for scheduling using GPHH. The results from this investigation were used to determine appropriate design parameters for island model which are effective toward evolving dispatching rules using GPHH for JSS problems. With the help of this supporting empirical evidence we identified a topology which was very effective and efficient for GPHH.

Active sampling heuristics using island model

Similar to the single objective case, it is important to identify those DJSS instances which highlight those shop scenarios which highlight the conflict between the scheduling objectives. An active sampling technique, called successive reject heuristic was developed in Chapter 5. The key findings from the earlier experiment encouraged us to use the island model topologies and migration policies for integrating the active sampling techniques for GPHH in the multi-objective JSS case. One key property of this sampling heuristic is that it successively narrows down the search space of training instances using the different islands. The performance of this approach is very good, significantly outperforming the existing approaches using most of the metrics namely hypervolume, inverted generational distance and spread which are used for comparing EMO algorithms.

On further analyzing the characteristics of frequently selected training instances, more insights were obtained providing more clarity on the reasons for the favourable outcome. In particular, we could observe that the selected DJSS instances highlighted the conflict between the scheduling objectives. It further highlighted the necessity of active sampling approaches for GPHH.

6.3 Future Work

This thesis presents the research in the direction of addressing the difficulties evolving effective dispatching rules for a dynamic job shop scheduling problem using GPHH in a complex and uncertain shop environment. There are many directions which could be further explored. Some of these are presented below.

- The flexible representation of genetic programs was leveraged to incorporate uncertainty information into the dispatching rules.However, in the whole thesis only uncertainty in *processing times* was consid-

ered. There are many other dynamic factors like sudden arrival of jobs, sudden change in priority of jobs, etc., which are responsible for the uncertainty in the shop environment. Therefore, in order to further achieve the goal of practical scheduling approaches, GPHH techniques which incorporate more such factors must be studied.

For multi-objective scheduling, this becomes even more difficult. For example, the bottleneck and non-bottleneck machines are associated with specific dispatching rules. But for a multi-objective JSS problem, the solution is a Pareto front corresponding to a set of dispatching rules. Therefore, when different classes of machines are considered, more nuanced methods are required to handle the multi-objective problems.

- This thesis developed methods to extract features from DJSS problem instances which are essentially numeric. The shop is a complicated system with varying characteristics and complex interaction between its components. Therefore, it is important to develop better feature extraction techniques which can capture the complexity of the shop. For example, structural features represented using graph data structures could be extracted from the DJSS instances. This will require more sophisticated feature selection and pattern recognition techniques in combination with some pre-processing steps.

- The research in the application of parallel evolutionary algorithms, island models in particular is growing. This thesis presented a simple heuristic toward active sampling of training instances. There is a clear scope for incorporating the techniques from the theory of multi-armed bandit, though it is comparatively more complicated due to the involvement of multiple objectives. Furthermore, the relationship between the selected training instances and the regions of Pareto front is not studied. This will help in developing new EMO algorithms for JSS with significantly improved performance of the

Pareto set of dispatching rules. In order to address the computational challenges, more topologies and migration policies of the island model should be investigated.

Bibliography

[1] Genetic Programming. https://upload.wikimedia.org/wikipedia/commons/7/77/Genetic_Program_Tree.png. Accessed: 2015-10-04.

[2] ADAMS, J., BALAS, E., AND ZAWACK, D. The shifting bottleneck procedure for job shop scheduling. *Management science 34*, 3 (1988), 391–401.

[3] AGGARWAL, C. C., KONG, X., GU, Q., HAN, J., AND PHILIP, S. Y. Active learning: A survey. In *Data Classification*. Chapman and Hall/CRC, 2014, pp. 589–623.

[4] AGHASSI, M., AND BERTSIMAS, D. Robust game theory. *Mathematical Programming 107*, 1-2 (2006), 231–273.

[5] AGRAWAL, P., AND RAO, S. Energy-aware scheduling of distributed systems. *Automation Science and Engineering, IEEE Transactions on 11*, 4 (2014), 1163–1175.

[6] AGUIRRE, A. M., LIU, S., AND PAPAGEORGIOU, L. G. Mixed integer linear programming based approaches for medium-term planning and scheduling in multiproduct multistage continuous plants. *Industrial & Engineering Chemistry Research 56*, 19 (2017), 5636–5651.

[7] ALBA, E. *Parallel metaheuristics: a new class of algorithms*, vol. 47. John Wiley & Sons, 2005.

[8] ALBA, E., AND DORRONSORO, B. *Cellular genetic algorithms*, vol. 42. Springer Science & Business Media, 2009.

[9] ALBA, E., AND TROYA, J. M. A survey of parallel distributed genetic algorithms. *Complexity 4*, 4 (1999), 31–52.

[10] ALLAHVERDI, A., PESCH, E., PINEDO, M., AND WERNER, F. Scheduling in manufacturing systems: new trends and perspectives, 2018.

[11] AMAYA, I., ORTIZ-BAYLISS, J. C., ROSALES-PEREZ, A., GUTIERREZ-RODRIGUEZ, A. E., CONANT-PABLOS, S. E., TERASHIMA-MARIN, H., AND COELLO, C. A. C. Enhancing selection hyper-heuristics via feature transformations. *IEEE Computational Intelligence Magazine 13*, 2 (2018), 30–41.

[12] ANTOS, A., GROVER, V., AND SZEPESVÁRI, C. Active learning in multi-armed bandits. In *International Conference on Algorithmic Learning Theory* (2008), Springer, pp. 287–302.

[13] ARAUJO, L., MERELO, J. J., MORA, A., AND COTTA, C. Genotypic differences and migration policies in an island model. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation* (2009), ACM, pp. 1331–1338.

[14] ARIK, O. A., AND TOKSARI, M. D. Multi-objective fuzzy parallel machine scheduling problems under fuzzy job deterioration and learning effects. *International Journal of Production Research 56*, 7 (2018), 2488–2505.

[15] ARUNARANI, A., MANJULA, D., AND SUGUMARAN, V. Task scheduling techniques in cloud computing: A literature survey. *Future Generation Computer Systems 91* (2019), 407–415.

[16] AUER, P., CESA-BIANCHI, N., AND FISCHER, P. Finite-time analysis of the multiarmed bandit problem. *Machine learning 47*, 2-3 (2002), 235–256.

[17] AYTUG, H., LAWLEY, M. A., MCKAY, K., MOHAN, S., AND UZSOY, R. Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research 161*, 1 (2005), 86–110.

[18] BÄCK, T. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

[19] BÄCK, T., FOGEL, D. B., AND MICHALEWICZ, Z. *Evolutionary computation 1: Basic algorithms and operators*. CRC press, 2018.

[20] BADER-EL-DEN, M., POLI, R., AND FATIMA, S. Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework. *Memetic Computing 1*, 3 (2009), 205–219.

[21] BARMAN, S. Simple priority rule combinations: an approach to improve both flow time and tardiness. *International Journal of Production Research 35*, 10 (1997), 2857–2870.

[22] BELIËN, J., GOOSSENS, D., AND VAN REETH, D. A mixed integer programming model for ex post optimization in fantasy sport games. In *European Conference on Operational Research (EURO XXVI-2013)* (2013), pp. 423–423.

[23] BERGSTRA, J., AND BENGIO, Y. Random search for hyper-parameter optimization. *Journal of Machine Learning Research 13*, Feb (2012), 281–305.

[24] BERGSTRA, J. S., BARDENET, R., BENGIO, Y., AND KÉGL, B. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems* (2011), pp. 2546–2554.

[25] BERTELS, A. R., AND TAURITZ, D. R. Why asynchronous parallel evolution is the future of hyper-heuristics: A cdcl sat solver case study. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion* (2016), ACM, pp. 1359–1365.

[26] BERTSIMAS, D., AND SIM, M. The price of robustness. *Operations research 52*, 1 (2004), 35–53.

[27] BIEGEL, J. E., AND DAVERN, J. J. Genetic algorithms and job shop scheduling. *Computers & Industrial Engineering 19*, 1-4 (1990), 81–91.

[28] BISHOP, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[29] BLUM, C., AND SAMPELS, M. An ant colony optimization algorithm for shop scheduling problems. *Journal of Mathematical Modelling and Algorithms 3*, 3 (2004), 285–308.

[30] BÖLTE, A., AND THONEMANN, U. W. Optimizing simulated annealing schedules with genetic programming. *European Journal of Operational Research 92*, 2 (1996), 402–416.

[31] BONABEAU, E., MARCO, D. D. R. D. F., DORIGO, M., THÉRAULAZ, G., THERAULAZ, G., ET AL. *Swarm intelligence: from natural to artificial systems*. No. 1. Oxford university press, 1999.

[32] BOUNEFFOUF, D., LAROCHE, R., URVOY, T., FÉRAUD, R., AND ALLESIARDO, R. Contextual bandit for active learning: Active thompson sampling. In *International Conference on Neural Information Processing* (2014), Springer, pp. 405–412.

[33] BRANKE, J., HILDEBRANDT, T., AND SCHOLZ-REITER, B. Hyper-heuristic evolution of dispatching rules: A comparison of rule representations. *Evolutionary computation* (2014).

[34] BRANKE, J., NGUYEN, S., PICKARDT, C., AND ZHANG, M. Automated design of production scheduling heuristics: A review.

[35] BRANKE, J., NGUYEN, S., PICKARDT, C. W., AND ZHANG, M. Automated design of production scheduling heuristics: A review. *IEEE Transactions on Evolutionary Computation 20*, 1 (2016), 110–124.

[36] BRANKE, J., SCHMECK, H., DEB, K., AND MAHESHWAR, R. S. Parallelizing multi-objective evolutionary algorithms: cone separation. In *IEEE Congress on Evolutionary Computation* (2004), vol. 2, pp. 1952–1957.

[37] BROCHU, E., CORA, V. M., AND DE FREITAS, N. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599* (2010).

[38] BURKE, E., KENDALL, G., SILVA, D. L., O'BRIEN, R., AND SOUBEIGA, E. An ant algorithm hyperheuristic for the project presentation scheduling problem. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on* (2005), vol. 3, IEEE, pp. 2263–2270.

[39] BURKE, E. K., GENDREAU, M., HYDE, M., KENDALL, G., OCHOA, G., ÖZCAN, E., AND QU, R. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society 64*, 12 (2013), 1695–1724.

[40] BURKE, E. K., HYDE, M., KENDALL, G., OCHOA, G., ÖZCAN, E., AND WOODWARD, J. R. A classification of hyper-heuristic approaches. In *Handbook of metaheuristics*. Springer, 2010, pp. 449–468.

[41] BURKE, E. K., HYDE, M. R., KENDALL, G., OCHOA, G., OZCAN, E., AND WOODWARD, J. R. Exploring hyper-heuristic methodologies with genetic programming. In *Computational intelligence*. Springer, 2009, pp. 177–201.

[42] BURKE, E. K., HYDE, M. R., KENDALL, G., AND WOODWARD, J. Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation* (2007), ACM, pp. 1559–1565.

[43] CANTU-PAZ, E. *Efficient and accurate parallel genetic algorithms*, vol. 1. Springer Science & Business Media, 2000.

[44] CANTÚ-PAZ, E., AND GOLDBERG, D. E. Are multiple runs of genetic algorithms better than one? In *Genetic and Evolutionary Computation Conference* (2003), Springer, pp. 801–812.

[45] CARRIÓN, M., ZÁRATE-MIÑANO, R., AND DOMÍNGUEZ, R. A practical formulation for ex-ante scheduling of energy and reserve in renewable-dominated power systems: Case study of the iberian peninsula. *Energies 11*, 8 (2018), 1939.

[46] CHAN, F., CHAN, H., LAU, H., AND IP, R. Analysis of dynamic dispatching rules for a flexible manufacturing system. *Journal of Materials Processing Technology 138*, 1-3 (2003), 325–331.

[47] CHEN, C., AND YIH, Y. Indentifying attributes for knowledge-based development in dynamic scheduling environments. *International Journal of Production Research 34*, 6 (1996), 1739–1755.

[48] CHENG, R., GEN, M., AND TSUJIMURA, Y. A tutorial survey of job-shop scheduling problems using genetic algorithms, part ii: hybrid genetic search strategies. *Computers & Industrial Engineering 36*, 2 (1999), 343–364.

