Boosting Stochastic Problem Solvers Through Online Self-Analysis of Performance

Vincent A. Cicirello

CMU-RI-TR-03-27

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Robotics.

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

July 21, 2003

This research has been funded in part by the Department of Defense Advanced Research Projects Agency and the U.S. Air Force Rome Research Laboratory under contracts F30602-97-2-0066 and F30602-00-2-0503 and by NASA under contract NCC2-1243. The views and conclusions contained in this document should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NASA, ARPA, the Air Force or the U.S. Government.

Abstract

In many combinatorial domains, simple stochastic algorithms often exhibit superior performance when compared to highly customized approaches. Many of these simple algorithms outperform more sophisticated approaches on difficult benchmark problems; and often lead to better solutions as the algorithms are taken out of the world of benchmarks and into the real-world. Simple stochastic algorithms are often robust, scalable problem solvers.

This thesis explores methods for combining sets of heuristics within a single stochastic search. The ability of stochastic search to amplify heuristics is often a key factor in its success. Heuristics are not, however, infallible and in most domains no single heuristic dominates. It is therefore desirable to gain the collective power of a set of heuristics; and to design a search control framework capable of producing a hybrid algorithm from component heuristics with the ability to customize itself to a given problem instance. A primary goal is to explore what can be learned from quality distributions of iterative stochastic search in combinatorial optimization domains; and to exploit models of quality distributions to enhance the performance of stochastic problem solvers. We hypothesize that models of solution quality can lead to effective search control mechanisms, providing a general framework for combining multiple heuristics into an enhanced decision-making process. These goals lead to the development of a search control framework, called QD-BEACON, that uses online-generated statistical models of search performance to effectively combine search heuristics. A prerequisite goal is to develop a suitable stochastic sampling algorithm for combinatorial search problems. This goal leads to the development of an algorithm called VBSS that makes better use, in general, of the discriminatory power of a given search heuristic as compared to existing sampling approaches.

The search frameworks of this thesis are evaluated on combinatorial optimization problems. Specifically, we show that: 1) VBSS is an effective method for amplifying heuristic performance for the weighted tardiness sequencing problem with sequence-dependent setups; 2) QD-BEACON can enhance the current best known algorithm for weighted tardiness sequencing; and 3) QD-BEACON and VBSS together provide the new best heuristic algorithm for the constrained optimization problem known as RCPSP/max.

Acknowledgements

There are many people whose support and guidance in one form or another have helped to make this dissertation a reality. I would like to acknowledge as many of them as possible here.

First, I would like to thank my advisor, Steve Smith, for his wisdom and guidance throughout my four years here at CMU and for his encouragement to explore research directions that have lead into previously uncharted and interesting new territories.

Next, I would like to express my thanks and gratitude to the members of my thesis committee: Andrew Moore (CMU RI), Norman Sadeh (CMU ISRI), and Carla Gomes (Cornell University). Their advice and guidance throughout my thesis research have been immeasurable.

Thanks to the members of the ICL Laboratory, both past and present, for discussions that have lead my thoughts in directions that otherwise may not have been considered fully and for sharing their comments and suggestions during practices of my defense talk. I would like to specifically acknowledge: Dave Hildum, Larry Kramer, Heng "Harriet" Cao (now at IBM Research), Susan Buchman, Charlie Collins, Gabriella Cortellessa, Dave Crimm, Peter DeKlerk, Li Li, Jean Oh, Nicola Policella, Xiaofang Wang, Chris Young, and Qu "Joe" Zhou (now in Saskatoon, Canada).

Thanks also go to the members of the FIRE (Federation of Intelligent Robotic Explorers) Project. In working with them to develop the scheduling component of the FIRE Project, the algorithms of my thesis have benefitted greatly through insightful discussions. The FIRE Project members include: Jeff Schneider, Reid Simmons, Tony Stentz, Dani Goldberg, Drew Bagnell, Bernardine Dias, Trey Smith, Maayan Roth, Brennan Sellner, and David Apfelbaum.

Thanks to members of my research qualifier committee, Tucker Balch (now at Georgia Tech) and Al Rizzi, for their feedback on earlier research that lead to the foundation of the WHISTLING algorithm.

I would like to acknowledge Chris Atkeson whose insights during the discussion period of my thesis proposal last July lead me to explore concepts from extreme value theory that have greatly added to the theoretical foundation of my thesis research.

Thanks go to the Robotics Institute for giving me the opportunity and support to pursue a degree in Robotics here at CMU. Also, I would like to thank a few RI individuals for their support during my time here at CMU. Matt Mason's periodic graduate student lunches are always a source of intellectually, stimulating discussion that often leads in unpredictable

directions. Thanks to Suzanne Muth for all that she does for the students of the Robotics Institute. Thanks also to the former CIMDS research secretary Carol Boshears for her administrative support during my time in the Robotics Institute and in the ICL Laboratory.

Special thanks go to my family, especially my Mom and Dad, my sisters Debbie and Donna, Mom and Dad Silenzio, and my soon-to-be sister Roseann Silenzio for their support throughout this long process. I especially would like to thank my fiancée Michelle for her love, support, and encouragement, for putting up with my frequent drives between Philadelphia and Pittsburgh, for always being there for me, and simply for being Michelle.

In Memorium

– Pop-Pop (Gene Mingarino) and “Archibald the Mouse.”

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

In many combinatorial optimization domains, simpler stochastic algorithms often exhibit superior performance as compared to more carefully refined and highly customized approaches for the problem at hand. Examples of the success of such stochastic search algorithms abound: Space Telescope Scheduling [25], Project Scheduling [30], Solid Model Similarity Assessment [37, 39, 40, 41], Satisfiability [180, 159, 21], VLSI Channel Routing [20, 21], Graph Coloring [159], Constraint Satisfaction [84], and N-Queens [159] – just to mention a few.

Although there have been numerous successes in this arena, there is little in the way of formal analysis for this success. One potential explanation offered by researchers is the issue of over-fitting algorithms to benchmark instances at design-time. For example, Watson *et al.* studied the effects of problem structure on a variety of algorithms for the flow shop scheduling problem [201] and observed that simple stochastic algorithms such as iterative sampling [130] and heuristic-biased stochastic sampling [25] outperformed the more carefully tuned search procedures considered in the study as real-world problem structure was added to the problems. The explanation given by the authors of the study is that the more carefully refined approaches become over-fitted to the benchmark instances of the problem; while the simpler stochastic algorithms are more robust to variation in problem structure. Similarly, Hooker argues that a prime problem with heuristic search algorithm research is that all too often the emphasis is placed on beating the existing algorithms for the problem of interest [103]. One of the potential unfortunate outcomes of this competitive view of heuristic search algorithm development is also an over-tuning to a set of benchmarks. In

Watson *et al.*'s study, it is shown that the more sophisticated algorithms can become over-fitted to benchmark instances and fail to live up to expectations when faced with problems consisting of real-world structure; but no satisfying explanation is really provided as to why the simpler (and often knowledge-poor) stochastic algorithms should do any better.

Others suggest that the issue is one of scalability (e.g., [159]). For an example consider complete systematic backtrack search procedures. No matter how intelligently these complete backtrack algorithms may search the space of solutions (e.g., through use of heuristics, search-space pruning, and other tricks), they are still faced with an exponential problem. As a consequence of this, these systematic algorithms are particularly prone to this exponential growth when "mistakes" are made early in the search since it might result in exhausting a large portion of unpromising search-space before backtracking to correct a particularly poor choice made early in the search. Many of the simpler stochastic algorithms scale better to problem size due to their lack of completeness. They are often memoryless and care not about systematically exploring the search space. Many iteratively probe from the root of the search space to a leaf making decisions randomly; while others iteratively modify the current search state (e.g., simulated annealing, tabu search) in some stochastic manner. However, why should we expect these algorithms, that consider what amounts to a small amount of the exponentially-sized search-space, to perform as well as they often do in these combinatorial optimization problems?

If you look closely at the behavior of some of these simple stochastic algorithms, then you can often gain some insight into questions such as these. For example, consider stochastic sampling algorithms. Iterative sampling is one such stochastic sampling algorithm that iteratively probes from the root of the search-space while making decisions unbiased at random [130]. Whenever a terminal node is reached, the search begins anew at the root of the search-space completely forgetting about which search-branches were followed on previous iterations. This algorithm clearly cannot be over-fitted to benchmarks since there are no parameters to tune. Also, if it heads down an unpromising branch in the search-space, then it is only going to forget that it followed that path in the first place rather than systematically exploring that region. A less naive approach is that of heuristic-biased stochastic sampling (HBSS) [25]. HBSS also makes each decision randomly, but uses a heuristic to order the choices at each decision point and uses that ordering as a means of biasing the stochastic decisions. HBSS allows the use of heuristic guidance, while using randomization to expand the focused search region. HBSS makes the assumption that it has a good search heuristic, but that it is not infallible and can often make mistakes. Ran-

domization of the heuristic is the method used by HBSS to handle the tradeoff of following the heuristic's advice and the possibility of missing out on better solutions elsewhere in the search-space. Like iterative sampling, HBSS iteratively probes from the root forgetting its previous search trajectories and thus avoiding unnecessary systematic exploration of unpromising search regions.

There are other stochastic elements of search algorithms whose performance have more formal and highly analyzed explanations and motivation. For example, in the field of constraint satisfaction, Gomes *et al.* have set out to explain and motivate the practice of cutting off a systematic (yet randomized) backtrack search procedure at some pre-specified runtime prior to finding a solution and randomly restarting the search along a different search trajectory [95, 92, 96, 33]. This practice is referred to as *rapid randomized restarts*. They illustrate that backtrack search algorithms using a randomized heuristic to select a systematic search order of the branches at decision points often exhibit formally heavy-tailed behavior in their runtime distributions. That is, for a large number of solution trajectories, the backtrack algorithm requires exponentially long runtimes to find a solution (the heavy-tail of the distribution); while there also exists for the same problem a high density of solution trajectories that lead almost directly to a solution. This provides rationale for the rapid randomized restart technique. By cutting off the search at some specified time-limit and restarting the algorithm along a new trajectory, you can abandon the runs that belong to the heavy-tail in favor of more promising runs that may lead to a solution in less time. The runs in the heavy-tail of the runtime distribution are runs that likely have become stuck exhausting unpromising portions of the search-space. Rapid randomized restarts gives the search a way out of such a bad search-space region without requiring it to exhaust it first. In general, the heavy-tailed nature of the runtime distributions provide the rationale for prescribing fixed restart cutoffs and under certain assumptions can also provide rationale for schedules of cutoffs. Chen, Gomes, and Selman [33] have illustrated formal search-space models that do exhibit the heavy-tailed phenomena; and have also illustrated search-space models for which heavy-tailed behavior is not exhibited by backtrack search (e.g., balanced search trees) and which consequently do not experience a runtime boost through rapid randomized restarts.

This thesis explores methods for taking a set of heuristics and combining them within a single stochastic search. The success of stochastic search algorithms is often due to their ability to effectively amplify the performance of search heuristics. Heuristics, however, are not infallible and in most domains there does not exist a single dominating heuristic.

Hence, it is desirable to be able to gain the collective power of a set of heuristics. The idea is rather than carefully customizing an algorithm, to design a search control framework capable of taking several simple search heuristics and turning them into a hybrid algorithm with the ability of customizing itself on a per problem instance basis. A primary goal of this thesis is to explore what can be learned by looking at the quality distributions of iterative stochastic search algorithms. Specifically, I focus on combinatorial optimization domains; and by "quality" refer to the objective values of the solutions produced by these algorithms across multiple iterations. A further goal of this thesis is to explore ways of exploiting models of such quality distributions to enhance the performance of stochastic problem solvers. Gomes *et al.*'s study of runtime distributions has lead to much success in constraint satisfaction domains. A hypothesis of this thesis is that models of the distributions of the solution qualities given by iterative stochastic search algorithms (in optimization domains) can lead to the development of effective search control mechanisms that can enhance the problem solving ability of such algorithms and that can provide a general framework for combining multiple search heuristics into an enhanced decision-making process. These goals lead to the development of a search control framework that uses online-generated statistical models of search performance to effectively combine multiple search heuristics and/or search algorithms. A prerequisite goal of this thesis is therefore to develop a suitable stochastic sampling algorithm for combinatorial search problems and to explore the effectiveness of stochastic (heuristic-guided) sampling algorithms in domains of combinatorial optimization. This prerequisite goal leads to the development of an algorithm called value-biased stochastic sampling that makes better use, in general, of the discriminatory power of a given search heuristic as compared to existing rank-biased approaches.