[49] CHENG, T., GUPTA, M., ET AL. Survey of scheduling research involving due date determination decisions. *European journal of operational research 38*, 2 (1989), 156–166.

[50] CHURCH, L. K., AND UZSOY, R. Analysis of periodic and event-driven rescheduling policies in dynamic shops. *International Journal of Computer Integrated Manufacturing 5*, 3 (1992), 153–163.

[51] COHN, D. A., GHAHRAMANI, Z., AND JORDAN, M. I. Active learning with statistical models. *Journal of artificial intelligence research 4* (1996), 129–145.

[52] CONWAY, R. W., MAXWELL, W. L., AND MILLER, L. W. *Theory of scheduling*. Courier Corporation, 2003.

[53] COWLING, P., KENDALL, G., AND SOUBEIGA, E. A hyperheuristic approach to scheduling a sales summit. In *International Conference on the Practice and Theory of Automated Timetabling* (2000), Springer, pp. 176–190.

[54] CRAINIC, T. G., AND TOULOUSE, M. Parallel strategies for meta-heuristics. In *Handbook of metaheuristics*. Springer, 2003, pp. 475–513.

[55] ČREPINŠEK, M., LIU, S.-H., AND MERNIK, M. Exploration and exploitation in evolutionary algorithms: A survey. *ACM Computing Surveys (CSUR) 45*, 3 (2013), 35.

[56] DAI, M., TANG, D., GIRET, A., SALIDO, M. A., AND LI, W. D. Energy-efficient scheduling for a flexible flow shop using an improved genetic-simulated annealing algorithm. *Robotics and Computer-Integrated Manufacturing 29*, 5 (2013), 418–429.

[57] DEB, K. Multi-objective optimization. In *Search methodologies*. Springer, 2014, pp. 403–449.

[58] DEB, K., AND JAIN, H. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: solving problems with box constraints. *Evolutionary Computation, IEEE Transactions on 18*, 4 (2014), 577–601.

[59] DEB, K., PRATAP, A., AGARWAL, S., AND MEYARIVAN, T. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation 6*, 2 (2002), 182–197.

[60] DELL'AMICO, M., AND TRUBIAN, M. Applying tabu search to the job-shop scheduling problem. *Annals of Operations research 41*, 3 (1993), 231–252.

[61] DEVECI, M., AND DEMIREL, N. C. A survey of the literature on airline crew scheduling. *Engineering Applications of Artificial Intelligence 74* (2018), 54–69.

[62] DIMOPOULOS, C., AND ZALZALA, A. M. Investigating the use of genetic programming for a classic one-machine scheduling problem. *Advances in Engineering Software 32*, 6 (2001), 489–498.

[63] DOLGUI, A., IVANOV, D., SETHI, S. P., AND SOKOLOV, B. Scheduling in production, supply chain and industry 4.0 systems by optimal control: fundamentals, state-of-the-art and applications. *International Journal of Production Research 57*, 2 (2019), 411–432.

[64] DORIGO, M., AND BIRATTARI, M. Ant colony optimization. In *Encyclopedia of machine learning*. Springer, 2011, pp. 36–39.

[65] DU, W., TANG, Y., LEUNG, S. Y. S., TONG, L., VASILAKOS, A. V., AND QIAN, F. Robust order scheduling in the discrete manufacturing industry: A multiobjective optimization approach. *IEEE Transactions on Industrial Informatics 14*, 1 (2018), 253–264.

[66] DUAN, C., DENG, C., GHARAEI, A., WU, J., AND WANG, B. Selective maintenance scheduling under stochastic maintenance quality with multiple maintenance actions. *International Journal of Production Research 56*, 23 (2018), 7160–7178.

[67] DURILLO, J. J., NEBRO, A. J., LUNA, F., AND ALBA, E. A study of master-slave approaches to parallelize nsga-ii. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on* (2008), IEEE, pp. 1–8.

[68] DURILLO, J. J., ZHANG, Q., NEBRO, A. J., AND ALBA, E. Distribution of computational effort in parallel moea/d. In *International Conference on Learning and Intelligent Optimization* (2011), Springer, pp. 488–502.

[69] EWALD, G., KUREK, W., AND BRDYS, M. A. Grid implementation of a parallel multiobjective genetic algorithm for optimized allocation of chlorination stations in drinking water distribution systems: Chojnice case study. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 38*, 4 (2008), 497–509.

[70] FRIEDRICH, T., OLIVETO, P. S., SUDHOLT, D., AND WITT, C. Analysis of diversity-preserving mechanisms for global exploration. *Evolutionary Computation 17*, 4 (2009), 455–476.

[71] FU, J., ZHAO, C., XU, Q., AND HO, T. C. Debottleneck of multistage material-handling processes via simultaneous hoist scheduling and production line retrofit. *Industrial & Engineering Chemistry Research 52*, 1 (2012), 123–133.

[72] FU, Y., DING, J., WANG, H., AND WANG, J. Two-objective stochastic flow-shop scheduling with deteriorating and learning effect in industry 4.0-based manufacturing system. *Applied Soft Computing 68* (2018), 847–855.

[73] FUKUNAGA, A. S. Automated discovery of local search heuristics for satisfiability testing. *Evolutionary computation 16*, 1 (2008), 31–61.

[74] GANTI, R., AND GRAY, A. G. Building bridges: viewing active learning from the multi-armed bandit lens. In *Proceedings of the*

Twenty-Ninth Conference on Uncertainty in Artificial Intelligence (2013), AUAI Press, pp. 232–241.

[75] GAREY, M. R., JOHNSON, D. S., AND SETHI, R. The complexity of flowshop and jobshop scheduling. *Mathematics of operations research 1*, 2 (1976), 117–129.

[76] GEIGER, C. D., UZSOY, R., AND AYTUĞ, H. Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach. *Journal of Scheduling 9*, 1 (2006), 7–34.

[77] GLOVER, F., AND LAGUNA, M. Tabu search. In *Handbook of combinatorial optimization*. Springer, 1998, pp. 2093–2229.

[78] GONZALEZ, T., AND SAHNI, S. Flowshop and jobshop schedules: complexity and approximation. *Operations research 26*, 1 (1978), 36–52.

[79] GORGES-SCHLEUTER, M., AND GORGES-SCHLEUTER, M. *Genetic algorithms and population structures: a massively parallel algorithm*. 1991.

[80] GRAJDEANU, A. Parallel models for evolutionary algorithms. *ECLab, George Mason University 38* (2003).

[81] GUO, Y., AND GREINER, R. Optimistic active-learning using mutual information. In *IJCAI* (2007), vol. 7, pp. 823–829.

[82] GUPTA, A., ONG, Y.-S., AND FENG, L. Multifactorial evolution: toward evolutionary multitasking. *IEEE Transactions on Evolutionary Computation 20*, 3 (2016), 343–357.

[83] GÜREL, S., AND CINCIOĞLU, D. Rescheduling with controllable processing times for number of disrupted jobs and manufacturing cost objectives. *International Journal of Production Research 53*, 9 (2015), 2751–2770.

[84] HANSEN, N., AND OSTERMEIER, A. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on* (1996), IEEE, pp. 312–317.

[85] HANSEN, P., AND MLADENOVIĆ, N. Variable neighborhood search: Principles and applications. *European journal of operational research 130*, 3 (2001), 449–467.

[86] HARTER, A., TAURITZ, D. R., AND SIEVER, W. M. Asynchronous parallel cartesian genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (2017), ACM, pp. 1820–1824.

[87] HEGER, J., BRANKE, J., HILDEBRANDT, T., AND SCHOLZ-REITER, B. Dynamic adjustment of dispatching rule parameters in flow shops with sequence-dependent set-up times. *International Journal of Production Research 54*, 22 (2016), 6812–6824.

[88] HEGER, J., HILDEBRANDT, T., AND SCHOLZ-REITER, B. Switching dispatching rules with gaussian processes. In *Robust Manufacturing Control*. Springer, 2013, pp. 91–103.

[89] HEGER, J., HILDEBRANDT, T., AND SCHOLZ-REITER, B. Dispatching rule selection with gaussian processes. *Central European Journal of Operations Research 23*, 1 (2015), 235–249.

[90] HEIDRICH-MEISNER, V. Interview with richard s. sutton., 2009.

[91] HELD, M., AND KARP, R. M. The traveling-salesman problem and minimum spanning trees: Part ii. *Mathematical programming 1*, 1 (1971), 6–25.

[92] HILDEBRANDT, T. Jasima – an efficient java simulator for manufacturing and logistics. *http://code.google.com/p/jasima* (2012).

[93] HILDEBRANDT, T. Jasima; an efficient java simulator for manufacturing and logistics. *Last accessed 16* (2012).

[94] HILDEBRANDT, T., AND BRANKE, J. On using surrogates with genetic programming. *Evolutionary computation 23*, 3 (2015), 343–367.

[95] HOLLOWAY, C. A., AND NELSON, R. T. Job shop scheduling with due dates and variable processing times. *Management Science 20*, 9 (1974), 1264–1275.

[96] HOLT, C. C. Forecasting seasonals and trends by exponentially weighted moving averages. *International journal of forecasting 20*, 1 (2004), 5–10.

[97] HONKOMP, S., MOCKUS, L., AND REKLAITIS, G. A framework for schedule evaluation with processing uncertainty. *Computers & chemical engineering 23*, 4 (1999), 595–609.

[98] HUANG, K., YANG, H., KING, I., AND LYU, M. R. Local learning vs. global learning: An introduction to maxi-min margin machine. In *Support vector machines: theory and applications*. Springer, 2005, pp. 113–131.

[99] HUANG, K.-L., AND LIAO, C.-J. Ant colony optimization combined with taboo search for the job shop scheduling problem. *Computers & operations research 35*, 4 (2008), 1030–1046.

[100] HUNT, R., JOHNSTON, M., AND ZHANG, M. Evolving less-myopic scheduling rules for dynamic job shop scheduling with genetic programming. In *Proceedings of the 2014 conference on Genetic and evolutionary computation* (2014), ACM, pp. 927–934.

[101] HUNT, R., JOHNSTON, M., AND ZHANG, M. Evolving "less-myopic" scheduling rules for dynamic job shop scheduling with genetic programming. In *Proceedings of the 2014 Annual Conference*

on Genetic and Evolutionary Computation (New York, NY, USA, 2014), GECCO '14, ACM, pp. 927–934.

[102] ILLETSKOVA, M., BERTELS, A. R., TUGGLE, J. M., HARTER, A., RICHTER, S., TAURITZ, D. R., MULDER, S., BUENO, D., LEGER, M., AND SIEVER, W. M. Improving performance of cdcl sat solvers by automated design of variable selection heuristics. In *Computational Intelligence (SSCI), 2017 IEEE Symposium Series on* (2017), IEEE, pp. 1–8.

[103] ISHIBUCHI, H., AND MURATA, T. A multi-objective genetic local search algorithm and its application to flowshop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 28*, 3 (1998), 392–403.

[104] JACOBS, T. W., AND GIL, A. E. Method and system for print shop job routing, Nov. 3 2015. US Patent 9,176,690.

[105] JAIN, S., AND FOLEY, W. Dispatching strategies for managing uncertainties in automated manufacturing systems. *European Journal of Operational Research 248*, 1 (2016), 328–341.

[106] JAKOBOVIĆ, D., AND BUDIN, L. *Dynamic Scheduling with Genetic Programming*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 73–84.

[107] JAKOBOVIĆ, D., AND BUDIN, L. Dynamic scheduling with genetic programming. In *Genetic Programming*. Springer, 2006, pp. 73–84.

[108] JAKOBOVIĆ, D., JELENKOVIĆ, L., AND BUDIN, L. Genetic programming heuristics for multiple machine scheduling. In *Genetic Programming*. Springer, 2007, pp. 321–330.

[109] JAMRUS, T., CHIEN, C.-F., GEN, M., AND SETHANAN, K. Hybrid particle swarm optimization combined with genetic operators

for flexible job-shop scheduling under uncertain processing time for semiconductor manufacturing. *IEEE Transactions on Semiconductor Manufacturing 31*, 1 (2017), 32–41.

[110] JANAK, S. L., LIN, X., AND FLOUDAS, C. A. A new robust optimization approach for scheduling under uncertainty: Ii. uncertainty with known probability distribution. *Computers & chemical engineering 31*, 3 (2007), 171–195.

[111] JAYAMOHAN, M., AND RAJENDRAN, C. New dispatching rules for shop scheduling: a step forward. *International Journal of Production Research 38*, 3 (2000), 563–586.

[112] JENSEN, M. T. Generating robust and flexible job shop schedules using genetic algorithms. *Evolutionary Computation, IEEE Transactions on 7*, 3 (2003), 275–288.

[113] JIA, Z., AND IERAPETRITOU, M. G. Uncertainty analysis on the righthand side for milp problems. *AIChE journal 52*, 7 (2006), 2486–2495.

[114] JOHNSON, S. M. Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly 1*, 1 (1954), 61–68.

[115] KACPRZYK, J., AND PEDRYCZ, W. *Springer handbook of computational intelligence*. Springer, 2015.

[116] KARUNAKARAN, D., MEI, Y., CHEN, G., AND ZHANG, M. Dynamic job shop scheduling under uncertainty using genetic programming. *Intelligent and Evolutionary Systems* (2016), 195.

[117] KARUNAKARAN, D., MEI, Y., CHEN, G., AND ZHANG, M. Toward evolving dispatching rules for dynamic job shop scheduling under

uncertainty. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2017), ACM, pp. 282–289.

[118] KELLER, R. E., AND POLI, R. Cost-benefit investigation of a genetic-programming hyperheuristic. In *Artificial Evolution* (2008), Springer, pp. 13–24.

[119] KENNEDY, J. Particle swarm optimization. In *Encyclopedia of machine learning*. Springer, 2011, pp. 760–766.

[120] KIM, Y. K., PARK, K., AND KO, J. A symbiotic evolutionary algorithm for the integration of process planning and job shop scheduling. *Computers & operations research 30*, 8 (2003), 1151–1171.

[121] KIRKPATRICK, S., GELATT, C. D., AND VECCHI, M. P. Optimization by simulated annealing. *science 220*, 4598 (1983), 671–680.

[122] KNOWLES, J., AND CORNE, D. On metrics for comparing nondominated sets. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on* (2002), vol. 1, IEEE, pp. 711–716.

[123] KOOP, G., POIRIER, D. J., AND TOBIAS, J. L. *Bayesian econometric methods*. Cambridge University Press, 2007.

[124] KOUVELIS, P., DANIELS, R. L., AND VAIRAKTARAKIS, G. Robust scheduling of a two-machine flow shop with uncertain processing times. *Iie Transactions 32*, 5 (2000), 421–432.

[125] KOUVELIS, P., AND YU, G. *Robust discrete optimization and its applications*, vol. 14. Springer Science & Business Media, 2013.

[126] KOZA, J. R. *Genetic programming: on the programming of computers by means of natural selection*, vol. 1. MIT press, 1992.

[127] KREMER, J., STEENSTRUP PEDERSEN, K., AND IGEL, C. Active learning with support vector machines. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 4*, 4 (2014), 313–326.

[128] KU, W.-Y., AND BECK, J. C. Mixed integer programming models for job shop scheduling: A computational analysis. *Computers & Operations Research 73* (2016), 165–173.

[129] KULESHOV, V., AND PRECUP, D. Algorithms for multi-armed bandit problems. *arXiv preprint arXiv:1402.6028* (2014).

[130] KULKARNI, K., AND MANOHAR, P. Methods and systems for routing and scheduling print jobs, Mar. 9 2017. US Patent App. 14/848,445.

[131] KUTSUNA, N., HIGAKI, T., MATSUNAGA, S., OTSUKI, T., YAMAGUCHI, M., FUJII, H., AND HASEZAWA, S. Active learning framework with iterative clustering for bioimage classification. *Nature communications 3* (2012), 1032.

[132] LAND, A. H., AND DOIG, A. G. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society* (1960), 497–520.

[133] LÄSSIG, J., AND SUDHOLT, D. The benefit of migration in parallel evolutionary algorithms. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation* (2010), ACM, pp. 1105–1112.

[134] LÄSSIG, J., AND SUDHOLT, D. Experimental supplements to the theoretical analysis of migration in the island model. In *International Conference on Parallel Problem Solving from Nature* (2010), Springer, pp. 224–233.

[135] LÄSSIG, J., AND SUDHOLT, D. Adaptive population models for offspring populations and parallel evolutionary algorithms. In *Proceedings of the 11th workshop proceedings on Foundations of genetic algorithms* (2011), ACM, pp. 181–192.

[136] LÄSSIG, J., AND SUDHOLT, D. Analysis of speedups in parallel evolutionary algorithms and (1+ λ) eas for combinatorial optimization. *Theoretical Computer Science 551* (2014), 66–83.

[137] LAWRENCE, S. R., AND SEWELL, E. C. Heuristic, optimal, static, and dynamic schedules when processing times are uncertain. *Journal of Operations Management 15*, 1 (1997), 71–82.

[138] LENSTRA, J. K., AND KAN, A. R. Computational complexity of discrete optimization problems. *Annals of Discrete Mathematics 4* (1979), 121–140.

[139] LEÓN, C., MIRANDA, G., SEGREDO, E., AND SEGURA, C. Parallel hypervolume-guided hyperheuristic for adapting the multi-objective evolutionary island model. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2008)*. Springer, 2009, pp. 261–272.

[140] LEWIS, D. D., AND CATLETT, J. Heterogeneous uncertainty sampling for supervised learning. In *Machine Learning Proceedings 1994*. Elsevier, 1994, pp. 148–156.

[141] LI, Z., AND IERAPETRITOU, M. Process scheduling under uncertainty: Review and challenges. *Computers & Chemical Engineering 32*, 4 (2008), 715–727.

[142] LIN, X., JANAK, S. L., AND FLOUDAS, C. A. A new robust optimization approach for scheduling under uncertainty:: I. bounded uncertainty. *Computers & chemical engineering 28*, 6 (2004), 1069–1085.

[143] LIU, B., WANG, L., AND JIN, Y.-H. An effective pso-based memetic algorithm for flow shop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 37*, 1 (2007), 18–27.

[144] LIU, Y. *Multi-objective optimisation methods for minimising total weighted tardiness, electricity consumption and electricity cost in job shops through scheduling*. PhD thesis, University of Nottingham, 2014.

[145] LIU, Y., DONG, H., LOHSE, N., AND PETROVIC, S. Reducing environmental impact of production during a Rolling Blackout policy A multi-objective schedule optimisation approach. *Journal of Cleaner Production*, 0 (2015), .

[146] LIU, Y., MEI, Y., ZHANG, M., AND ZHANG, Z. A predictive-reactive approach with genetic programming and cooperative co-evolution for uncertain capacitated arc routing problem. *Evolutionary computation* (2019), 1–25.

[147] LUQUE, G., AND ALBA, E. *Parallel genetic algorithms: Theory and real world applications*, vol. 367. Springer, 2011.

[148] MARCO, N., PERIAUX, J., ET AL. A parallel genetic algorithm for multi-objective optimization in computational fluid dynamics.

[149] MEI, Y., NGUYEN, S., XUE, B., AND ZHANG, M. An efficient feature selection algorithm for evolving job shop scheduling rules with genetic programming. *IEEE Transactions on Emerging Topics in Computational Intelligence 1*, 5 (2017), 339–353.

[150] MEI, Y., AND ZHANG, M. Genetic programming hyper-heuristic for multi-vehicle uncertain capacitated arc routing problem. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (New York, NY, USA, 2018), GECCO '18, ACM, pp. 141–142.

[151] MICHALEWICZ, Z., AND FOGEL, D. B. *How to solve it: modern heuristics*. Springer Science & Business Media, 2013.

[152] MITCHELL, M. *An introduction to genetic algorithms*. MIT press, 1998.

[153] MIYASHITA, K. Job-shop scheduling with genetic programming. In *Proceedings of the 2Nd Annual Conference on Genetic and Evolutionary Computation* (San Francisco, CA, USA, 2000), GECCO'00, Morgan Kaufmann Publishers Inc., pp. 505–512.

[154] MULVEY, J. M., VANDERBEI, R. J., AND ZENIOS, S. A. Robust optimization of large-scale systems. *Operations research 43*, 2 (1995), 264–281.

[155] MYRDAL, G. *Monetary equilibrium*. London: W. Hodge, 1939.

[156] NGUYEN, S. *Automatic design of dispatching rules for dispatching rules for job shop scheduling with genetic programming*. PhD dissertation, Victoria University of Wellington, 2013.

[157] NGUYEN, S. Automatic design of dispatching rules for job shop scheduling with genetic programming.

[158] NGUYEN, S., MEI, Y., AND ZHANG, M. Genetic programming for production scheduling: a survey with a unified framework. *Complex & Intelligent Systems 3*, 1 (2017), 41–66.

[159] NGUYEN, S., AND ZHANG, M. A pso-based hyper-heuristic for evolving dispatching rules in job shop scheduling. In *Evolutionary Computation (CEC), 2017 IEEE Congress on* (2017), IEEE, pp. 882–889.

[160] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. A coevolution genetic programming method to evolve scheduling policies for dynamic multi-objective job shop scheduling problems. In *IEEE Congress on Evolutionary Computation* (2012), pp. 1–8.

[161] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. *Evolutionary Computation, IEEE Transactions on 17*, 5 (2013), 621–639.

[162] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. Dynamic multi-objective job shop scheduling: A genetic programming approach. In *Automated Scheduling and Planning*. Springer, 2013, pp. 251–282.

[163] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming. *IEEE Transactions on Evolutionary Computation 18*, 2 (2014), 193–208.

[164] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. Automatic programming via iterated local search for dynamic job shop scheduling. *Cybernetics, IEEE Transactions on 45*, 1 (2015), 1–14.

[165] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. Genetic programming for job shop scheduling. In *Evolutionary and Swarm Intelligence Algorithms*. Springer, 2019, pp. 143–167.

[166] NGUYEN, S., ZHANG, M., AND TAN, K. C. Surrogate-assisted genetic programming with simplified models for automated design of dispatching rules. *IEEE transactions on cybernetics 47*, 9 (2016), 2951–2965.

[167] NILAKANTAN, J. M., HUANG, G. Q., AND PONNAMBALAM, S. An investigation on minimizing cycle time and total energy consumption in robotic assembly line systems. *Journal of Cleaner Production 90* (2015), 311–325.

[168] NOWAK, K., IZZO, D., AND HENNES, D. Injection, saturation and feedback in meta-heuristic interactions. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation* (2015), ACM, pp. 1167–1174.

[169] ORÇUN, S., ALTINEL, İ. K., AND HORTAÇSU, Ö. Scheduling of batch processes with operational uncertainties. *Computers & chemical engineering 20* (1996), S1191–S1196.

[170] OUELHADJ, D., AND PETROVIC, S. A survey of dynamic scheduling in manufacturing systems. *Journal of scheduling 12*, 4 (2009), 417.

[171] ÖZDEMIR, S., BARAA, A. A., AND KHALIL, Ö. A. Multi-objective evolutionary algorithm based on decomposition for energy efficient coverage in wireless sensor networks. *Wireless personal communications 71*, 1 (2013), 195–215.

[172] PANWALKAR, S. S., AND ISKANDER, W. A survey of scheduling rules. *Operations research 25*, 1 (1977), 45–61.

[173] PARK, J., MEI, Y., NGUYEN, S., CHEN, G., JOHNSTON, M., AND ZHANG, M. *Genetic Programming Based Hyper-heuristics for Dynamic Job Shop Scheduling: Cooperative Coevolutionary Approaches*. Springer International Publishing, Cham, 2016, pp. 115–132.

[174] PARK, J., MEI, Y., NGUYEN, S., CHEN, G., AND ZHANG, M. Investigating a machine breakdown genetic programming approach for dynamic job shop scheduling. In *European Conference on Genetic Programming* (2018), Springer, pp. 253–270.