1.2 Overview

This thesis analyzes and offers a characterization of the inner workings of simple iterative stochastic search techniques in combinatorial optimization domains. In a sense, this research proposes to do for iterative stochastic optimization algorithms what Gomes *et al.* have been doing for backtrack constraint satisfaction search algorithms. That is, I offer a formal (descriptive) model of search performance, specifically focusing on stochastic sampling algorithms. I further go a step beyond what has been done for backtrack search and use the resulting formal model of the operation of the search to provide the optimizing search with online (prescriptive) guidance. This thesis is threefold:

1. Advantage / leverage of a stochastic sampling algorithm.

2. Descriptive problem solving analysis / characterization.

3. Online prescriptive guidance from the descriptive analysis.

1.2.1 Advantage / Leverage of a Stochastic Sampling Algorithm

In this research, I specifically study the performance of algorithms that are based on the iterative sampling algorithm. In iterative sampling, decisions are made unbiased at random from the root of the search-space to a leaf. This is then iterated. In many cases, there may be heuristics available for the problem at hand that can lend some guidance to the problem. Iterative sampling as it is defined does not follow any such guidance, but there are approaches based on iterative sampling that do. One crucial aspect of any such search procedure is the approach taken to randomizing the base search heuristic. Conceptually, the goal is to perturb the choice order prescribed by the heuristic at a given decision point in a way that preserves much of the heuristic's original bias. A simple, non-interfering strategy is to simply break ties randomly in those decision contexts where two or more top choices are ranked equivalently (i.e., making random decisions only when the heuristic fails to make a selection). A somewhat more aggressive approach, called *heuristic equivalency* [92], chooses randomly among all choices having a heuristic value within $H\%$ of the highest. A more general framework called *Heuristic-Biased Stochastic Sampling* (HBSS) utilizes a specified bias function as a basis for deviating from the choice order given by the base heuristic [25]. Through the use of different bias functions, a wide range of randomization strategies can be realized.

I present an alternative approach to randomizing search heuristics within an iterative sampling framework. The approach taken in this thesis has much in common with HBSS, but with one important difference. Instead of operating with respect to the rank ordering of choices that is implied by application of the heuristic, our approach biases selection on the actual values assigned by the heuristic to each possible choice. Like Oddi and Smith [153], we assume that the heuristic can be more or less discriminating in different decision contexts and argue that the heuristic's degree of preference for one choice over another should impact the random selection process. We designate the algorithm *Value-Biased Stochastic Sampling* (VBSS). We further define a specific mode of computation for the stochastic decisions in VBSS that is inspired by a stochastic model of how wasp colonies self-organize (or order themselves) into dominance hierarchies. This technique is lifted from the biological

context and reformulated as a general framework for randomizing heuristics in a stochastic search context. We designate this approach *Wasp beHavior Inspired STochastic sampLING* (WHISTLING). VBSS and WHISTLING are presented in greater detail in Chapter 4.

1.2.2 Descriptive Problem Solving Analysis / Characterization

Bresina *et al.* introduced the idea of a *quality density function* (QDF) [24, 23]. The QDF is the distribution of the quality of solutions obtained from sampling uniformly from a solution space. In a sense, it characterizes the space of solutions according to their qualities. If the particular problem domain satisfies certain conditions then the QDF can be generated by performing some large number of iterations of the iterative sampling algorithm (i.e., making each search decision at random according to a uniform distribution over the possible choices). Bresina uses the QDF to compare the performance of various algorithms on his particular problem domain. First, he computes an estimate of the QDF for a problem instance using iterative sampling. Then, he solves the instance with each of the algorithms he wishes to benchmark. He then scores each algorithm according to the number of standard deviations away from the mean of the QDF is the algorithm's quality. This allows for a more statistically meaningful algorithmic comparison across problem instances than simply saying algorithm X produced a solution whose quality was Y% better than that of algorithm Z.

One can also view quality distributions from the perspective of a particular stochastic sampling algorithm, giving an idea of how the algorithm can be expected to perform on various problem instances. Thus, taking as inspiration the QDF of Bresina, I define an *algorithm quality density function* (AQDF) as the QDF obtained from sampling from a solution space via a particular stochastic search algorithm rather than uniform samples. For some problem instances, a particular iterative stochastic search algorithm may lead to a high density of "good" solutions on successive iterations of the algorithm; while on other instances, it may not. Similarly for some algorithms, a particular class of problem instances may be solved very well; while other algorithms may have a difficult time finding a "good" solution to these instances. The AQDF offers a methodology for modeling the performance of a stochastic search algorithm for a specific problem instance. The AQDF is presented in greater detail in Chapter 6.

1.2.3 Online Prescriptive Guidance from the Descriptive Analysis

Another crucial aspect of an iterative sampling approach using heuristic information (e.g., HBSS, VBSS, WHISTLING) is the choice of base heuristic itself. Often, for any given problem domain there may exist a number of carefully designed heuristics. Each of these heuristics may have particular classes of problem instances for which they are more highly regarded when used deterministically without any search. In most interesting problem domains, it is rare that any one heuristic will dominate all others in general (e.g., dispatch scheduling policies [145]). For example, for a particular scheduling objective, one heuristic may perform better in general for lightly loaded problem instances while another may perform better in general for more heavily loaded instances. When employing an iterative sampling search procedure, it may not be very clear which of these heuristics should lead to better results in your domain or the problem characteristics that would lead to an informed algorithmic choice may not be available until execution time. Also, the more state-of-the-art heuristics, particularly dispatch scheduling policies, very often are parameterized and these parameters are often over-fitted to a set of benchmark problems a priori. In practice, some other set of parameters for the heuristic may be more appropriate and may lead to better solutions. This heuristic with different sets of parameters can be seen as different heuristics for the same problem – each of which may be more appropriate for different problem characteristics such as the factory load, tightness of duedates, etc.

I present an approach to stochastic sampling that will allow the use of multiple base search heuristics. The approach of this thesis adaptively chooses from among the base search heuristics for further iterations of the algorithm. This adaptive decision is based on estimates of the probability of finding better solutions than the best found so far for further iterative samples using each base heuristic. Given a relatively few samples of the iterative algorithm, the approach estimates the AQDF. Then, using these estimates of the AQDFs for each of the base heuristics which adapt as iterations of the algorithm are performed, we can decide which heuristic is likely to be most fruitful for further iterative samples. Thus, we can take the descriptive analysis of the algorithm's operation and turn it into prescriptive guidance for the remainder of the search. The AQDFs can also be used to consider the computation time / solution quality tradeoff of performing more iterations of the algorithm with any heuristic. Perhaps, given the AQDFs, it may be deemed too unlikely for the search to find a significantly better solution within a reasonable number of additional iterations. For example, perhaps you have X problems to solve and limited time to solve them. You can spend some number of iterations on each problem and use the

AQDFs to decide which problem appears to be leading in the most fruitful direction. We designate this approach *Quality Distribution Based sEArch CONtrol* (QD-BEACON). The QD-BEACON acronym can also serve as an analogy: just as boats use lighthouse beacons for guidance and aircraft use radar beacons for guidance, stochastic search methods such as VBSS or WHISTLING can use QD-BEACON for guidance. QD-BEACON is presented in greater detail in Chapter 7.

1.2.4 Evaluation

The framework of this thesis is validated and evaluated on various combinatorial optimization problems. Two specific problems upon which the experimentation focuses are weighted tardiness scheduling with sequence-dependent setups (see Chapter 5, also see Chapter 8 for the problem without setups) and resource constrained project scheduling with time windows (see Chapter 9). The former is a difficult, yet real-world relevant scheduling problem with insufficient coverage in the literature; thus a competitive approach to this problem is a significant contribution in its own right. The latter adds an interesting and challenging dimension to the study in that it involves solving an NP-hard constraint satisfaction problem within the confines of an optimization process.

To evaluate the performance of the framework, the optimal or best-known solutions of benchmark problems are used where available. Unfortunately, for many of the problems of interest in this study, optimal solutions are too costly to compute. When such a comparison is not realistically possible, the performance of the framework is compared to that of other related algorithms as well as to current best known algorithms.

1.3 Organization

The remainder of this thesis is organized as follows:

- **Chapter 2**. This Chapter overviews related work in the area of search algorithms. It specifically focuses on stochastic search algorithms and heuristic-guided search. Topics discussed in detail include: discrepancy search, stochastic sampling, stochastic local search, problem-space motivated search algorithm design, and methods of combining multiple search algorithms such as through the concept of an algorithm portfolio.

- **Chapter 3**. This Chapter overviews related work in the areas of metareasoning and metalevel search control. Topics discussed include: metalevel control parameter optimization, learning models of search control, and the body of work that has been done on anytime computation.

- **Chapter 4**. This Chapter presents the VBSS framework and its algorithmic details, including a presentation of a specific mode of computation for the stochastic decisions of the VBSS framework called WHISTLING.

- **Chapter 5**. This Chapter conducts an analysis of the performance of VBSS on a difficult combinatorial optimization problem known as weighted tardiness scheduling with sequence-dependent setups. Specifically, the potential benefits of the value-biased approach over the rank-biased alternative are explored. A second topic of the discussion is value-biasing the starting solution configurations of a local search approach to the problem. A comparison of VBSS with systematic heuristic guided search algorithms such as discrepancy search is also presented.

- **Chapter 6**. This Chapter defines the algorithm quality density function that is the key underlying characterization tool used in later chapters to guide search control decisions.

- **Chapter 7**. This Chapter details the Quality Distribution Based sEArch CONtrol framework. QD-BEACON is a framework that provides stochastic search algorithms with the functionality necessary for modeling quality distributions associated with searching with different heuristics (models of AQDFs) and with the functionality necessary for exploiting these models to enhance search performance. Alternative modeling techniques for the AQDF are presented. Also defined in this Chapter is the exploration/exploitation strategy used by the framework. In the presentation of this exploration strategy, a new variation of the multiarmed bandit problem, that we call the *Max k-Armed Bandit Problem* is presented and analyzed.

- **Chapter 8**. This Chapter revisits the weighted tardiness scheduling problem discussed earlier in Chapter 5. However, in this Chapter, we consider the somewhat easier version of the problem without sequence-dependent setups. Though easier, it is still a difficult NP-hard optimization problem. Due to the multiple heuristics available for this problem, it is considered a good candidate problem for which to test the efficacy of the QD-BEACON framework. QD-BEACON is used to enhance

the performance of state-of-the-art solution procedures for the problem. In particular, a QD-BEACON enhanced version of the previous best known algorithm for the problem is now the new best known algorithm for the problem.

- **Chapter 9**. This Chapter explores the use of VBSS and QD-BEACON in the domain of resource constrained project scheduling with time windows. This problem is one for which finding feasible solutions alone is NP-hard. Thus solution procedures are faced not only with the difficult optimization problem, but also with a difficult constraint satisfaction problem as well. A state-of-the-art priority-rule based search algorithm for the problem is enhanced using VBSS and QD-BEACON. The resulting algorithm is superior to the current best performing heuristic algorithm (ISES) for the problem and competitive with a number of truncated branch-and-bound approaches to the problem. The QD-BEACON/VBSS Iterative Priority-Rule Method is also able to improve upon the best known solutions to a few of the problem instances in the benchmark set.

- **Chapter 10**. In this final Chapter, contributions of this thesis to the larger research community are discussed. Additionally, potential directions for future research building from the frameworks and results of this thesis are considered.

Chapter 2

Related Work: Search Algorithms

2.1 Overview

In this Chapter, I discuss related work on search algorithms. Section 2.2 describes systematic search procedures that have developed from work on constraint satisfaction. Section 2.3 discusses stochastic sampling algorithms. Section 2.4 discusses stochastic local search methods. Section 2.5 discusses recent studies of problem space characteristics as motivation for more effective search algorithm design. One particularly successful example that is discussed in detail is the heavy-tailed nature of the runtime distributions of backtracking search in constraint satisfaction as motivation for rapid randomized restarts. In Section 2.6, frameworks for using collections of search algorithms in parallel are described. A summary concludes the Chapter in Section 2.7.