[175] PARK, J., MEI, Y., NGUYEN, S., CHEN, G., AND ZHANG, M. An investigation of ensemble combination schemes for genetic programming based hyper-heuristic approaches to dynamic job shop scheduling. *Applied Soft Computing 63* (2018), 72–86.

[176] PARK, J., NGUYEN, S., ZHANG, M., AND JOHNSTON, M. Evolving ensembles of dispatching rules using genetic programming for job shop scheduling. In *European Conference on Genetic Programming* (2015), Springer, pp. 92–104.

[177] PICKARDT, C. W., HILDEBRANDT, T., BRANKE, J., HEGER, J., AND SCHOLZ-REITER, B. Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems. *International Journal of Production Economics 145*, 1 (2013), 67–77.

[178] PILLAY, N., AND QU, R. Packing problems. In *Hyper-Heuristics: Theory and Applications*. Springer, 2018, pp. 67–73.

[179] PINEDO, M. *Planning and scheduling in manufacturing and services*, vol. 24. Springer, 2005.

[180] PINEDO, M., AND SINGER, M. A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop. *Naval Research Logistics 46*, 1 (1999), 1–17.

[181] PINEDO, M., AND WEISS, G. The largest variance first policy in some stochastic scheduling problems. *Operations Research 35*, 6 (1987), 884–891.

[182] PINEDO, M. L. *Scheduling: theory, algorithms, and systems*. Springer Science & Business Media, 2012.

[183] PIROUZI, S., AGHAEI, J., VAHIDINASAB, V., NIKNAM, T., AND KHODAEI, A. Robust linear architecture for active/reactive power scheduling of ev integrated smart distribution networks. *Electric Power Systems Research 155* (2018), 8–20.

[184] POLI, R., KENNEDY, J., AND BLACKWELL, T. Particle swarm optimization. *Swarm intelligence 1*, 1 (2007), 33–57.

[185] POTTER, M. A., AND DE JONG, K. A. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary computation 8*, 1 (2000), 1–29.

[186] POTTS, C. N., AND STRUSEVICH, V. A. Fifty years of scheduling: a survey of milestones. *Journal of the Operational Research Society* (2009), S41–S68.

[187] RABELO, L., AND JONES, A. Job shop scheduling. In *Encyclopedia of Operations Research and Management Science*, S. Gass and M. Fu, Eds. Springer US, 2013, pp. 817–830.

[188] RAHMATI, S. H. A., AHMADI, A., AND GOVINDAN, K. A novel integrated condition-based maintenance and stochastic flexible job

shop scheduling problem: simulation-based optimization approach. *Annals of Operations Research* (2018), 1–39.

[189] RAI, S. System and method for assigning print jobs to autonomous cells in a transaction printing environment, Nov. 15 2011. US Patent 8,059,292.

[190] RAI, S. Workflow scheduling method and system, May 13 2014. US Patent 8,725,546.

[191] RAI, S., DUKE, C. B., LOWE, V., QUAN-TROTTER, C., AND SCHEERMESSER, T. Ldp lean document production-or-enhanced productivity improvements for the printing industry. *Interfaces 39*, 1 (2009), 69–90.

[192] RAI, S., GODAMBE, A. V., DUKE, C. B., AND WILLIAMS, G. H. Printshop resource optimization via the use of autonomous cells, July 18 2006. US Patent 7,079,266.

[193] RAJENDRAN, C., AND HOLTHAUS, O. A comparative study of dispatching rules in dynamic flowshops and jobshops. *European journal of operational research 116*, 1 (1999), 156–170.

[194] RAMASESH, R. Dynamic job shop scheduling: a survey of simulation research. *Omega 18*, 1 (1990), 43–57.

[195] RASMUSSEN, C. E. Gaussian processes in machine learning. In *Advanced lectures on machine learning*. Springer, 2004, pp. 63–71.

[196] ROSENBLATT, M. J., AND LEE, H. L. Economic production cycles with imperfect production processes. *IIE transactions 18*, 1 (1986), 48–55.

[197] ROTHLAUF, F. Representations for evolutionary algorithms. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evo-*

lutionary Computation Conference: Late Breaking Papers (2009), ACM, pp. 3131–3156.

[198] SABOORI, B., SABOORI, B., CARLSON, J. S., AND SÖDERBERG, R. Introducing fast robot roller hemming process in automotive industry. *World Academy of Science, Engineering and Technology 3* (2009).

[199] SABUNCUOGLU, I., AND BAYIZ, M. Analysis of reactive scheduling problems in a job shop environment. *European Journal of operational research 126*, 3 (2000), 567–586.

[200] SAFARZADEGAN GILAN, S., GOYAL, N., AND DILKINA, B. Active learning in multi-objective evolutionary algorithms for sustainable building design. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016* (2016), ACM, pp. 589–596.

[201] SCHEIN, A. I., AND UNGAR, L. H. Active learning for logistic regression: an evaluation. *Machine Learning 68*, 3 (2007), 235–265.

[202] SEGURA, C., MIRANDA, G., AND LEÓN, C. Parallel hyperheuristics for the frequency assignment problem. *Memetic Computing 3*, 1 (2011), 33–49.

[203] SELS, V., GHEYSEN, N., AND VANHOUCKE, M. A comparison of priority rules for the job shop scheduling problem under different flow time-and tardiness-related objective functions. *International Journal of Production Research 50*, 15 (2012), 4255–4270.

[204] SETTLES, B. Active learning literature survey. 2010. *Computer Sciences Technical Report 1648* (2014).

[205] SETTLES, B., AND CRAVEN, M. An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the conference on empirical methods in natural language processing* (2008), Association for Computational Linguistics, pp. 1070–1079.

[206] SETTLES, B., CRAVEN, M., AND RAY, S. Multiple-instance active learning. In *Advances in neural information processing systems* (2008), pp. 1289–1296.

[207] SEUNG, H. S., OPPER, M., AND SOMPOLINSKY, H. Query by committee. In *Proceedings of the fifth annual workshop on Computational learning theory* (1992), ACM, pp. 287–294.

[208] SHA, D., AND LIN, H.-H. A multi-objective pso for job-shop scheduling problems. *Expert Systems with Applications 37*, 2 (2010), 1065–1070.

[209] SHIUE, Y.-R. Data-mining-based dynamic dispatching rule selection mechanism for shop floor control systems using a support vector machine approach. *International Journal of Production Research 47*, 13 (2009), 3669–3690.

[210] SHROUF, F., ORDIERES-MERÉ, J., GARCÍA-SÁNCHEZ, A., AND ORTEGA-MIER, M. Optimizing the production scheduling of a single machine to minimize total energy consumption costs. *Journal of Cleaner Production 67* (2014), 197–207.

[211] SKOLICKI, Z. An analysis of island models in evolutionary computation. In *Proceedings of the 7th annual workshop on Genetic and evolutionary computation* (2005), ACM, pp. 386–389.

[212] SKOLICKI, Z., AND DE JONG, K. The influence of migration sizes and intervals on island models. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation* (2005), ACM, pp. 1295–1302.

[213] SMITH-MILES, K. A., JAMES, R. J., GIFFIN, J. W., AND TU, Y. A knowledge discovery approach to understanding relationships between scheduling problem structure and heuristic performance.

In *International Conference on Learning and Intelligent Optimization* (2009), Springer, pp. 89–103.

[214] SORIA-ALCARAZ, J. A., ESPINAL, A., AND SOTELO-FIGUEROA, M. A. Evolvability metric estimation by a parallel perceptron for on-line selection hyper-heuristics. *IEEE Access 5* (2017), 7055–7063.

[215] SPRAVE, J. A unified model of non-panmictic population structures in evolutionary algorithms.

[216] SPRECHER, A., KOLISCH, R., AND DREXL, A. Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem. *European Journal of Operational Research 80*, 1 (1995), 94–102.

[217] SRINIVAS, N., KRAUSE, A., KAKADE, S. M., AND SEEGER, M. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995* (2009).

[218] STARNES, D. S., YATES, D., AND MOORE, D. S. *The practice of statistics*. Macmillan, 2010.

[219] SUBRAMANIAM, V., LEE, G., HONG, G., WONG, Y., AND RAMESH, T. Dynamic selection of dispatching rules for job shop scheduling. *Production planning & control 11*, 1 (2000), 73–81.

[220] SUDHOLT, D. Parallel evolutionary algorithms. In *Springer Handbook of Computational Intelligence*. Springer, 2015, pp. 929–959.

[221] SUN, L., LIN, L., GEN, M., AND LI, H. A hybrid cooperative coevolution algorithm for fuzzy flexible job shop scheduling. *IEEE Transactions on Fuzzy Systems* (2019).

[222] SUTTON, R. S., BARTO, A. G., BACH, F., ET AL. *Reinforcement learning: An introduction*. MIT press, 1998.

[223] TAILLARD, E. Benchmarks for basic scheduling problems. *european journal of operational research 64*, 2 (1993), 278–285.

[224] TAPIERO, C. S. Ex-post inventory control. *International journal of production research 38*, 6 (2000), 1397–1406.

[225] TAY, J. C., AND HO, N. B. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering 54*, 3 (2008), 453–473.

[226] TENNE, Y., AND GOH, C.-K. *Computational intelligence in expensive optimization problems*, vol. 2. Springer Science & Business Media, 2010.

[227] TKINDT, V., AND BILLAUT, J.-C. *Multicriteria scheduling: theory, models and algorithms*. Springer Science & Business Media, 2006.

[228] TOKIC, M., AND PALM, G. Value-difference based exploration: adaptive control between epsilon-greedy and softmax. In *Annual Conference on Artificial Intelligence* (2011), Springer, pp. 335–346.

[229] TOMASSINI, M. *Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time (Natural Computing Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[230] VAESSENS, R. J. M., AARTS, E. H., AND LENSTRA, J. K. Job shop scheduling by local search. *INFORMS Journal on Computing 8*, 3 (1996), 302–317.

[231] VAN DEN AKKER, M., AND HOOGEVEEN, H. Minimizing the number of late jobs in a stochastic setting using a chance constraint. *Journal of Scheduling 11*, 1 (2008), 59–69.

[232] VAN LAARHOVEN, P. J., AARTS, E. H., AND LENSTRA, J. K. Job shop scheduling by simulated annealing. *Operations research 40*, 1 (1992), 113–125.

[233] VAN ONSEM, W., AND DEMOEN, B. Parhyflex: A framework for parallel hyper-heuristics. In *Proceedings of the 25th Benelux Conference on Arti cial Intelligence* (2013), vol. 28, pp. 231–238.

[234] VERDERAME, P. M., ELIA, J. A., LI, J., AND FLOUDAS, C. A. Planning and scheduling under uncertainty: a review across multiple sectors. *Industrial & engineering chemistry research 49*, 9 (2010), 3993–4017.

[235] VOGIATZIS, D., AND TSAPATSOULIS, N. Active learning for microarray data. *International Journal of Approximate Reasoning 47*, 1 (2008), 85–96.

[236] VOUDOURIS, C., AND TSANG, E. P. Guided local search. In *Handbook of metaheuristics*. Springer, 2003, pp. 185–218.

[237] WANG, H., JIN, Y., AND DOHERTY, J. Committee-based active learning for surrogate-assisted particle swarm optimization of expensive problems. *IEEE transactions on cybernetics 47*, 9 (2017), 2664–2677.

[238] WANG, J.-Q., CHEN, J., ZHANG, Y., AND HUANG, G. Q. Schedule-based execution bottleneck identification in a job shop. *Computers & Industrial Engineering 98* (2016), 308–322.

[239] WANG, Y., JIN, X., XIE, L., ZHANG, Y., AND LU, S. Uncertain production scheduling based on fuzzy theory considering utility and production rate. *Information 8*, 4 (2017), 158.

[240] WERNER, F. Scheduling under uncertainty. *Unpublished document. Online at: https://feb. kuleuven. be/eng/tew/academic/prodbel/PMS2012/pdf% 20file% 20talk% 20werner [1]. pdf* (2012).

[241] WIEGAND, R. P. *An analysis of cooperative coevolutionary algorithms*. PhD thesis, George Mason University, 2003.

[242] WILCOXON, F. Individual comparisons by ranking methods. *Biometrics bulletin* (1945), 80–83.