2.2 Systematic Search Procedures

2.2.1 Discrepancy Search

Harvey and Ginsberg consider the idea that for some constraint satisfaction problems, successor ordering heuristics in a tree search could lead directly to a solution; and that in the cases where a heuristic fails to lead directly to a solution, it may have succeeded had it not made a few mistakes along its path through the search tree. *Limited Discrepancy Search* (LDS) is designed with this rationale in mind [100]. LDS begins by following the successor ordering heuristic through the search tree to a leaf. If the leaf is a solution, the search ends. Otherwise, it systematically considers all paths through the search tree with at most

1 "discrepancy" with the ordering heuristic (i.e., at most one decision is made contrary to the choice of the heuristic). If it fails to find a solution, then it considers all paths with at most 2 discrepancies with the heuristic, and then at most three, and so forth until either a solution is found or the search space is exhausted. *Improved Limited Discrepancy Search* (ILDS) eliminates much of the redundancy of the LDS algorithm by ensuring that each leaf node is visited at most once [126].

Walsh has similar motivation in the design of his *Depth-bounded Discrepancy Search* (DDS) [199]. Walsh, however, acknowledges the idea that search heuristics are often less-informed at the top of the search tree and often very-informed near the leafs of the tree. To deal with this, he combines aspects of LDS with aspects of iterative deepening search [125]. On iteration 0, DDS follows the heuristic's advice to a leaf. On iteration $i+1$, DDS considers all discrepancies at depth i or less systematically in order of increasing discrepancy.

Although LDS and DDS are motivated by some of the same rationale as the work of this thesis, their goals are somewhat different and are not well-suited to the types of problems for which I am concerned. LDS and DDS are concerned with finding a *feasible* solution as opposed to finding an *optimal* or *near-optimal* solution. Consider a problem where every leaf node is a solution but with drastically differing objective values. LDS or DDS would not have any idea of when to stop searching in this instance. Also, although they have the advantage of being complete search procedures (i.e., guaranteed to find a solution if one exists), this advantage can be a disadvantage when scalability to very large problem instances is considered due to its worst-case exponential complexity. Often, non-systematic search procedures are the ones that scale best to larger combinatorial problems [130, 84, 159]. Later in this thesis, results are presented comparing LDS and DDS to the stochastic sampling algorithm VBSS which illustrates the limitations of these systematic search procedures in some combinatorial optimization domains.

2.2.2 Incremental Search

Pemberton and Korf consider variations of well-known systematic search algorithms for use in real-rime decision-making [155]. Specifically, they present incremental versions of depth-first branch-and-bound[1] and best-first search. They define the following algorithms:

- *Iterative-deepening branch-and-bound* iteratively performs depth-first branch-and-bound searches to progressively greater depths until time runs out.

[1] See the Operations Research literature for a detailed description of branch-and-bound search.

- *Incremental branch-and-bound* precalculates the maximum depth that a depth-first branch-and-bound is guaranteed to complete within the available time for the branching factor of the search-space under worst-case conditions. It then executes a depth-first branch-and-bound to this precomputed maximum depth.

- *Bounded memory best-first search* bounds the size of the open-list of search nodes. The search proceeds until either time runs out or the open-list exceeds its maximum allowed size.

- *Incremental recursive best-first search* is an incremental version of an algorithm called *recursive best-first search (RBFS)*. RBFS maintains the current search path and the heuristic values for all of the siblings of nodes along the current path [127]. Incremental RBFS performs RBFS one node expansion or backtracking step at a time until time runs out.

2.3 Stochastic Sampling Algorithms

A stochastic sampling algorithm constructs a solution from scratch on each iteration, probing from the root of the search tree iteratively. Each of these decisions is made stochastically. The details of this stochastic decision process are dependent upon the particular algorithm of interest. Figure 2.1 shows, as an example, the search space for a stochastic sampler and a simple four city TSP. The probe that is indicated by the arrows in the Figure represents the tour of the cities, $A \rightarrow C \rightarrow B \rightarrow D$. Similarly, each of the possible probes from the root of this search tree to a terminal node represents a tour of the cities.[2]

2.3.1 Iterative Sampling

Iterative Sampling [130] is a simple algorithm that begins at the root of the search tree and chooses a branch to follow from the root at random. From this successor node in the search space, it again chooses randomly a branch to follow, and so on and so forth until it either finds a solution or hits a dead-end. If the latter, then it begins again at the top of the search tree and iterates the process. If the former, then if we are simply looking for *some* solution then we are done. Otherwise, if we are looking for the best solution we can find, then the process iterates some number of times until we are satisfied with the quality of the best

[2]For this representation of a TSP search tree, there may be multiple probes from the root that correspond to the same tour of the cities. For example, the path $A \rightarrow C \rightarrow B \rightarrow D$ is equivalent to $A \rightarrow D \rightarrow B \rightarrow C$.

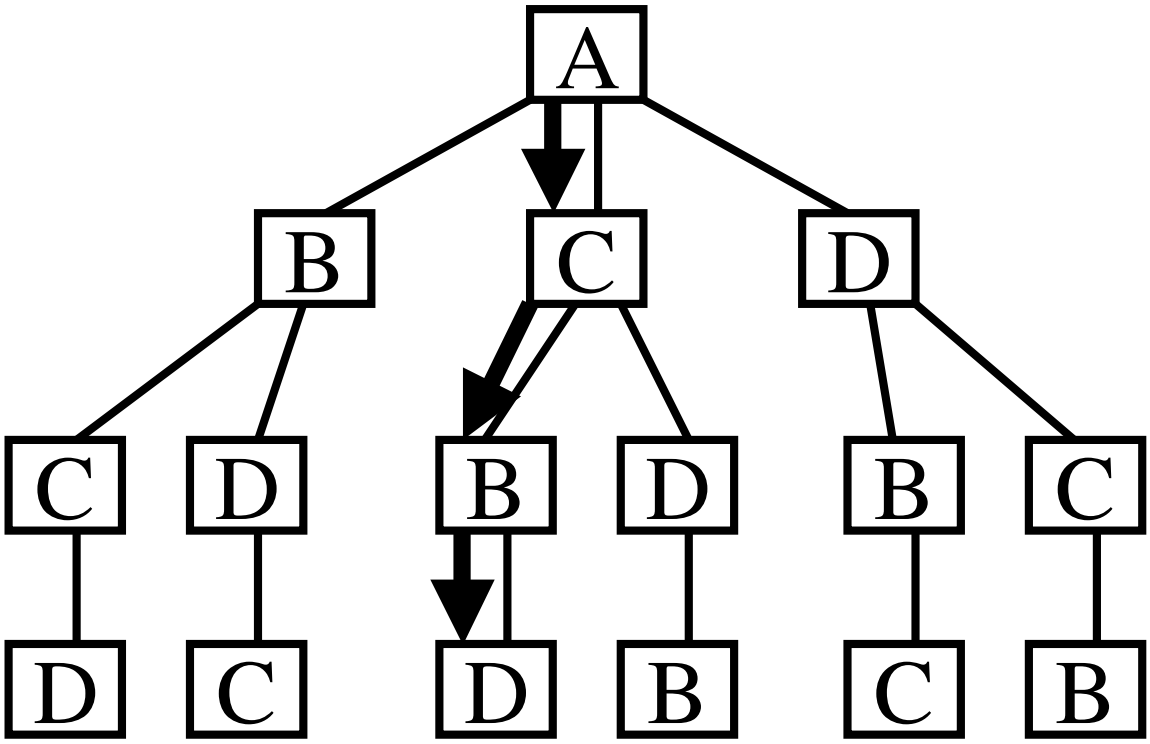


Figure 2.1: Search space for a stochastic sampler and a simple four city TSP. The probe indicated by arrows represents the tour of the cities, $A \rightarrow C \rightarrow B \rightarrow D$.

solution found. It is a rather simple, easy to implement algorithm; but it is naive and does not consider any search/state information, nor does it consider any existing heuristics for the problem. Its ability to find solutions relies entirely on the assumption that there may exist many solutions in the search-space. Furthermore, its ability to find "good" solutions in an optimization context relies entirely on the assumption that there exists a high density of such "good" solutions. In most domains of practical interest, these assumptions tend to be overly optimistic, although some promising results have been obtained for a flow-shop scheduling problem [201].[3] Algorithm 2.1 shows the iterative sampling algorithm for a generic search-tree.

2.3.2 Heuristic-Biased Stochastic Sampling

As indicated in Chapter 1, Bresina's HBSS [25] framework provides a general basis for amplifying heuristic performance through randomization. HBSS operates within a global

[3]Iterative sampling outperformed a few highly-customized problem-specific algorithms as real-world problem structure was added to the problem. The problem-specific heuristic algorithms had been over-fitted to the benchmark set.

Algorithm 2.1: Iterative Sampling (IS)
Input: Number of iterations I; an "objective" function; and a search-tree T.
Output: A solution S.
IS(I, objective, T)
(1) bestsofar $\leftarrow$ nil
(2) **repeat** I times
(3) $S \leftarrow$ root search-node of T
(4) **while** S is a decision node of T
(5) **select** uniformly at random choice C from S
(6) $S \leftarrow$ Successor(S,C)
(7) **if** objective(S) is superior to objective(bestsofar)
(8) bestsofar $\leftarrow S$
(9) **return** bestsofar

search paradigm, where partial solutions are extended by adding one new decision at each step of the search. Like iterative sampling, a random choice process is invoked to make each decision; but unlike iterative sampling, this process is biased according to a pre-specified heuristic for the problem at hand. Specifically, the heuristic is used to first prioritize the alternatives that remain feasible at a given decision point, and then a bias function is superimposed over this ranking to stochastically select from this ranked set. Once a complete solution is generated, it is evaluated according to the global optimization criteria. The search process is then repeated some number of times and the best solution generated is taken as the final result.

The specific problem considered by Bresina was telescope observation scheduling. Given a set of potential observation tasks and an objective criterion (e.g., maximize viewing time), the problem is to produce a schedule (i.e., a sequence of tasks) for execution during the next period. Formulated within HBSS, generation of a schedule proceeds in a forward dispatching manner, by repeatedly ranking the subset of tasks that remain "unscheduled", and then choosing the next task to append to the current (partial) schedule.[4] This process iterates until either all potential tasks have been scheduled or the time frame has been exhausted.[5] The resulting schedule is then evaluated globally and the search process is restarted. The HBSS algorithm is illustrated in a general search context in Algorithm 2.2.

[4]Re-ranking is necessary at each step because the state (e.g., the position of the telescope and current time), and thus the heuristic ordering, change each time a new task is added to the tentative schedule. Also contributing to the context-dependent nature of the ranking is the fact that some observation tasks are only schedulable within specific time windows and thus are not always feasible choices.

[5]In most cases, it is not possible to schedule all desired observations in Bresina's domain within the allotted time window.

Algorithm 2.2: Heuristic-Biased Stochastic Sampling (HBSS)
Input: Number of iterations I; a "heuristic" function; a "bias" function; an "objective" function; and a search-tree T.
Output: A solution S.

```
HBSS(I, heuristic, bias, objective, T)
(1)   bestsofar ← solution S obtained if "heuristic" is followed from T
(2)   repeat I times
(3)     S ← root search-node of T
(4)     while S is a decision node of T
(5)       foreach choice C from S
(6)         score[C] ← heuristic(C, S)
(7)       sort all choices C according to score[C]
(8)       totalweight ← 0
(9)       foreach choice C from S
(10)        rank[C] ← sort position of C
(11)        weight[C] ← bias(rank[C])
(12)        totalweight ← totalweight + weight[C]
(13)      foreach choice C from S
(14)        prob[C] ← weight[C] / totalweight
(15)      select randomly the choice C biased according to prob[C]
(16)        S ← Successor(S, C)
(17)    if objective(S) is superior to objective(bestsofar)
(18)      bestsofar ← S
(19)  return bestsofar
```

The ability to use different bias functions within HBSS provides a means of placing more or less emphasis on following the advice of the base heuristic. A number of polynomial bias functions of the form r^{-n} and an exponential bias function of the form e^{-r}, are proposed and explored by Bresina, where r is the rank of the choice in question [25]. As pointed out by Bresina, the choice of bias function can and should be made based on overall confidence in the base heuristic. If the heuristic is deemed strong, then it makes sense to follow it more often; if the heuristic is weak, then a more disruptive bias is called for. The potential problem is that heuristics are typically more or less informed in different decision contexts; and it is not possible to calibrate the degree of randomness allowed according to this dynamic aspect of problem solving state. This capability requires movement away from an approach to random bias based strictly on rank order and toward an approach based on heuristic valuations.

2.3.3 Heuristic Equivalency

Heuristic equivalency chooses randomly among all choices having a heuristic value within $H\%$ of the highest [92]. Gomes *et al.* employ heuristic equivalency within a systematic backtracking search [95, 92, 96]. As will be discussed later, the purpose of randomizing a systematic backtrack search in constraint satisfaction domains is to take advantage of the high variability in the runtime distribution that results. Cutting off such a search before it runs to completion and restarting the search procedure can lead to an algorithm with a large improvement in overall runtime.

Oddi and Smith explored using heuristic equivalency[6] within a stochastic sampling process for solving a generalized job shop scheduling problem [153]. One important distinction in this approach, as compared to rank-based approaches to stochastic sampling such as HBSS, is acknowledgment of the fact that a heuristic may be more or less informed in different decision-making contexts, and hence the degree of confidence in the heuristic can vary from decision to decision. Rather than rely on a static bias function as is used in HBSS, Oddi and Smith bias decisions dynamically. They define non-deterministic variants of search control heuristics that vary the degree of randomness as a function of how informed the heuristic is through the use of heuristic equivalency. A variant of this idea is also exploited by Cesta *et al.* in solving a resource-constrained project scheduling problem [30, 31, 32].[7]

One drawback to (or it can be argued feature of) heuristic equivalency is that only the choices within the threshold $H\%$ of the preferred choice are considered in the stochastic decision. There is a probability of 0 in choosing any of the other choices. This can require careful tuning of the threshold $H\%$ or else too many or too few choices will be considered equivalent.

2.3.4 Nested Partitions

Shi and Ólafsson have recently proposed a randomized algorithm for optimization problems that they call *Nested Partitions* [182]. We describe this here in that one of its steps uses stochastic sampling to evaluate how "promising" a region of the search-space is. The algorithm works as follows:

[6] They use the term *acceptance band* in place of heuristic equivalency.

[7] This algorithm is presented in detail later in Chapter 9 where it is used as a benchmark for our experiments.

1. Partitioning step: Let $\sigma(k)$ be the current "most promising" region of the search-space. At the root of the search, the entire search-space is this region (i.e., $\sigma(0)$ is the entire search-space). $\sigma(k)$ has $M_{\sigma(k)}$ subregions. Partition $\sigma(k)$ into its subregions, $\sigma_1(k), \ldots, \sigma_{M_{\sigma(k)}}(k)$. Define $\sigma_{M_{\sigma(k)}+1}(k)$ to be the region of the search-space that excludes $\sigma(k)$.

2. Random sampling step: Conduct a random sampling of N_j points in each region $\sigma_j(k)$ for $j = 1, \ldots, M_{\sigma(k)} + 1$. Shi and Ólafsson use iterative sampling in their experiments, but they do not make any restrictions on the sampling algorithm used. A more sophisticated procedure that uses heuristic guidance, such as HBSS, may lead to better results, but they have not investigated this.

3. Estimate the "Promising Index" step: Given the random samples obtained in the previous step, define the "promising index" of a partition $\sigma_j(k)$ as $I(\sigma_j(k))$. The promising index is based on the objective values of points in the region obtained from the random sampling. Shi and Ólafsson define this as the value of the best point found in the given region by the sampling step.

4. Backtracking step: Determine the "most promising" region for the next iteration. Take the region, $\sigma_j(k)$, with the best value of $I(\sigma_j(k))$. If two or more regions appear equally promising, then choose one of these at random. If this region is a subregion of $\sigma(k)$ (i.e., $j \leq M_{\sigma(k)}$), then the "most promising" region for the next iteration, $\sigma(k+1)$, is $\sigma_j(k)$. Otherwise, if the "most promising" region appears to be the surrounding region (i.e., $\sigma_{M_{\sigma(k)}+1}(k)$), then we backtrack to the super-region of $\sigma(k)$ (i.e., $\sigma(k-1)$).

Note that if we are at the root of the search-space, then there is no surrounding region since we partition the entire "most promising" region. Also note that if we are at the maximum depth of the search-space (i.e., the "most promising" region cannot be further partitioned), we do not stop the search at this point. We continue iterating where we compute the "promising indices" of this current "most promising" region and the surrounding region. If the current "most promising" region of maximum search-space depth continues to remain "most promising" then we continue the next iteration with it again as the "most promising". Such behavior may mean that the algorithm has converged (or is converging) upon the global optimal (i.e., the global optimal lies in this region). If this is not the case, then the global optimal must lie in the surrounding region. Since the sampling procedure

is required to give a positive probability of sampling any point in the given region, then we will with positive probability backtrack eventually to further explore other search-space regions. Shi and Ólafsson prove that the Nested Partitions method converges to a global optimum in finite time.

An example of how Nested Partitions may be applied to a combinatorial optimization problem is easily given for the TSP. The "most promising" region $\sigma(k)$ can be partitioned according to the city visited in the k-th position of the tour. Sampling from each of the subregions is straightforward. Cities 0 through $k-1$ are fixed according to the chain of superregions. City k is fixed for each of the subregions of $\sigma(k)$. And it is simply necessary to conduct a random sampling procedure for city positions greater than k in the tour for the purpose of estimating the "promising indices".

2.4 Stochastic Local Search Techniques

Many of the search techniques so far discussed have been constructive. That is, they begin with an empty solution (i.e., at the root of the search-space) and each decision that is made adds one more piece to the current solution. For example, in Bresina's telescope scheduling domain, at a given node in the search-space HBSS randomly chooses which observation to add next to the schedule that it is constructing. There is another entire collection of stochastic search techniques that can be lumped into a category called *local search*. A local search method begins with a complete randomly generated solution and iteratively modifies that solution in algorithm-dependent ways in its search for a solution that optimizes the problem's objective function. Many of these local search algorithms are then iterated some number of times beginning at other randomly chosen starting configurations. Figure 2.2 illustrates an example of local search.

2.4.1 Hill-Climbing

One relatively straightforward local search technique is that of *hill-climbing* [168]. A hill-climbing algorithm considers the space of candidate solutions as laid out on a landscape where altitude at a given point in the landscape is determined by the objective value of the candidate solution located at that point. Given some set of allowed local moves, hill-climbing chooses the move that advances from the current solution to the highest neighboring point in the landscape and continues until a locally optimal peek is reached and then

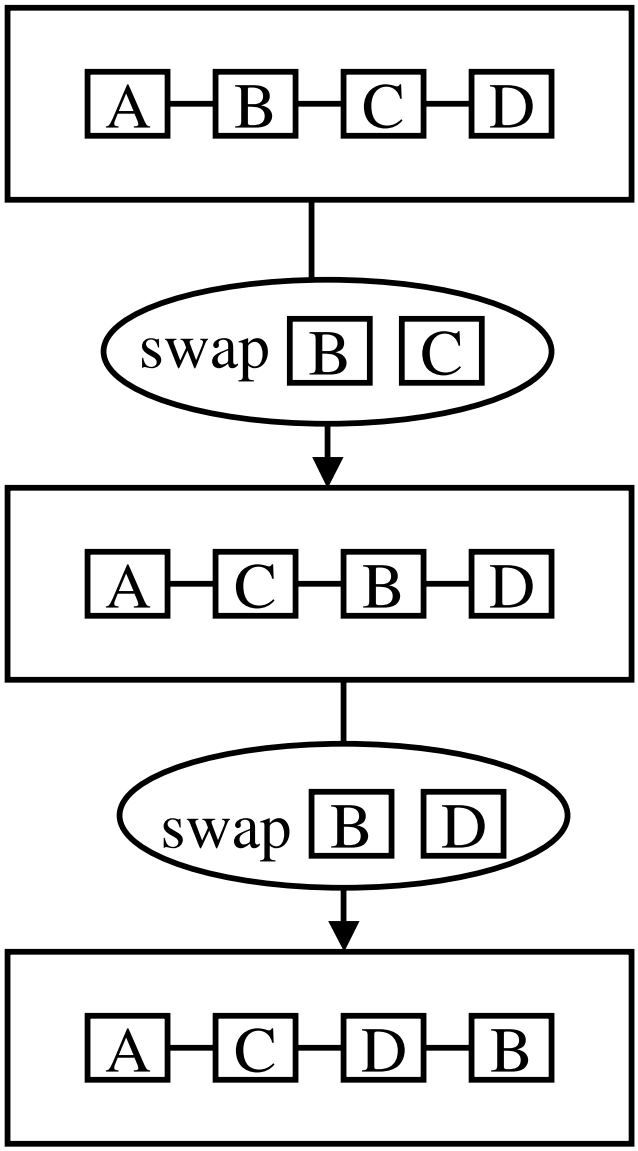


Figure 2.2: Example of a local search algorithm for a four city TSP: an initial state modified by a sequence of two operations.

iterates this process from another randomly chosen starting configuration. In a sense, starting from a random point in the search-space, hill-climbing then follows an uphill greedy heuristic. This is usually referred to as steepest-ascent hill-climbing. Another variation of hill-climbing is that of first-ascent hill-climbing in which each move that is made is the first found that improves your position on the fitness landscape. Movesets can be as simple as for example swapping two elements in a sequencing problem; or they can be more complex such as for example the approach to sequencing problems known as dynasearch [51]

in which a best set of independent pairwise swaps is found by dynamic programming. Dynasearch will be discussed in detail later in a specific problem domain context. Since each iteration of a hill-climbing search finds a locally optimal peek in the objective landscape, these algorithms are usually iterated a large number of times returning the best solution of those iterations in the hopes of stumbling upon the global optimum.

2.4.2 Simulated Annealing

Another popular local search technique, *simulated annealing* [118, 198], in a sense, can be viewed as a randomized, heuristically-biased, variant of the hill-climbing algorithm. Simulated annealing begins, as does hill-climbing, at a randomly chosen candidate solution configuration. Potential moves from this configuration are chosen from the set of allowed moves at random. If this move improves the search's current position in the objective landscape, it is made. Otherwise, if the move takes the search in a downhill direction on the objective landscape, then it is made with a probability that decreases as the degree of downhill movement increases and also decreases as the search length increases. In other words, if you consider a heuristic that values downhill moves according to how steep the descent, then the stochastic decision in simulated annealing is biased by this heuristic. The probability that a move is accepted is according to the Boltzmann distribution $\exp(\frac{-\Delta E}{T_i})$ where ΔE is the "cost" associated with making the move (i.e., amount of downhill movement in a maximization problem or uphill movement in a minimization problem) and T_i is the current *temperature*. Simulated annealing allows fewer downhill moves as the search progresses. This is accomplished by a *cooling schedule* that decreases the temperature parameter periodically (e.g., by updating according to $T_{i+1} = \alpha T_i$ for some positive constant $\alpha < 1$). The final stages of a simulated annealing search essentially correspond to a first-ascent hill-climb. Simulated annealing's allowance of downhill moves (i.e., moves that seemingly take you further from the optimal solution) serves the purpose of providing escape routes from local extrema.

There has been a great deal of success in applying simulated annealing to difficult and important optimization problems. A few successful applications of simulated annealing algorithms include: VLSI design [204], call routing in telecommunications networks [206], clustering [139], SPECT image reconstruction [136], digital filter design [10], graph drawing [60], job-shop scheduling [170], vehicle routing [170], and radio network base station positioning [120].

2.4.3 Threshold Accepting

Threshold accepting is a local search algorithm very closely related to simulated annealing [71]. Threshold accepting begins at a randomly chosen candidate solution configuration. Potential moves are chosen from the set of allowed moves at random. If this move improves the search's current position in the objective landscape, it is made. Otherwise, if the move takes the search in a downhill direction on the objective landscape, then it is made provided it does not degrade the current solution state by more than a threshold value V. The threshold V is decreased over time during the search analogously to the decreasing of the temperature parameter of simulated annealing.