[243] WITT, C. Worst-case and average-case approximations by simple randomized search heuristics. In *Annual Symposium on Theoretical Aspects of Computer Science* (2005), Springer, pp. 44–56.

[244] XIAO, N., AND ARMSTRONG, M. P. A specialized island model and its application in multiobjective optimization. In *Genetic and Evolutionary Computation Conference* (2003), Springer, pp. 1530–1540.

[245] YAO, X., LIU, Y., AND LIN, G. Evolutionary programming made faster. *IEEE Transactions on Evolutionary computation 3*, 2 (1999), 82–102.

[246] YAO, Y., PENG, Z., AND XIAO, B. Parallel hyper-heuristic algorithm for multi-objective route planning in a smart city. *IEEE Transactions on Vehicular Technology 67*, 11 (2018), 10307–10318.

[247] YIN, W.-J., LIU, M., AND WU, C. Learning single-machine scheduling heuristics subject to machine breakdowns with genetic programming. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on* (2003), vol. 2, IEEE, pp. 1050–1055.

[248] ZHANG, J., DING, G., ZOU, Y., QIN, S., AND FU, J. Review of job shop scheduling research and its new perspectives under industry 4.0. *Journal of Intelligent Manufacturing* (2017), 1–22.

[249] ZHANG, R., AND WU, C. Bottleneck identification procedures for the job shop scheduling problem with applications to genetic algorithms. *The International Journal of Advanced Manufacturing Technology 42*, 11-12 (2009), 1153–1164.

[250] ZITZLER, E., THIELE, L., LAUMANNS, M., FONSECA, C. M., AND DA FONSECA, V. G. Performance assessment of multiobjective op-

timizers: An analysis and review. *IEEE Transactions on evolutionary computation 7*, 2 (2003), 117–132.

[251] ZIZTLER, E., LAUMANNS, M., AND THIELE, L. Spea2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. *Evolutionary Methods for Design, Optimization, and Control* (2002), 95–100.

Appendices

Appendix A

Supplementary Results for Chapter 4

In Chapter 4, we presented a summary of our results to highlight the significant improvement observed in scheduling performance using our proposed algorithms, particularly GPB. Here we present more detailed boxplots corresponding to the different test sets considered in our experiments.

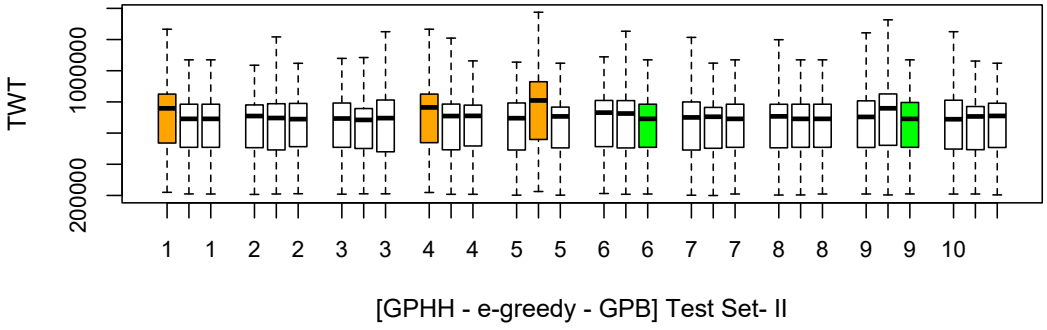


Figure A.1: Boxplots: Test Set II.

Since each test set consists of 30 DJSS instances, we randomly picked 10 instances from a test set and plotted the box plots for the total weighted tardiness (TWT) values obtained after scheduling using the evolved dis-

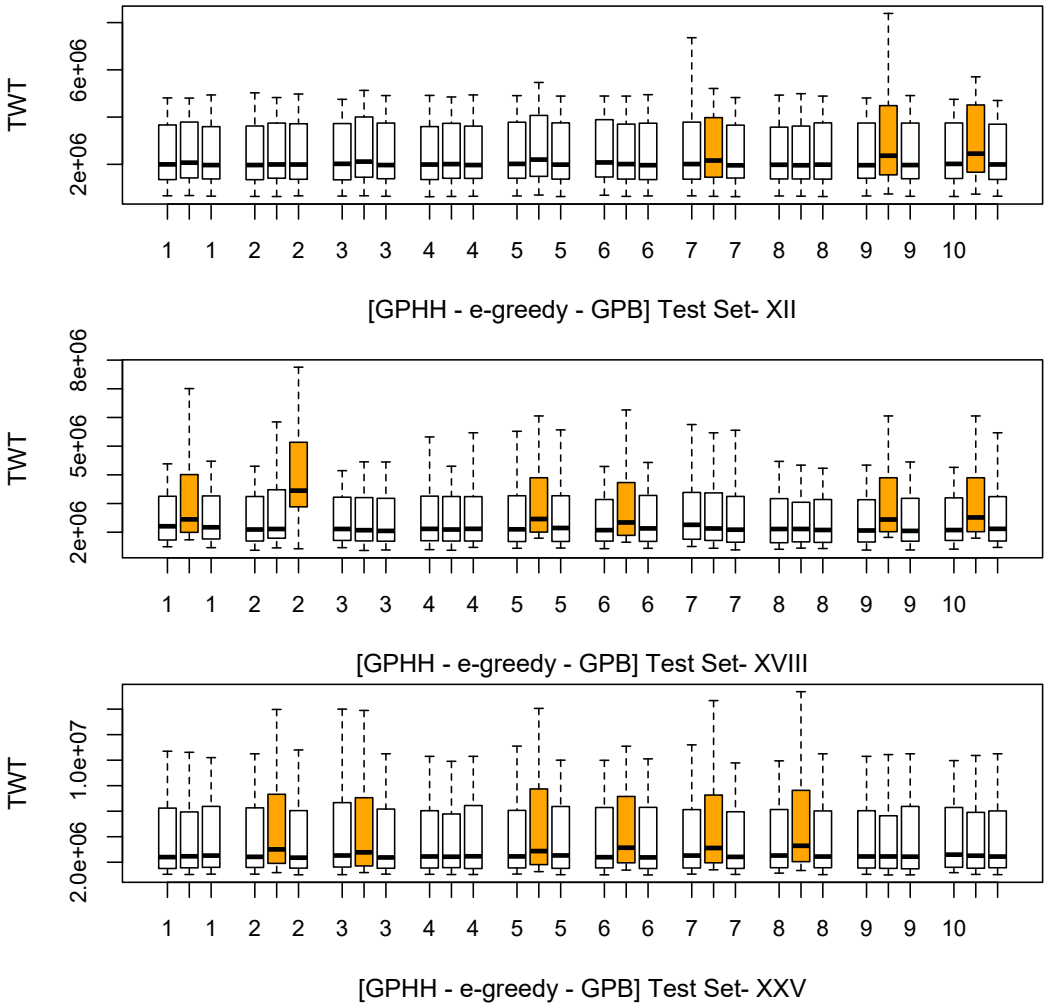


Figure A.2: Test Sets XII, XVIII and XXV: The result with highest medians are marked in orange.

patching rules obtained using each of the methods. The boxplots for the methods are ordered as GPHH, ϵ-greedy and GPB. Consider Fig. A.1 which shows boxplots for the test set II. Each triplet of boxplots (of a total of 10 such triplets) corresponds to a particular DJSS test instance. The boxplots marked in green are those whose medians are low (scheduling objective is to minimize tardiness) and indicate good performance. On the other hand,

the boxplots colored in orange indicate poor performance. We can see in Fig. A.1 that the performance of (standard) GPHH is significantly poor for the instances 1 and 4. We also point out that for the instances 6 and 9 as well, the performance of GPB is good. In Table 4.8, which summarizes the results using $[win\text{-}draw\text{-}lose]$, for test set II, the observed result was $[4\text{-}26\text{-}0]$. The boxplots in Fig. A.1 illustrate this observed fact. Furthermore, the performance of the ϵ-greedy method is poor for the test instance 5.

Similarly, for the test sets XII, XVIII and XXV the performance of the ϵ-greedy method was poor. This is illustrated through the boxplots in Fig. A.2. In Table 4.7, the corresponding result for the test sets XII, XVIII and XXV was $[1\text{-}3\text{-}26]$, $[1\text{-}5\text{-}24]$ and $[0\text{-}10\text{-}20]$ respectively. As expected, the performance of the ϵ-greedy method on a large number of DJSS test instances from these test sets is poor, as shown using the boxplots marked in orange.

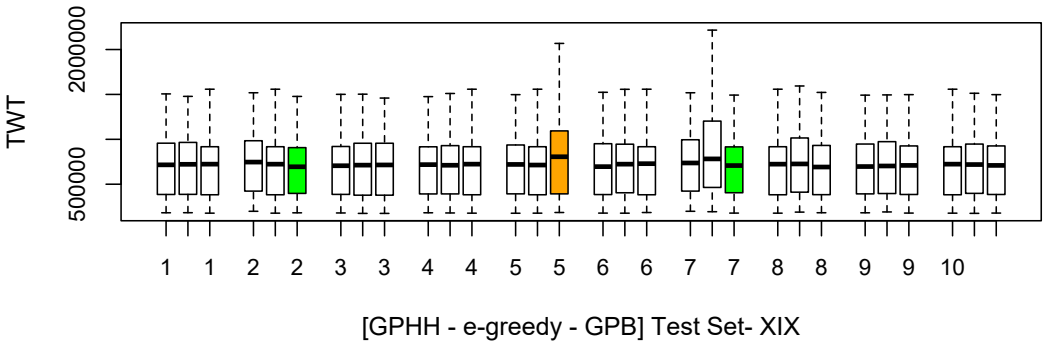


Figure A.3: The boxplot pertaining to the DJSS instance for which GPB performed poorly is highlighted.

For the test set XIX, the performance of GPB over some DJSS instances was poor ($[3\text{-}25\text{-}2]$ as in Table 4.8). The 5th DJSS instance in Fig. A.3 illustrates this observed fact.

Finally, in Fig. A.4, we present more boxplots corresponding to different test sets. We highlight a boxplot in orange if its performance is *very* poor and in green when the performance of the corresponding method

(particularly GPB) is good.

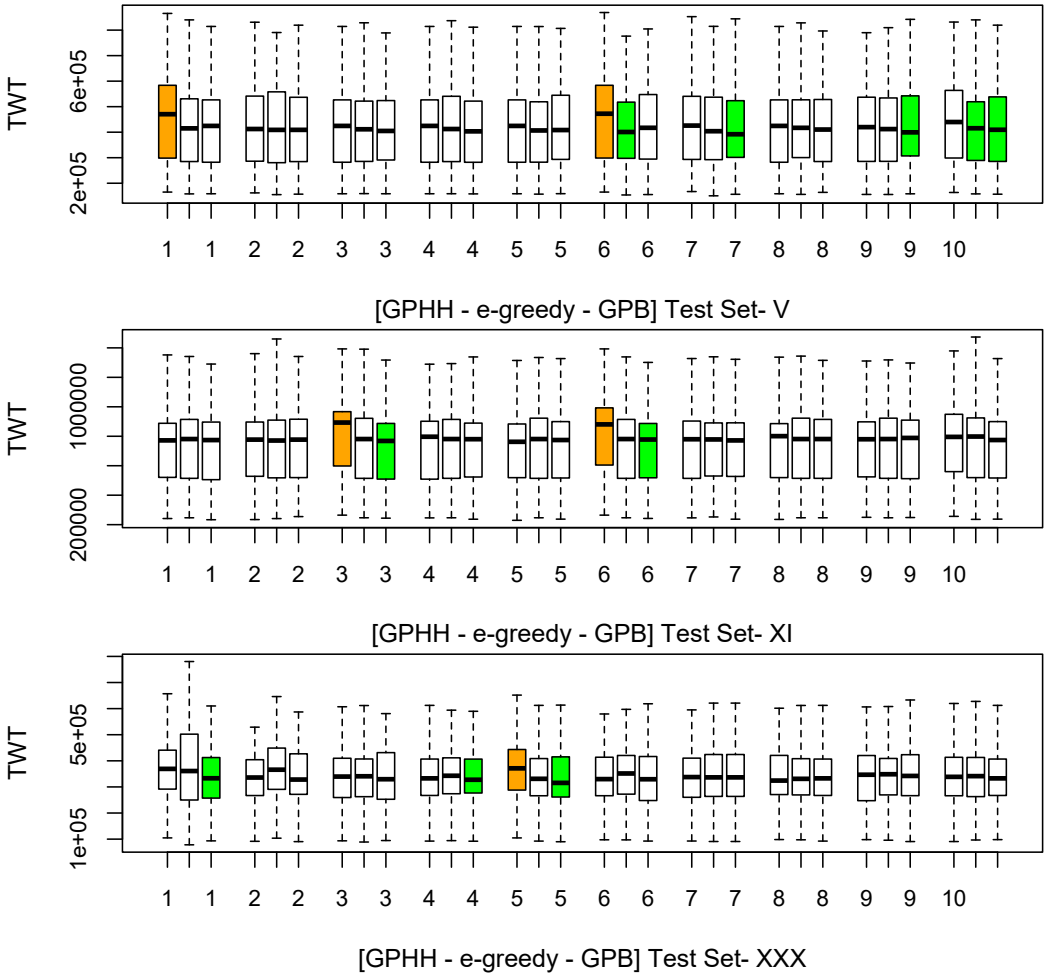


Figure A.4: The boxplots marked in green are better than the rest and the ones marked in orange indicate poor performance.