2.4.4 Tabu Search

Tabu search is a local search technique closely related to hill-climbing, but which incorporates the concept of a *tabu list* or a list of moves that are disallowed for some period of time in the search [87, 88]. In tabu search, you begin with some randomly generated initial solution and you iteratively modify that solution according to some set of moves. The move that is selected is the one that improves your position on the objective landscape the most, if such a move exists. If no such move exists, then you make the move that is least bad. Your choice of moves at each step consists of what remains after eliminating some set of "tabu" moves from the complete moveset. For example, the tabu list often includes the inverse of the moves made in the last t steps. This wandering around while disallowing some moves is a very aggressive way of avoiding local optima. For example, consider a case where you make the "least bad" move from some current search state s_1 (i.e., there were no improving moves) and further consider that the "best" move from the resulting state s_2 returns you to state s_1. By disallowing that move (i.e., adding it to the tabu list) you can avoid cycles in your search and can escape this local optimum. Tabu search returns the best solution state that it wandered through during the course of its search.

2.4.5 Genetic Algorithms

Genetic Algorithms [102, 91, 142] are stochastic search algorithms that come under the broader category of *evolutionary computation*[8]. The genetic algorithm (GA) is inspired

[8]The field of evolutionary computation is very broad and to properly survey the entire field would require a volume unto itself. For the sake of brevity, I have chosen to simply describe what is perhaps the most commonly employed of the evolutionary computation algorithms – the genetic algorithm.

by evolution theory and in a sense solves problems by means of the artificial evolution of solutions. The problem of interest is first encoded as *chromosomes*. The most common representation of a chromosome is that of a bit string.[9] Tentative solution configurations are mapped to bit strings. For example, if our problem was a function optimization problem of four real-valued parameters, then we could potentially represent a problem configuration as a bit string of length 128 where each of the four real-valued parameters is represented by 32 of the bits. The *fitness* of a chromosome is defined according to the objective value of the solution represented by the chromosome. Chromosomes with higher fitness values correspond to solutions of better objective value.

The GA then evolves a *population* of N such chromosomes. It begins with a population of randomly generated chromosomes and then iterates for some number of generations. In each *generation*, N chromosomes are selected from the population of the previous generation with replacement[10] and copied into the initial population of the current generation. This selection process is typically stochastic and biased in some manner by the fitness values. For example, one common selection method weights a roulette wheel according to the fitness values of the population and then spins this wheel N times. Each chromosome of this new initial population is then paired up with exactly one other chromosome. Each pair of chromosomes mates with some probability C – the crossover rate. During *crossover*, some amount of genetic material (the bits) is swapped between the pair. There are various methods for selecting how many such bits and which bits to swap. After the crossover phase, each bit of each chromosome is mutated with some small probability M – the mutation rate. If *mutation* is to occur, then the value of the given bit is flipped. This process is iterated for some large number of generations and the best solution from the final generation is returned.

If you consider a GA that uses selection alone (i.e., $M = 0$ and $C = 0$), then you would have a very expensive way of choosing the best solution from the initial random population (with high probability). If you consider a GA that uses selection and mutation, but not crossover (i.e., $C = 0$), then you would have the equivalent of a population-based variation of simulated annealing. The mutation operator considers random moves on the fitness landscape, and through selection you keep uphill moves with high probability but there is still a non-zero probability of keeping around chromosomes corresponding to downhill moves. Now, consider the complete GA with selection, mutation, and crossover, and you

[9] Other common representations include: permutations, vectors of reals, and in the case of *genetic programming* [128] – program trees.

[10] Any chromosome has a chance of being selected multiple times and possibly not at all.

essentially have the equivalent of a population-based version of simulated annealing with a way of jumping to potentially interesting but not yet explored hills in the fitness landscape (i.e., crossover).

2.4.6 Success of Local Search

There are many successful examples of stochastic local search algorithms: Satisfiability [180], Traveling Salesperson [135], Largest Common Subgraph [37, 39, 40, 41], Graph Coloring [159]. Perhaps the most successful of these is that of the WALKSAT algorithm for Satisfiability [180]. It is also perhaps the best example of a simple, yet highly successful, stochastic search algorithm. WALKSAT begins with a random initial assignment of variables. It then iteratively picks at random unsatisfied clauses. For each of these randomly selected unsatisfied clauses, with probability p it flips the value of a randomly selected variable from that clause and with probability $1 - p$ it flips the value of one of the variables in that clause according to a greedy heuristic. This greedy heuristic chooses the variable that reduces the global number of unsatisfied clauses by the greatest amount. This amount can also be 0 or negative, so if no improvement can be made by flipping one of the variables of this clause the heuristic favors the one that is least bad.

2.5 Search Algorithm Design Motivated by Search-Space Analysis

A number of researchers recently have been studying characteristics of search-spaces to both identify particularly difficult (or easy) problem instances and also to influence the development of new algorithms that exploit problem-space features. For example, Frank *et al.* studied characteristics of plateaus[11] and benches[12] encountered by local search strategies [83]. They argued that by studying the characteristics of benches and plateaus for a problem beforehand that the results of this study could be used in the design of a local

[11] A "plateau" in the search-space of a local search algorithm is a region of the search-space such that: each member state of the plateau has the same objective value as all other member states; each member state is a direct neighbor to at least one other member state; and there does not exist a neighbor of any member state of the plateau for which that neighbor has a better objective value than that of the plateau. A "local optima" is an example of a plateau of size 1.

[12] A "bench" is related to a plateau, but has the additional characteristic that one or more members of the plateau neighbor states with better objective values. In other words, a bench is a flat region of the search-space with one or more exits.

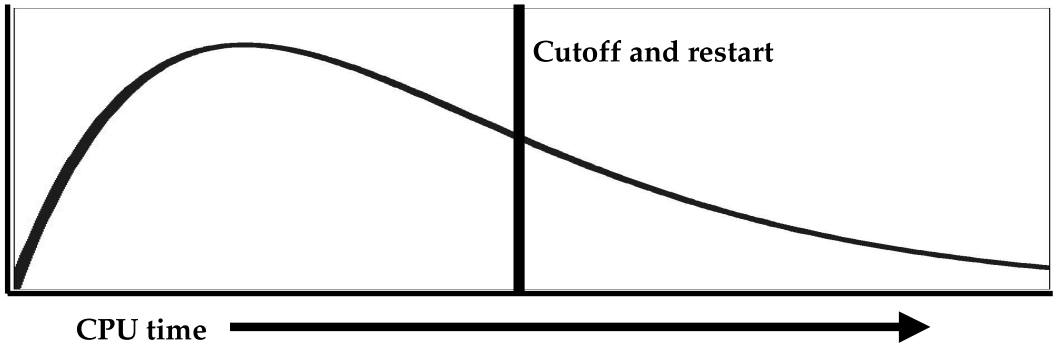
search technique for the problem. For example, for problem spaces with relatively small plateaus, the methods of tabu search, and other similar algorithms, that disallow the revisiting of some search states may be effective; while for problem spaces with very large (and difficult to escape) plateaus and benches, restarting the local search algorithm when encountering a plateau may be more effective than spending any search time on the plateau. A number of others have studied various problem domains to identify search-space features that are correlated to problem difficulty [49, 154, 185, 202, 203]. In all of these examples, characteristics of classes of difficult problem instances are identified and these characteristics can potentially lead to new or improved algorithms for the problems.

2.5.1 Rapid Randomized Restarts

One particularly successful example of a search-space analysis motivating algorithm design is that of heavy-tailed runtime distributions as motivation for rapid randomized restarts. In constraint satisfaction problem-solving (CSP) domains, Gomes *et al.* have observed a heavy-tailed nature of the search runtime distributions across different solution trajectories of the search-space [95, 92, 96]. That is, for a given CSP instance, the distribution of run-times for backtrack search procedures, across multiple runs, exhibits formally heavy-tailed behavior. Heavy-tailed distributions are extremely non-standard distributions that capture high variability and erratic behavior in random processes. These distributions are characterized by infinitely long tails, an infinite mean, and an infinite variance. Such heavy-tailed nature offers an explanation to the high variability of runtimes exhibited by backtrack search procedures.[13] It also provides a conceptual rationale for adopting an approach that cuts off search at specified computational time-limits (in essence, abandoning trials that belong to the heavy-tail) and repeatedly restarting along different solution paths. This approach allows the overall search process to reach more productive regions of this search space sooner. Gomes *et al.* call this approach *rapid randomized restarts* [96]. The approach has been shown to significantly reduce the run-times of complete search procedures on hard problem instances across a range of problem domains [92, 96, 153]. Figure 2.3 illustrates rapid randomized restarts and the motivation behind this procedure – the heavy-tailed nature of the runtime distributions of backtrack search in constraint satisfaction domains.

A major design consideration of a rapid randomized restart search strategy is defining the search cutoff point. Luby *et al.* showed provably optimal restart policies under the

[13] For bounded search-spaces, a bounded heavy-tailed runtime distribution has an exponentially long right-hand tail in the size of the space, an exponential mean in the size of the space, and an exponential variance.



- **Heavy-tailed behavior**
  - **Unbounded search spaces**
  - **Infinitely long tail**
  - **Infinite mean**
  - **Infinite variance**
- **Bounded heavy-tailed behavior**
  - **Bounded search spaces**
  - **Exponentially long tail**
  - **Exponential mean (in size of input)**
  - **Exponential variance (in size of input)**

Figure 2.3: Rapid randomized restarts and the heavy-tailed nature of backtrack search in constraint satisfaction domains.

assumptions that runs are independent and the only feasible observation is the length of a run [137]. Given complete knowledge of the runtime distribution, the optimal restart policy is to set a fixed cutoff point of c backtracks such that the expected time to find a solution is minimized by this cutoff. Luby *et al.* further showed that if nothing is known about the runtime distribution, then the schedule of cutoffs, $\{1, 1, 2, 1, 1, 2, 4, \ldots\}$, gives an expected total runtime within a log factor of the optimal fixed cutoff. By considering other features of a problem solver's state during the search in addition to the length of a run, both Horvitz *et al.* and Kautz *et al.* showed that Bayesian models can be constructed that lead to dynamic restart policies based on solver state observations that are superior to the optimal static policy [106, 115]. Ruan *et al.* further relaxed the run independence assumption and considered restart policies with dependent runs [164].

2.6 Combining Multiple Search Algorithms

In this Section, we overview general methods for combining multiple stochastic search algorithms into a single stochastic problem solver. Often different algorithms for the same problem perform better for problem instances with different characteristics. For example, Lagoudakis *et al.* considered the problem of sorting an array of numbers and used a Markov decision process model to determine an optimal policy (given a particular computer archi-

tecture) for choosing from among insertion sort, merge sort, and quicksort for the sorting performed at each recursive step based on the size of the subset of numbers to be sorted during this recursive step [129]. For more complex and difficult problems, the problem of combining search algorithms to leverage their advantages is not so simple. We now overview a few general methods for search algorithm combination.

2.6.1 Algorithm Portfolios

An *algorithm portfolio* is a collection of different algorithms and/or different copies of the same algorithm running on different processors or with interleaved execution on one or a few processors [94, 93, 112]. The primary motivation behind such an approach is to combine algorithms into a portfolio such that the portfolio approach gives superior performance compared to the component algorithms. By combining several algorithms with large variance in the runtime distributions into such a portfolio of algorithms, it is possible to take advantage of the non-negligible probability of finding a solution with a really short run of one of the algorithms. For example, systematic backtrack search randomized via heuristic equivalency in constraint satisfaction domains exhibits a high variability in the distribution of runtimes across multiple runs of the search. By running several copies of such an algorithm in parallel, one can take advantage of this runtime variability and improve their probability of finding a feasible solution quickly on a short run of one of the copies of the algorithm, or similarly, by running several different such "risky" algorithms in parallel. Gomes and Selman demonstrate the efficacy of the algorithm portfolio approach by showing for mixed integer programming (MIP) that although when running a single process a best-bound approach is superior to a depth-first strategy, a portfolio of high-variance depth-first runs outperforms a portfolio of best-bound runs [94]. The rapid randomized restart approach of Gomes *et al.* discussed above takes similar advantage of this variability by cutting off a randomized systematic search procedure at a pre-specified runtime prior to finding a solution and restarting [95, 92, 96]. Hoos has also similarly shown that stochastic local search algorithms in domains such as SAT and CSP can often exhibit a run-time speed-up through parallelization as a portfolio [105, 104]. In these domains, some stochastic local search algorithms are characterized by exponential run-time distributions which account for the effectiveness of parallelization.