Appendix B

Supplementary Results for Chapter 5

In Chapter 5, we presented a summary of results which showed the significant improvement observed in multi-objective scheduling performance using the proposed active learning heuristic. Here we present more detailed plots to highlight the observed results. We plot boxplot to visualize the improvement observed with respect to each of the metrics which we considered, namely hypervolume indicator (HVI), spread and inverted generational distance (IGD).

Firstly, we present the box plots for test sets 3-$\mathcal{Y}$ and 4-$\mathcal{Y}$ which are not clustered. In Fig. B.1, we present boxplots showing the performance of NSGAII, island model and our proposed successive reject heuristic on 10 of the test DJSS instances for HVI metric. It can be observed that our proposed approach outperforms on almost all the test instances which is consistent with the results shown in Table 5.12. Similarly, for the same pair of datasets, for the metrics SPREAD and IGD, the boxplots are presented in Figs. B.2 & B.3 respectively. Again this is consistent with our earlier observations which showed improved performance with respect to IGD but not with SPREAD.

For the test sets 3-I, 3-I, 3-III & 3-IV and 4-I, 4-I, 4-III & 4-IV, similar

comparisons using box plots are made. For HVI, Figs. B.4 & B.5 show the boxplots and which show similar characteristics to 3-$\mathcal{Y}$ and 4-$\mathcal{Y}$. similarly, for SPREAD, the Figs. B.6 & B.7 and for IGD, the Figs. B.8 & B.9 supplement the observations made earlier.

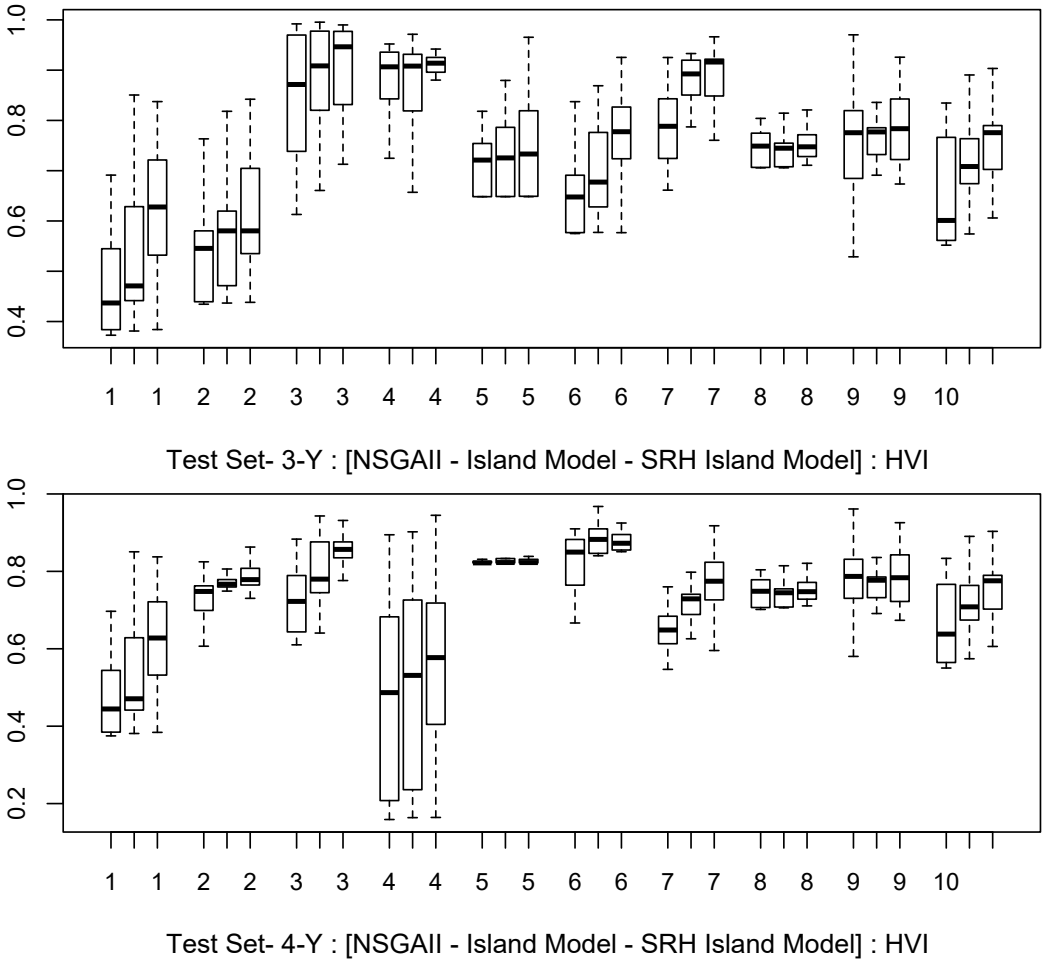


Figure B.1: Test sets: 3-$\mathcal{Y}$ and 4-$\mathcal{Y}$. For HVI, the higher value corresponds to better performance.

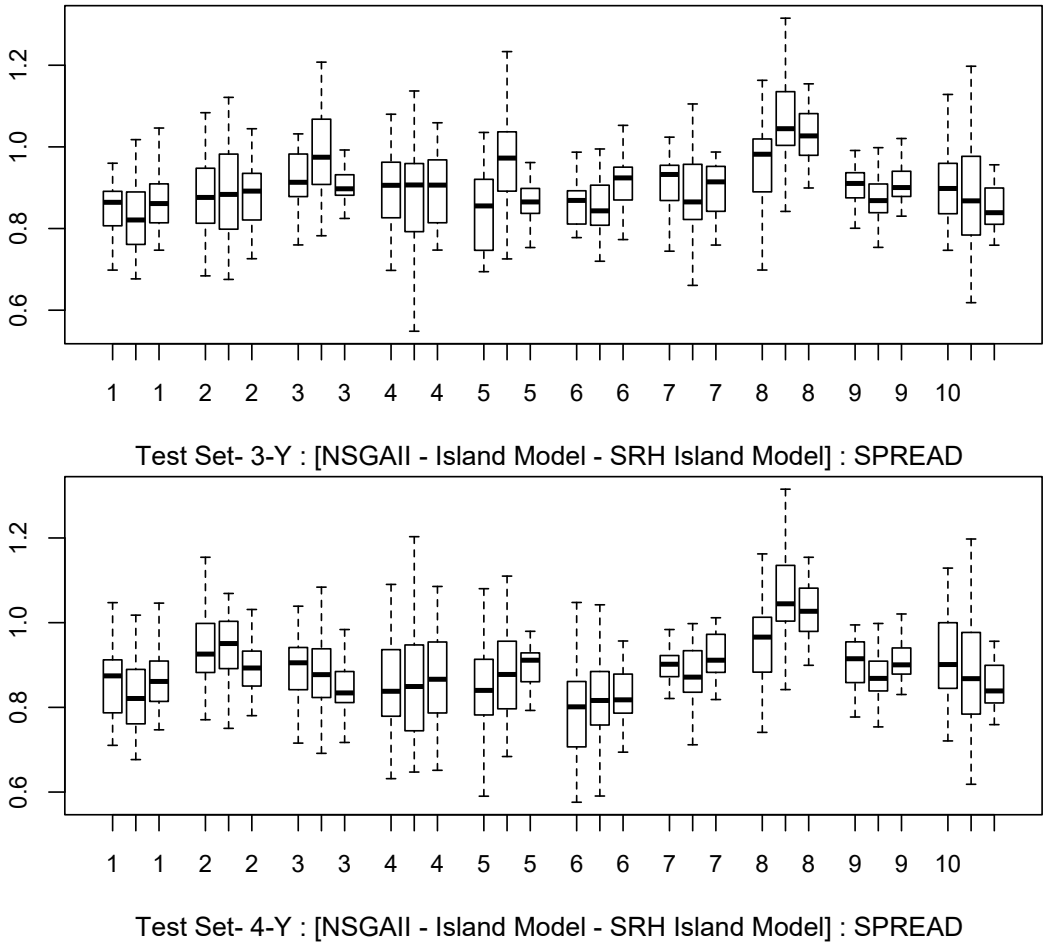


Figure B.2: Test sets: 3-$\mathcal{Y}$ and 4-$\mathcal{Y}$. Lower value of the metric (SPREAD) corresponds to better performance.

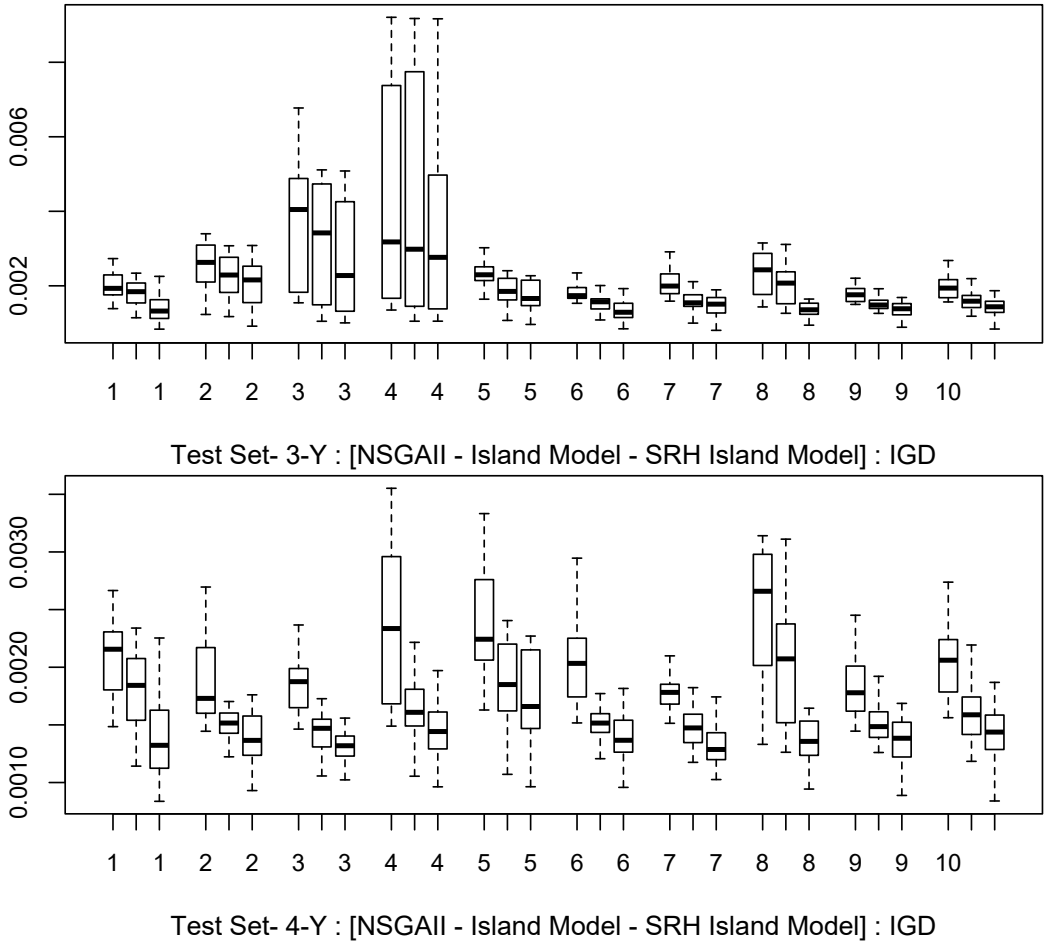


Figure B.3: Test sets: 3-$\mathcal{Y}$ and 4-$\mathcal{Y}$. The order of boxplots is same as mentioned in the caption. Lower value of the metric (IGD) corresponds to better performance.