2.6.2 Asynchronous Teams

Similarly, in combinatorial optimization domains, Talukdar *et al.* have acknowledged that all of the available search algorithms have strengths and weaknesses. Some produce very good solutions but are slow; while others are fast, but with a tradeoff of solution quality. Given this, they have developed a framework that allows a number of these algorithms to search for solutions cooperatively. They call this framework an *asynchronous team* (A-Team) [192, 35, 191, 169]. Essentially, an A-Team is a set of memories and a set of autonomous agents. Each memory may be comprised of a set of complete tentative solutions to the problem or partial solutions to the problem. Perhaps, one memory may use one problem representation while another memory may use some other representation. Some subset of the agents are implementations of various search algorithms – constructive agents. For example, one agent may be a simulated annealing algorithm; while another agent may be a hill-climber. Each agent takes as its input tentative solutions selected from a given input memory, improves (or tries to improve) upon those solutions, and then outputs its solutions to a given output memory. The input and output memories can be the same for a given agent, multiple agents may share an input and/or an output memory, and the input memory of one agent may be the output memory of some other agent (and vice-versa). In addition to the constructive agents, there are destructive agents. A destructive agent removes from a given memory tentative solutions that it deems are not very promising. For example, a destructive agent may use a procedure similar to roulette wheel selection in a GA to select solutions to delete from a memory biased according to how unfit they are. Or for example, they may use a mechanism similar to tabu lists from tabu search to delete solution configurations that satisfy some criteria. All of the agents in the A-Team execute asynchronously with no centralized control governing the amount of execution time each is granted. Talukdar *et al.* have shown this framework to often give superior performance as compared to using any of the underlying component algorithms alone. A-Teams appear to be an effective framework for building a hybrid search algorithm from a collection of search algorithms for a problem. An A-Team is essentially an algorithm portfolio with the ability of algorithms to communicate and improve upon each others solutions.

2.6.3 Meta-Planner

A related idea from the AI planning community is the *Meta-Planner* of Howe *et al.* that combines several planning algorithms into a sort of algorithm portfolio (though they do not

use the term) [111]. Howe *et al.* performed a study of a number of planning algorithms on a suite of benchmark planning problems. They showed that no one planner dominated across the problem set and that each of the planners performed best on one or more problem instances as compared to the other planners in the study. They also showed that problem features such as number of actions, number of predicates, number of goals, number of predicates in the initial condition, and the number of objects can be predictive of the performance of a particular planner on a problem instance.

Using the data from the study, Howe *et al.* developed Meta-Planner. Meta-Planner orders the component planning algorithms according to $\frac{P(A_i)}{T(A_i)}$ where $P(A_i)$ is the expected probability of success of algorithm A_i and $T(A_i)$ is the expected runtime. These models are linear regression models of the performance data from the study. After ordering the algorithms, they are executed according to a round robin control strategy. An algorithm is executed for a time slice equal to the expected time required to solve the problem. If it solves the problem, then planning comes to an end. If it fails, then the planner is removed from the round robin. If it uses its allocated time slice without finding a solution and without failing, then it is suspended and control passes to the next planner. This continues until either a solution is found, all planners fail, or a preset threshold amount of time is exceeded.

2.7 Summary

In this Chapter, various related work on search algorithms have been discussed.

Discrepancy search algorithms like LDS and DDS are motivated by the idea that you may have a good search ordering heuristic for your problem, but that the heuristic may make some small number of mistakes during the course of the search. LDS and DDS are systematic search algorithms designed with constraint satisfaction domains in mind and attempt to ensure completeness by exhausting the search-space if necessary. In optimization domains, these systematic procedures may suffer from scalability issues.

The stochastic sampling algorithms presented are more closely related to the work of this thesis. Their underlying motivation is similar to LDS and DDS. However, rather than trying to guarantee completeness and optimality, stochastic sampling algorithms trade these off in favor of scalability and trying to find "good enough" solutions quickly. The novel algorithm that will be presented later in this thesis falls into this category of stochastic sampling.

Stochastic local search is another class of stochastic search algorithms in which some initial complete solution (or solutions) is (are) iteratively modified via various operators; rather than building solutions from scratch on each iteration as is done by stochastic sampling search. The approach to search control that will be taken later in this thesis, although designed with stochastic sampling in mind, may also be relevant to local search techniques. For example, later in this thesis, results are presented involving a hybrid algorithm where the solutions generated by the stochastic sampler are then taken to local extrema by a hill-climbing algorithm. The resulting algorithm is then controlled by the search control framework, QD-BEACON, of this thesis.

Search-space analysis recently has proven fruitful in providing motivation for search algorithm design. For example, we saw Gomes *et al.*'s rapid randomized restart technique in which a complete randomized backtrack search is abandoned and restarted if some pre-specified runtime is exceeded. This practice is motivated by the heavy-tailed nature of the runtime distributions of backtrack search in these problem domains.

Algorithm portfolios offer a method of taking advantage of high variance runtime distributions and combining many "risky" algorithms with parallel (or interleaved) execution to create a single meta-algorithm with reduced variance and a significantly shorter expected overall runtime. Similarly, A-Teams were seen as a potentially effective framework for allowing a collection of stochastic search procedures to collaborate in their search for a solution to a given problem – effectively taking the good aspects of the component algorithms. But, within these architectures, all agents (or search algorithms) work autonomously and are essentially given equal priority by the execution architecture. Via the search control framework presented in this thesis, QD-BEACON, it may be possible to more effectively coordinate the share of execution time amongst some subset of the components of an A-Team or an algorithm portfolio. More important for the aims of this thesis, QD-BEACON is a potentially effective mechanism for combining multiple stochastic samplers into a single hybrid search algorithm in much the same way that algorithm portfolios and A-Teams provide an architecture for combining multiple search algorithms.

Chapter 3

Related Work: Metareasoning and Metalevel Search Control

3.1 Overview

In this Chapter, I discuss related work on metareasoning and metalevel search control. Section 3.2 discusses the work that has been done with metalevel control parameter optimization – particularly the work that has been done with GAs and considering the metalevel problem as one of static optimization. Section 3.3 discusses research dealing with making search control decisions based on models of search performance learned during execution (e.g., deciding when to restart the search or using heuristics that adapt according to search progress). Section 3.4 discusses metareasoning and the work that has been done with anytime algorithms. A summary concludes the Chapter in Section 3.5.

3.2 Metalevel Control Parameter Optimization

One important issue that often needs to be resolved in the design phase of a stochastic search algorithm is how to set the various control parameters of the algorithm. The genetic algorithm (GA) is a good case example to consider due to the large number of parameters involved. For example, there is the mutation rate, the crossover rate, various choices of crossover operator, the population size, and potentially many more specialized parameters depending upon the choice of these operators and other more advanced issues with the GA. Many researchers have considered this issue of GA control parameter selection [62, 61, 98, 172, 22, 205, 28, 72, 42]. One of the earliest studies of GA control parameters is that of De

Jong [62]. He analyzed a class of GAs for function optimization. De Jong's optimal control parameters became widely used despite lack of knowledge of optimality with respect to problems outside of his test collection. Schaffer *et al.* expanded upon De Jong's test suite and performed a more systematic study of the effects of the control parameters [172].

Grefenstette saw the problem of tuning the control parameters of the primary GA as a secondary or metalevel optimization problem and was one of the first to apply a metalevel GA approach [98]. Grefenstette's metalevel GA was parameterized with De Jong's optimal control parameters: population of 50, cross-over rate of 0.6, mutation rate of 0.001, generation gap of 1.0, scaling window of 7, and an elitist strategy [62]. Grefenstette's results showed a slight improvement over De Jong's with control parameters: population size of 30, cross-over rate of 0.95, mutation rate of 0.01, and an elitist strategy [98].

Others have also applied meta-level GAs to control parameter optimization. Bramlette presents modified methods of selecting initial populations and mutation operators for improving the performance of GAs for function optimization [22]. He tests his techniques on a meta-level GA to optimize the control parameters for some other GA. Wu and Chow apply a meta-level GA approach to optimizing the control parameters of GAs for nonlinear mixed discrete-integer optimization problems [205]. Cicirello and Smith acknowledge the computational expense associated with computing the fitness function within a metalevel GA (since it requires executing the primary GA some number of times) and instead build a neural network model of the performance of the primary GA to use as the fitness function at the metalevel [42]. They apply their approach to optimizing the control parameters of a GA for the largest common subgraph problem.

Metalevel optimization is not limited to GA control parameters. Koch *et al.* meta-optimize the control parameters of an evolution strategy [119]. De Jong applies a GA approach to control parameter optimization of dynamical systems, not necessarily to control parameters of a GA [61]. Morley uses a GA to optimize the parameters of the bidding rules in his market-based mechanism for dynamically assigning trucks to paint booths in vehicle paintshops [144, 143]. Campos *et al.* do likewise for their biologically motivated multi-agent system for the same problem [27]. Many more examples of metalevel control parameter optimization can be given.

3.3 Search Control Guided by Learned Models

3.3.1 The STAGE Algorithm

The success of local search algorithms, such as hill-climbing, simulated annealing, etc., often depends largely on characteristics of the objective function landscape for the problem at hand (for example, quantity and frequency of local optima, size of plateaus and benches, etc.). Boyan observes that often it is possible to define an alternative objective function consisting of the same global optima but which may have a landscape that is more suitable for effective local search than the original objective function. Boyan's STAGE algorithm is designed with this motivation [20, 21, 19]. The STAGE algorithm takes as input various problem state features, including the original objective function value of a given problem state. It also takes as input a local search algorithm (e.g., simulated annealing or hill-climbing). It then proceeds in stages (thus its name STAGE). During the first stage, the algorithm follows the local search in the original objective landscape. When a local optima is encountered, STAGE then uses statistical learning (e.g., linear regression, locally-weighted regression) to learn a value function mapping the problem state features of a local search start state to the objective value of the expected local optima encountered by a search beginning at that state. After this learning stage, STAGE then applies the local search to the new objective landscape given by this learned model. It then cycles back to the first stage performing a local search in the original objective function domain, and so forth. If the search in the learned evaluation function space does not move the overall search to a new local optima, then STAGE restarts from a randomly determined starting solution.

3.3.2 Expected Cost Improvement Distributions

Sadeh *et al.* define what they call the *Expected Cost Improvement Distribution* (ECID) and use this tool to motivate the restarting of a simulated annealing search [170]. They recognize that finding near-optimal solutions to combinatorial optimization problems with simulated annealing can often be an expensive process computationally requiring a large number of restarts. To deal with this computational expense, their procedure learns to recognize and abandon runs that do not look very promising in favor of restarting the search.

Across multiple runs of simulated annealing on the given problem instance, they build probabilistic models at a set of temperature checkpoints. Each of these is a distribution of the expected improvement in cost of continuing the search below the given temperature

threshold. They assume these distributions can be modeled as normal distributions. The mean of this distribution is:

$$\mu_n^t = \frac{\sum_{i=1}^{n}(c_i^t - c_i^0)}{n} \tag{3.1}$$

and the standard deviation is:

$$\sigma_n^t = \sqrt{\frac{\sum_{i=1}^{n}((c_i^t - c_i^0) - \mu_n^t)^2}{n-1}} \tag{3.2}$$

where c_i^t is the cost of the best solution at check point temperature t in the i-th run and c_i^0 is the cost of the best solution obtained at temperature $T = T_1$.

Then, during the $(n+1)$-th restart of the simulated annealing search, this distribution is computed for each check point given the results of the first n restarts. The current run is abandoned if:

$$\frac{c_{n+1}^t - \mu_n^t - x_n}{\sigma_n^t} > \tau \tag{3.3}$$

where x_n is the cost of the best solution found during the previous n runs and τ is a threshold value.

Whenever a run is abandoned or completed, Sadeh *et al.*'s approach then uses the ECIDs to decide whether to begin a fresh run of simulated annealing or to "reanneal" the solution of a previous run at one of the checkpoint temperatures. They consider three possibilities for making this decision. The first chooses the restart point expected to maximize the rate at which the cost of the current best solution will improve. The second chooses randomly among those within some threshold of the restart point expected to maximize the same criteria. The third chooses according to a Boltzmann distribution weighting higher those choices with a higher expected rate of improvement over the best solution.