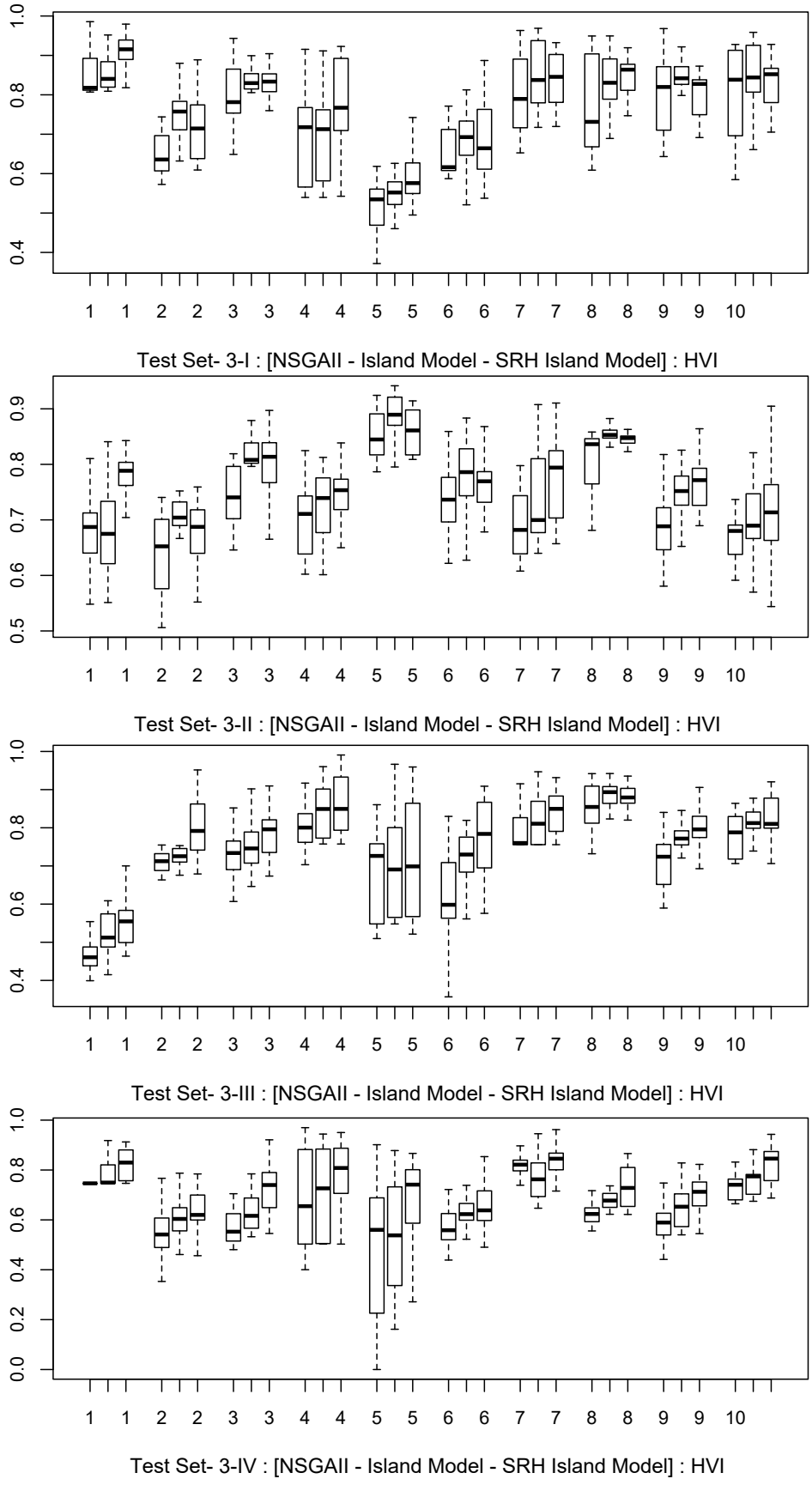


Figure B.4: Test sets: 3-I, 3-I, 3-III and 3-IV. For HVI, the higher value corresponds to better performance.

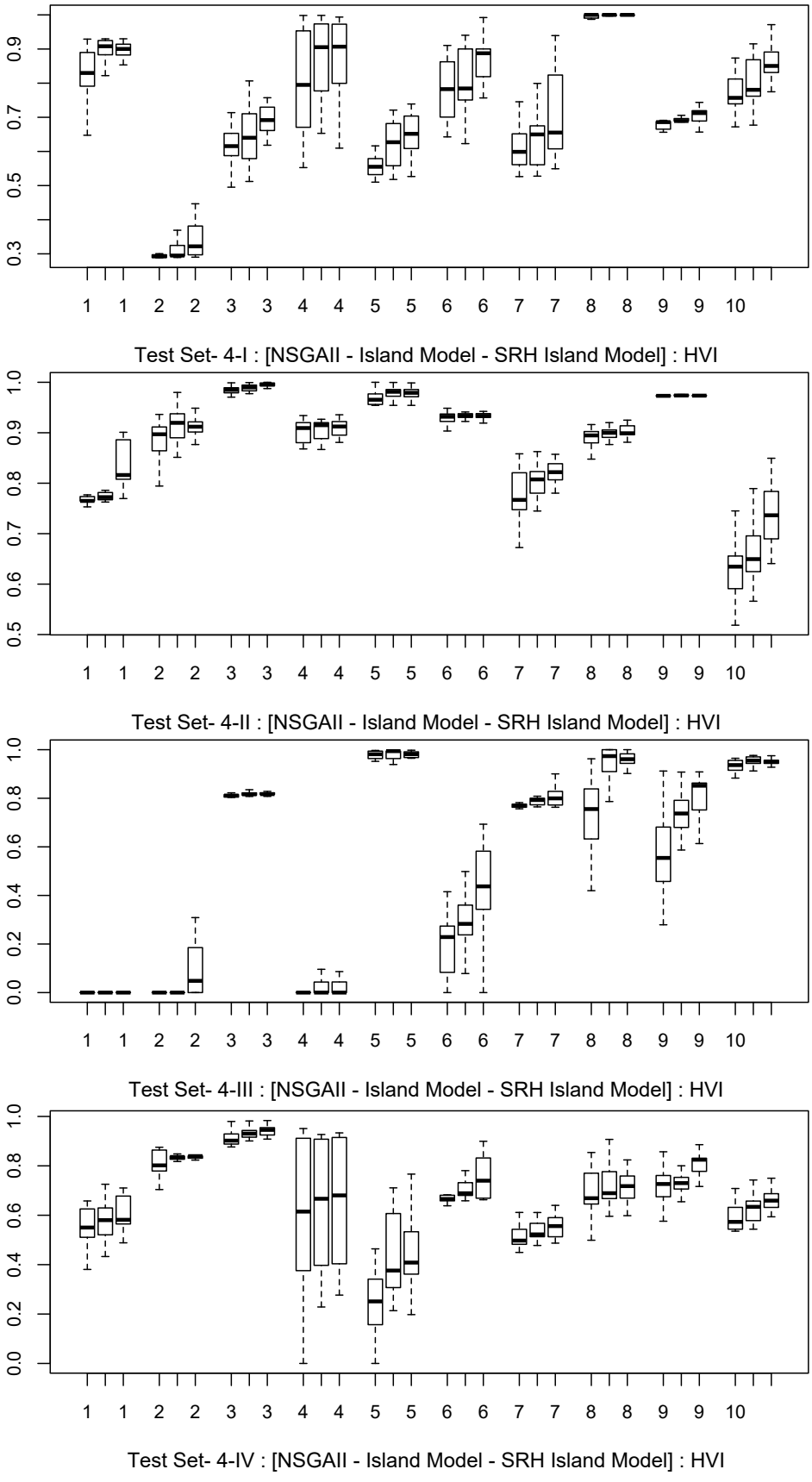


Figure B.5: Test sets: 4-I, 4-I, 4-III and 4-IV. For HVI, the higher value corresponds to better performance.

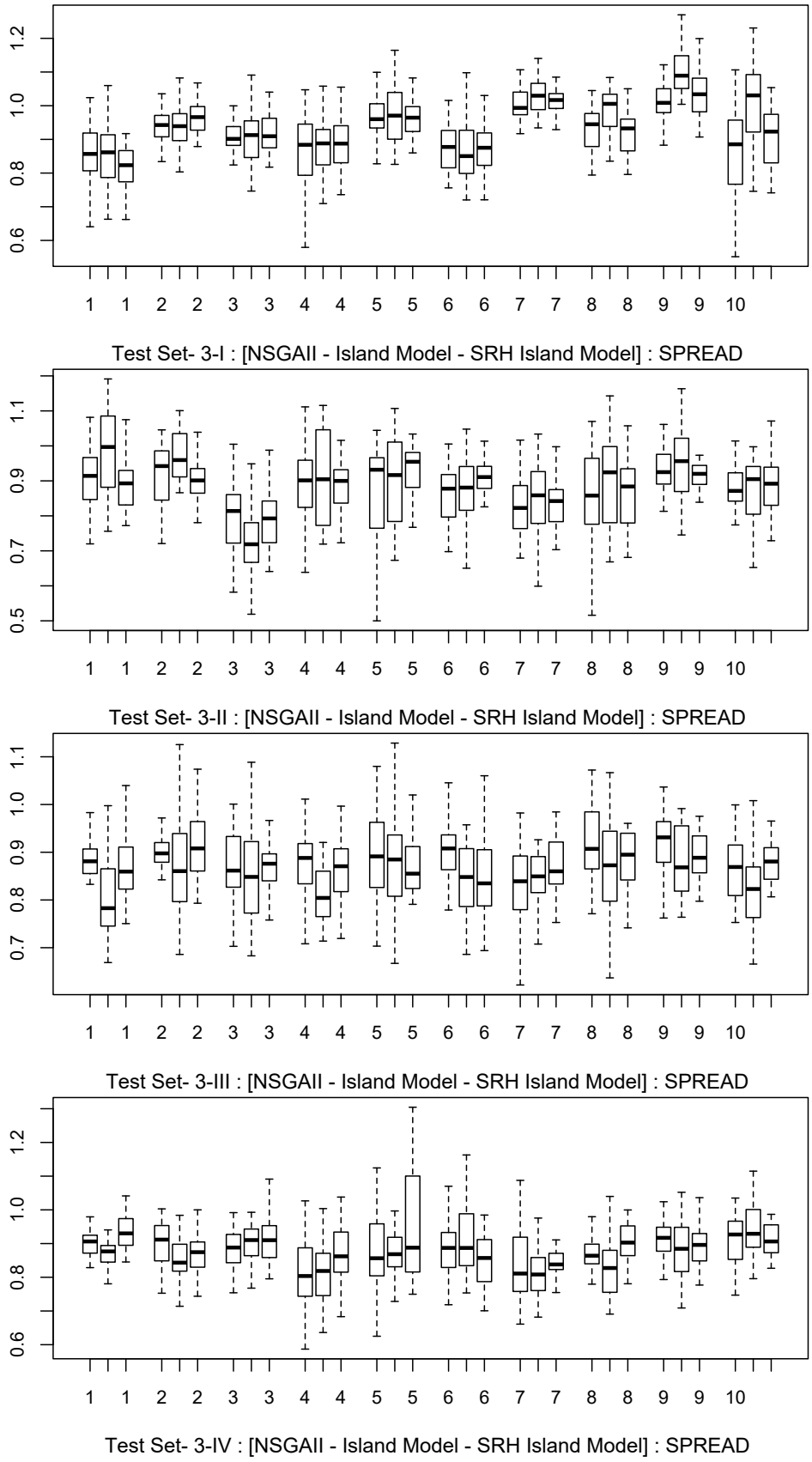


Figure B.6: Test sets: 3-I, 3-I, 3-III and 3-IV. Lower value of the metric (SPREAD) corresponds to better performance.