3.3.3 Ant Colony Optimization

A heuristic-biased stochastic search technique that has gained in popularity in the computational intelligence community in recent years is the *Ant Colony Optimization (ACO)* meta-heuristic [67, 68, 69]. ACO uses a population of ant-like agents that communicate indirectly via trail laying and following to build solutions to the optimization problem at hand. Each of these ant-like agents builds a solution stochastically and biases its random decisions according to the values of a heuristic as well as the values of artificial quantities of pheromone that evolve over time based on obtained solution quality. For instance, in one variation of ACO an artificial ant chooses search-space branch b_i from among a set of

alternatives $\{b_1, b_2, \ldots, b_n\}$ with probability:

$$P(b_i) = \frac{h(b_i)^\alpha p(b_i)^\beta}{\sum_{j=1,\ldots,n} h(b_j)^\alpha p(b_j)^\beta} \tag{3.4}$$

where $h()$ returns a heuristic value of the choice, $p()$ returns the quantity of artificial pheromone located on the search-space branch, and α and β are system parameters that trade-off how heavily the heuristic and the pheromone influence the bias of the stochastic search. The pheromone quantities are updated by one of several methods (e.g., all of the ants update the search-space branches which they used according to the quality of their respective solutions).

There are numerous applications of ACO to various combinatorial optimization problems including: the sequential ordering problem [85], job shop scheduling [197], flow shop scheduling [189], vehicle routing [26, 86], bus driver scheduling [78], tardiness scheduling problems [5, 65], and resource-constrained project scheduling [141]. There are also extensions of ACO designed for dynamically changing optimization problems in a few domains. For example, Schoonderwoerd *et al.* [173, 174] have developed an effective system called Ant Based Control (ABC) for adapting routing tables in circuit-switched networks based on the ACO framework. Similarly, Di Caro and Dorigo [66] have developed a system called AntNet in which artificial ants adapt the routing tables of packet-switched networks. Also based on the ACO paradigm, Cicirello and Smith developed a method for routing jobs to multi-purpose machines in dynamic flow-shops to minimize setups that is called AC^2 [43]. It unfortunately is prone to convergence to sub-optimal equilibrium in a game-theoretic sense [38].

The ACO framework is an improvement over that of HBSS in that it uses the actual heuristic values within its stochastic decisions and is thus capable of varying the degree of randomness according to how informed the heuristic is, but in general its evolution of artificial pheromone quantities and convergence toward a solution are slow and not well-suited for applications where you desire a solution quickly. It is best-suited to problems where you are interested in finding near-optimal solutions and where you can afford extra computation time. Its performance can be compared to evolutionary methods such as genetic algorithms.

3.3.4 Adaptive Probing

Ruml describes a stochastic sampling method that he calls *adaptive probing* [165, 166, 167]. In adaptive probing, the search builds online a probabilistic model of the cost of choosing any given action at any depth of the search. Adaptive probing assumes that the effects of choosing the i-th most preferred action at depth j is the same for all search states at depth j. Each estimated cost is assumed to be the mean of a normal distribution, with all action costs at a given depth of the search space having the same variance. The cost of taking action i at depth j is $a_j(i)$. If the depth of the search tree is d and the branching factor b, then it is necessary to maintain db of these cost models. Ruml uses a perceptron learning rule to update these parameters after each probe from the root of the space to a leaf according to the observed leaf cost (objective value of a solution). If the d actions taken during a probe are $\{i_1, \ldots, i_d\}$, then each of the d cost means $\{a_1(i_1), \ldots, a_d(i_d)\}$ are updated according to $\eta \frac{l_k - \hat{l}_k}{d}$ where l_k, $\hat{l}_k$, η are the observed leaf cost, the estimated leaf cost, and the learning rate respectively. During a probe, actions are chosen according to the probability that it leads to a solution with a lower cost (lower objective value).

In a sense, adaptive probing can be seen as a stochastic sampling method that attempts to learn a search heuristic for the problem online. A heuristic can also be incorporated into the method by specifying an initial probing policy based on the heuristic and then slowly discounting its use in favor of the learned model. Ruml shows that adaptive probing is competitive to systematic search procedures such as DFS, LDS, and DDS in many cases except when the heuristic is very strong. In this case, it suffers from the need to learn that the heuristic is a good one. Thus, it is not of much use to us in domains where we know we have a good heuristic such as in many scheduling domains where the OR community has spent much time and effort in designing strong dispatch heuristics. Furthermore, as the search-space grows so do the number of parameters we need to maintain and learn with the search. The db parameters can get difficult to deal with for problems with large branching factors and large depths. For example, a 1000 city TSP would require maintaining 1,000,000 cost models. Also, it assumes that the actions (as well as the costs of those actions) at a particular depth of the search are the same independent of your previous actions and how you arrived at that level of the search. This is often an impractical assumption. For example, during a tree search for a TSP, the actions (or cities available to visit next) available at a search node at depth j are dependent upon the path taken from the root to that node at depth j (i.e., the cities already visited). The technique is interesting just the same in that it attempts to do something closely related with the approach of this thesis. Ruml's

adaptive probing builds and utilizes online a model of the costs of choosing actions learned during the search; whereas our approach builds and utilizes online a model of the objective values of the leaf nodes (feasible solutions) of the search-space.

3.3.5 Hyperheuristics

Hyperheuristics represent a new type of local search algorithm [55, 54, 56, 58, 57, 117]. A hyperheuristic is a method of combining multiple local neighborhood search operators (e.g., insertions, removes, adds, swaps) into a single local search algorithm.

In the most basic form of a hyperheuristic, Cowling *et al.* define what they call *simple hyperheuristics* [55, 54]. Specifically, they define four simple hyperheuristic algorithms: 1) SimpleRandom repeatedly chooses a local operator uniformly at random, applying it once; 2) RandomDescent repeatedly chooses an operator uniformly at random, applying it iteratively until no further improvement can be made; 3) RandomPerm chooses a random permutation of all local operators and applies each one consecutively in the chosen order, cycling around to the first; and 4) RandomPermDescent does the same as RandomPerm except that each local operator is applied repeatedly until no further improvement can be made before moving on to the next operator in the permutation.

The simple hyperheuristics listed above are not particularly interesting in light of the topic of this Chapter. As so far described, hyperheuristics would seem a better fit in the discussion of Chapter 2. But Cowling *et al.* also define what they call a *choice function hyperheuristic* which adapts a functional ranking of each low-level search operator [55, 54]. The choice functions are calculated based on the individual performance of a low-level operator, the joint performance of pairs of operators (e.g., a particular operator might help some other operator improve the current solution, while alone, that operator might not lead to any improvement), and the amount of time that has elapsed since the operator's last use. The choice function of a local neighborhood operator N_j is defined as:

$$f(N_j) = \alpha f_1(N_j) + \beta f_2(N_k, N_j) + \delta f_3(N_j), \tag{3.5}$$

where N_k is the previous operator applied.

The function $f_1(N_j)$ is the individual performance of an operator and is updated during the search according to the following:

$$f_1^{new}(N_j) = \frac{I(N_j)}{T(N_j)} + \alpha f_1^{old}(N_j), \tag{3.6}$$

where $I(N_j)$ is the change in objective value since the last time operator N_j was applied and $T(N_j)$ is the amount of CPU time taken.

The function $f_2(N_k, N_j)$ is the joint performance of N_k and N_j and is updated during the search as follows:

$$f_2^{new}(N_k, N_j) = \frac{I(N_k, N_j)}{T(N_k, N_j)} + \beta f_2^{old}(N_k, N_j), \tag{3.7}$$

where $I(N_k, N_j)$ is the change in objective value since the last time N_j was called immediately after N_k and $T(N_k, N_j)$ is the amount of CPU time taken.

The function $f_3(N_j)$ is the amount of CPU time that has elapsed since the last time local neighborhood operator N_j was applied.

The α, β, and δ are parameters that determine the degree to which each of these three parts play a role in the determination of the choice function. These parameters can either be fixed beforehand via some ad hoc tuning procedure [55, 58, 57] or they can be adjusted with the search via simple reinforcement learning update rules [54, 56, 117].

Furthermore, an approach called *hyper-GA* has been considered in which a genetic algorithm is used to evolve sequences of local neighborhood operators [53]. In the hyper-GA, each chromosome in the population represents a sequence of local neighborhood operators. The goal of the hyper-GA search is to find a sequence of local neighborhood operators that when applied to the problem (e.g., in a manner analogous to the RandomPermDescent simple hyperheuristic) leads to good solutions. The *adaptive length chromosome hyper-GA (ALChyper-GA)* extends this idea beyond using fixed length sequences of local neighborhood operators and allows the GA to search for the length of such a sequence in addition to the sequence itself [99].

Others have begun using hyperheuristic approaches to combine multiple problem specific heuristics into improved problem solvers. For example, Ross *et al.* consider the bin-packing problem [162]. Their hyperheuristic approach to the bin-packing problem uses a learning classifier system to learn a choice function for deciding which of a set of 14 bin-packing heuristics to apply to the current search state. Specifically, the search-space is that of a constructive search, rather than a local search; and each of the heuristics in the choice set can be used to decide things such as which item to place into which bin.

Rossi-Doria and Paechter have used an evolutionary algorithm in solving timetabling problems [163]. In their approach, the genetic representation encodes a sequence of problem specific heuristics, rather than encoding a solution configuration. The fitness of an

individual chromosome of the population is based on the quality of the solution found by using the sequence of heuristics to construct a solution to the given problem instance.

A system closely related to hyperheuristics is Epstein's FORR architecture [75]. FORR combines the advice of multiple heuristics by weighted voting and by a mechanism for learning the weights used in the voting. Epstein *et al.* use FORR in the domain of constraint programming within their Adaptive Constraint Engine [76].

Also related to hyperheuristics is *the DragonBreath Engine* of Nareyek which uses a local search method to solve constraint programming problems [148, 149]. In this system, each constraint in the problem has a cost determined by the degree to which it is satisfied at any given time during the search. For example, for a constraint that is satisfied, the cost is 0; and unsatisfied constraints have some positive cost value. During each step of the search, one of the constraints that has positive cost is selected. Upon selection, a heuristic is chosen to improve the cost of the constraint by changing the values of some set of variables. For each constraint, DragonBreath may have one or more heuristics that can be used to improve the selected constraint's cost. Each of these heuristics is assigned a weight by the system and the weights are adapted during the search by simple reinforcement learning rules. That is, whenever a constraint is chosen for improvement, the weight of the heuristic most recently chosen for that constraint is adjusted based on the current cost of the constraint and the cost of that constraint the last time it was selected. If the cost showed improvement, then the weight for the heuristic is increased (and decreased otherwise). After adjusting the weight of the most recently chosen heuristic, the heuristic with the highest weight for the selected constraint is applied to improve the current cost of that constraint. Nareyek considers several alternatives for the weight adjustment rules.

3.3.6 Wasp-Inspired Scheduling

Theraulaz *et al.* present a model for the self-organization that takes place within a colony of wasps [196]. Interactions between members of the colony and the local environment result in dynamic distribution of tasks such as foraging and brood care. In addition, a hierarchical social order among the wasps of the colony is formed through interactions among individual wasps of the colony. This emergent social order is a succession of wasps from the most dominant to the least dominant.

This model of wasp behavior describes the nature of interactions between an individual wasp and its local environment with respect to task allocation [196]. They model the colony's self-organized allocation of tasks using what they refer to as response thresholds.

An individual wasp has a *response threshold* for each zone of the nest. Based on a wasp's threshold for a given zone and the amount of stimulus from brood located in that zone, a wasp may or may not become engaged in the task of foraging for that zone. A lower threshold for a given zone amounts to a higher likelihood of engaging in activity given a stimulus. Bonabeau, Theraulaz, and Deneubourg discuss a model in which these thresholds remain fixed over time [16]. Later they consider that a threshold for a given task decreases during time periods when that task is performed and increases otherwise [195]. They give examples of different insect species for which each of their mathematical models most closely agree.

Bonabeau *et al.* demonstrate how this model leads to a distributed system for allocating mail retrieval tasks to a group of mail carriers [15]. Although they deal with a "toy" problem, they successfully illustrate the potential of systems inspired by the underlying wasp behavioral model. Campos *et al.* take the model a step further illustrating a connection between market-based mechanisms and wasp-inspired systems [27]. They apply a system inspired by the natural model to a simulation of the real-world vehicle paintshop problem of Morley [144, 143]. The insect-inspired bidding rules of Campos *et al.* result in a slight improvement over the market-mechanism of Morley. Similarly, Cicirello and Smith adopt this task allocation model for their *routing wasps* to assign (or route) jobs to machines in a distributed factory setting [47, 44, 46]. It should be noted that no actual search is being performed to select which machine to assign a given job. Using simple update rules, the response thresholds adapt over time. This adaptation of response thresholds can be thought of as learning a heuristic that evaluates the trade-off associated with bidding or not bidding on a particular job. This heuristic is then followed stochastically by each machine agent via a single sample drawn by a stochastic sampling procedure to decide whether or not to bid on the given job.