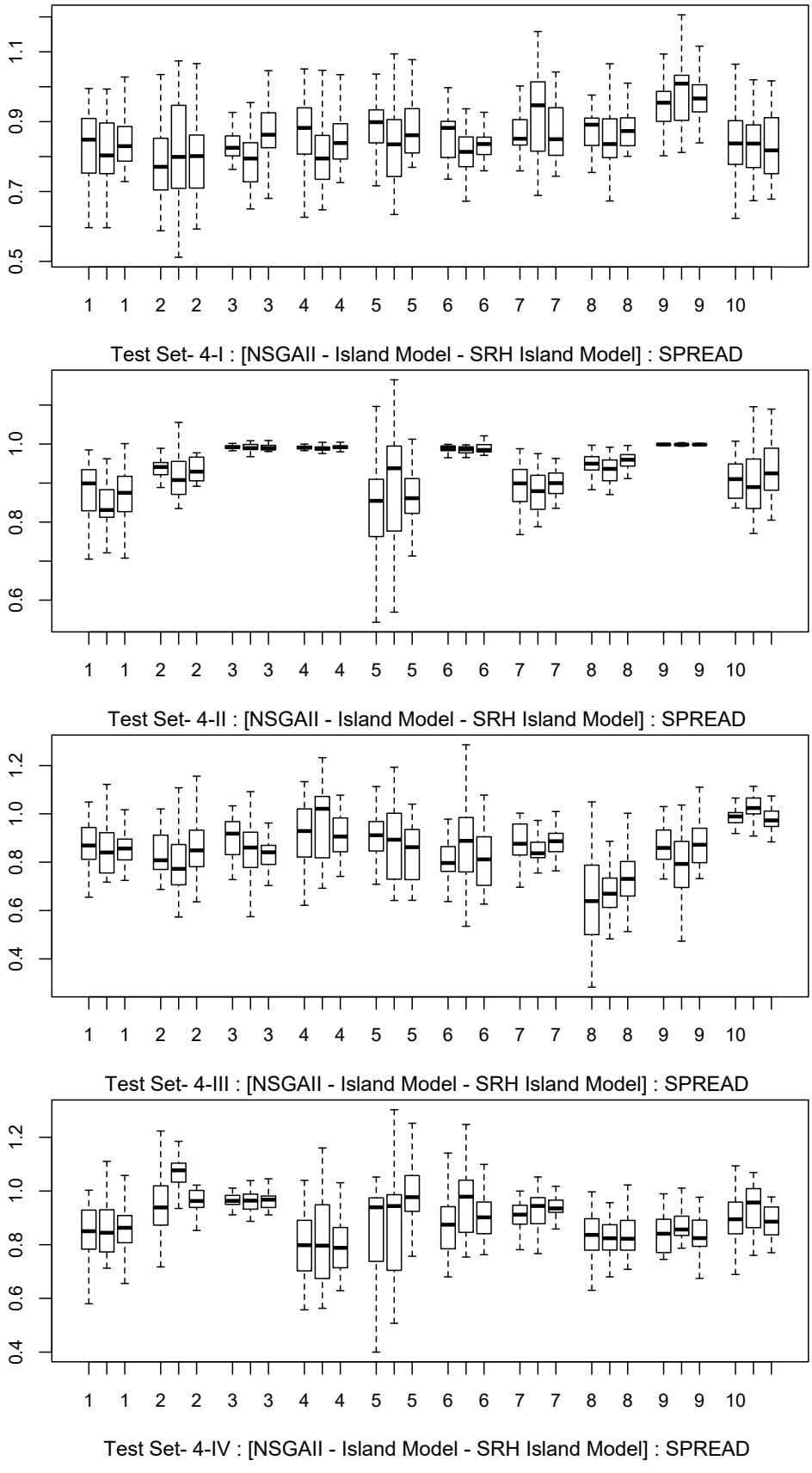


Figure B.7: Test sets: 4-I, 4-I, 4-III and 4-IV. Lower value of the metric (SPREAD) corresponds to better performance.

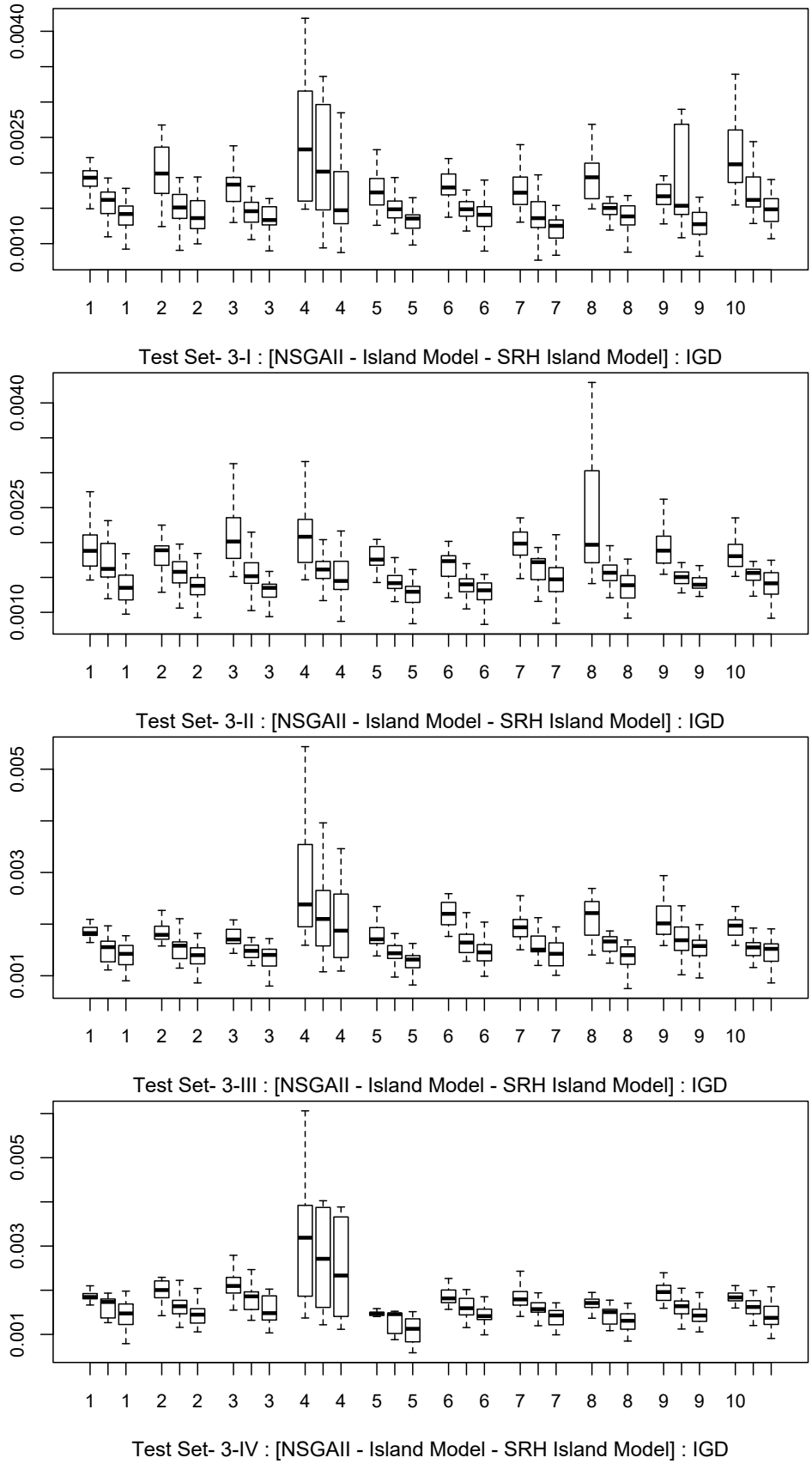


Figure B.8: Test sets: 3-I, 3-I, 3-III and 3-IV. Lower value of the metric (IGD) corresponds to better performance.

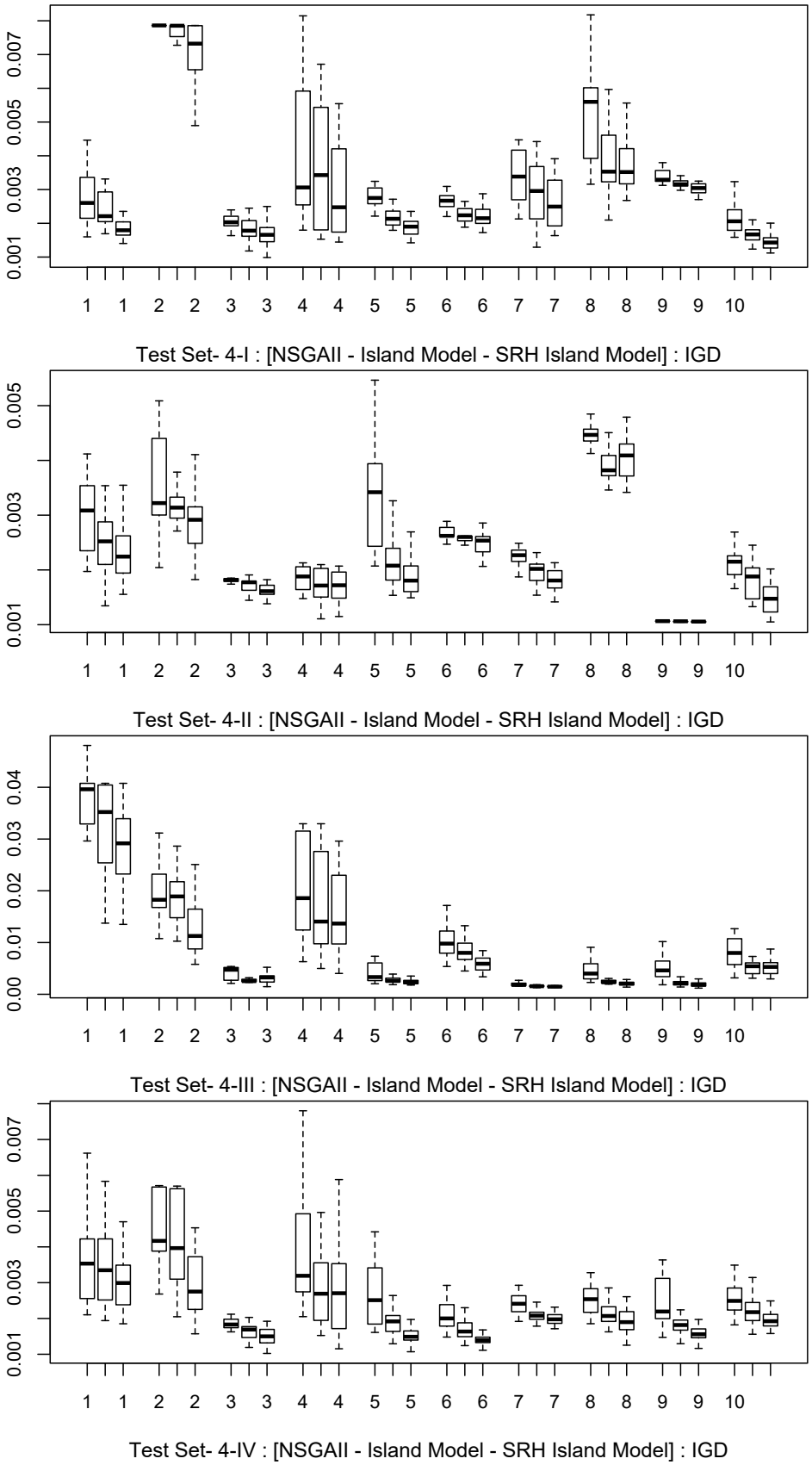


Figure B.9: Test sets: 4-I, 4-I, 4-III and 4-IV. Lower value of the metric (IGD) corresponds to better performance.