This model of wasp behavior also describes the nature of wasp-to-wasp interactions that take place within the nest [196]. When two individuals of the colony encounter each other, they may with some probability interact with each other. If this interaction takes place, then the wasp with the higher social rank will have a higher probability of dominating in the interaction. Through such interactions as these, wasps within the colony self-organize themselves into a dominance hierarchy. Theraulaz, Bonabeau, and Deneubourg discuss a number of ways of modeling the probability of interaction during an encounter which range from always interacting to interacting based upon certain tendencies of the individuals [194]. In the *scheduling wasp* definition of Cicirello and Smith, we used this concept to

model job priority and to prioritize jobs in a given machine queue [47, 45]. These scheduling wasps are at the basis of the WHISTLING algorithm presented later in this thesis.

3.3.7 Interval Estimation

Much of the work on reinforcement learning, especially exploration policies, can be considered related to the work of this thesis. Here I will summarize one such exploration strategy that is most relevant and which deals with the issue of selecting from among multiple actions in a reinforcement learning domain. This is similar to our need to select from among multiple heuristics within a stochastic search algorithm. Kaelbling introduced the reinforcement learning strategy known as *interval estimation* [113]. In interval estimation, for each action, a confidence interval for the expected reward is computed. The action with the largest upper bound on this confidence interval is selected and executed. The rationale is that an action with a large expected reward would have a large upper bound in the confidence interval. Also, an action that has not been tried sufficiently often will also tend to have a large upper bound since the width of the confidence interval will be larger with fewer samples.

3.4 Metareasoning and Anytime Computation

3.4.1 Anytime Algorithms

An *anytime algorithm* is an algorithm whose result is expected to improve in terms of quality as a function of increasing computation time. Dean and Boddy originated the term, anytime algorithm [64, 12, 13]. They used this term as a name for the class of algorithms with the following properties:

1 Preemptability: the algorithm can be suspended and resumed with minimal overhead.

2 Interruptibility: the algorithm can be terminated at any time and and will return some answer.

3 Well-Behaved Improvement: the answers returned improve in some well-behaved manner as a function of time.

Horvitz simultaneously developed related ideas and referred to algorithms with these properties as *flexible inference strategies* [107, 108]. There are a number of other properties

that have been found desirable for a number of reasons by researchers. Zilberstein outlines some of these additional properties [208]:

4 Measurable Quality: the quality of an approximate result can be determined precisely.

5 Recognizable Quality: the quality of an approximate result can easily be determined at runtime (i.e., in constant time).

6 Monotonicity: the quality of the result is a nondecreasing function of time and input quality.

7 Consistency: the quality of result is correlated with computation time and input quality.

8 Diminishing Returns: the improvement in solution quality is larger at the early stages of computation, and diminishes over time.

It can be noted that these additional properties can be used in various combinations to give more specific definitions of Dean and Boddy's third property.

There are many types of algorithms that fall into this class of anytime algorithms. For example, all of the algorithms discussed in Chapter 2 can be employed as an anytime algorithm. Discrepancy search methods such as LDS and DDS can be implemented in such a way to allow for interruption and to return the best solution found by the time of interruption. Stochastic sampling algorithms as well can be implemented in this way. Local search algorithms are designed to iteratively improve upon some initial solution or set of initial solutions. Some other types of algorithms that can be used to develop anytime computation procedures include (but are not limited to) numerical approximation algorithms (e.g., Taylor series approximation) [158] and dynamic programming [9]. In general, any truncated complete search procedure can also be used in an anytime setting.

3.4.2 Performance Profiles

A means of quantitatively expressing the effects of computation time on solution quality is required for effective metareasoning about anytime algorithms. Dean and Boddy, in their work on time-dependent planning, use a tool called a *performance profile* (PP) [64, 12, 13]. A PP of an anytime algorithm is a function that maps computation time to the expected

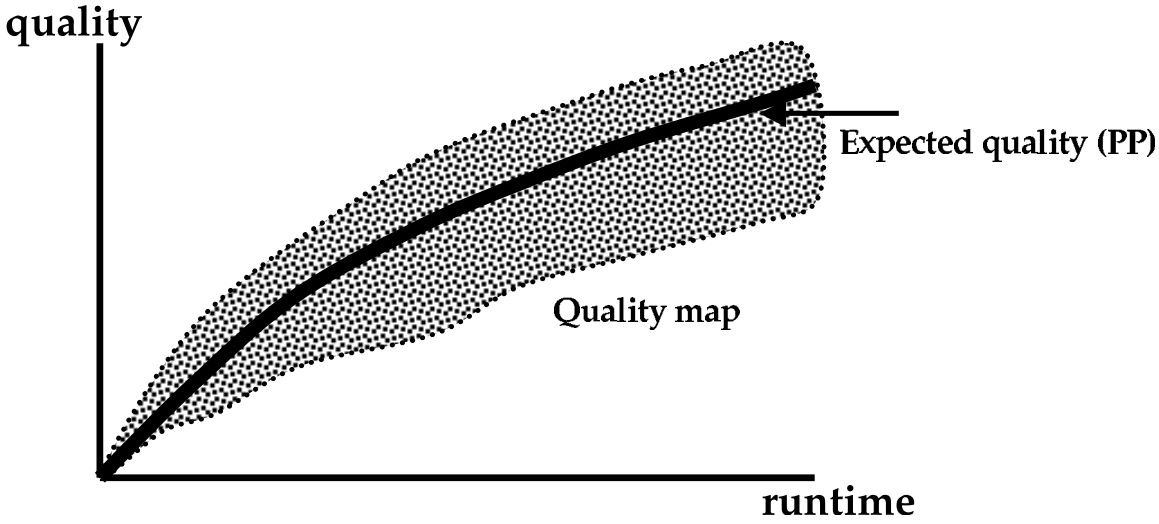


Figure 3.1: Example of a performance profile. Shown is an example of a quality map (i.e., the quality of results produced by the anytime algorithm for a set of random instances and random amounts of compute time). The PP is the expected quality as a function of time.

quality of the result. It is typically generated using a set of problem instances that are considered to be representative of the types of problem instances that will be encountered in the future. For each of the instances in this set, a random amount of computation time is chosen and a result is computed by the anytime algorithm for that amount of time. The quality of the results and the computation times to generate those results are used to compute the PP of the anytime algorithm. "Quality" in terms of a PP is typically something along the lines of percent improvement over the initial solution. Horvitz uses a similar construct to describe what he terms the *object-related value* of flexible computation [107, 108]. An example of a PP can be seen in Figure 3.1.

3.4.3 Deliberation-Scheduling

Dean and Boddy consider the problem of deliberation-scheduling for time-dependent planning problems [64, 12]. They define a planning problem as *time-dependent* if the time spent planning has a cost. *Deliberation-scheduling* is a procedure for allocating computational resources to a set of anytime algorithms based on expectations regarding their performance. For example, a time-dependent planning problem can have some set of events $C = c_1, c_2, \ldots, c_n$ that an agent must respond to. Each of these events can have a corresponding anytime algorithm for the purpose of choosing an action to take in response to the event. The job of the deliberation-scheduling algorithm is then to allocate computa-

tional resources among these anytime algorithms to maximize the agent's expected utility. Dean and Boddy give a polynomial-time algorithm for optimally solving this problem if the performance profiles of the anytime algorithms are monotonically increasing, continuous, and piece-wise differentiable functions [64]. The time spent by the deliberation-scheduling algorithm is assumed negligible. The deliberation-scheduling algorithm that they call *expectation-driven iterative refinement* is applied to a robot courier problem [12]. Horvitz, similarly, presents deliberation-scheduling algorithms [107, 108].

3.4.4 Compilation of Anytime Algorithms

A more generalized metareasoning problem concerns the use of anytime algorithms as the components of a larger system. Each component may take as input the output of one or more other anytime components. The deliberation-scheduling problem that arises from such a system is how to allocate computation time to the various components to maximize the expected output quality of the complete system. To deal with this problem, Zilberstein defines a *conditional performance profile* (CPP) of an anytime algorithm as a function mapping input quality and computation time to a probability distribution over the quality of results [207]. The anytime algorithm composition problem is NP-Complete [209, 207]. However, the *local compilation* approach of Zilberstein can lead to a linear time offline compilation (under certain assumptions) – though the result is not guaranteed globally optimal. In local compilation, the quality of the output of each component is optimized by considering only the performance profiles of its immediate sub-components. These sub-components are treated as elementary anytime algorithms in the local compilation. If a sub-component is not elementary, then its performance profile is determined recursively by local compilation [209, 207].

3.4.5 Anytime Computation and Negotiation

Larson and Sandholm consider problems in negotiation in which agents are assumed to be bounded rational [131, 132, 133]. Specifically, they consider that each agent may have an intractable individual problem and that there is a potential gain from pooling their resources to instead solve the joint problem comprised of their individual problems. This joint problem, too, is intractable. During deliberation, the agents can choose to compute on the solution to their own problem, to compute on the solution to another agent's problem, or to compute on the solution to the joint problem. Upon reaching the deliberation deadline,

the bargaining phase begins in which one agent makes an offer for how to divide the value of the joint solution and the other agent either accepts or rejects the offer (i.e., decides to either implement the proposed joint solution or the individual problem solutions). Larson and Sandholm consider the problem game-theoretically and define a *deliberation equilibrium* as the perfect Bayesian equilibrium of a game where deliberation actions are a part of each agent's strategy [131, 132]. They present algorithms for finding the equilibria for a number of situations such as whether or not the deadline is known, whether or not the proposer is known in advance, and whether the performance profiles are deterministic or stochastic.

This negotiation problem is somewhat related to the ideas of this thesis. In the negotiation problem, it must be decided which of three intractable problems to compute on, given performance profiles for anytime algorithms for the problems. In this thesis, we are instead interested in choosing among heuristics to use within a stochastic sampler (i.e., choosing among variations of an anytime algorithm) in solving a single intractable problem.

In solving the negotiation problem, Larson and Sandholm define a tree-structured performance profile that they call a *performance profile tree*. The tree structure allows them to condition on the results of computation so far. Each node in the tree represents solutions of a particular quality. For example, if quality is defined as percent improvement over an initial solution configuration, then the root of a performance profile tree will represent solutions of quality 0. Given that there are many paths through such a tree, there may be nodes at different levels that have the same quality as well as multiple nodes at the same level with the same quality. Edges in the tree represent the probability of reaching a future tree node (i.e., the child of the edge) given the current node (i.e., the parent of the edge). By multiplying the probabilities along paths rooted at the current node, one can compute the probability of reaching any given future node in the performance profile tree.

3.5 Summary

In this Chapter related work in metareasoning and metalevel search control has been discussed.

There has been much work in optimizing the control parameters of various search algorithms such as genetic algorithms at the metalevel. Much of this work has focused on optimization of these parameters a priori.

Next we discussed metareasoning approaches that learn models of search performance

online during execution for the purpose of search performance enhancement. For example, we saw that Sadeh's Expected Cost Improvement Distributions were used to identify and abandon less promising runs of simulated annealing. Ant colony optimization, adaptive probing, and hyperheuristics similarly all use weights of one type or another which they adjust according to search performance to improve search control decision-making policies. Boyan's STAGE algorithm learns an alternative evaluation function for use by the search in place of the original objective function. Of these, the approach of this thesis is most closely related to hyperheuristics in that both can be used to adapt control policies for effective choice of heuristics within a search framework.

The final part of the Chapter gave an overview of some related work from the area of anytime computation. Most relevant is the concept of a performance profile, and its variations such as Larson and Sandholm's performance profile trees. The AQDF that will be presented later in Chapter 6 can be viewed, as is later discussed, as a detailed model of a particular time slice of a PP. The AQDF also differs from a PP in other ways including being problem instance dependent while a PP is problem class dependent.

Chapter 4

VBSS: Value Biased Stochastic Sampling

4.1 Overview

In this Chapter, we present our novel approach to stochastic sampling. Acknowledging the fact that our heuristic may be more or less discriminating from context to context, we define *value-biased stochastic sampling* (VBSS) in Section 4.2. Our method for computing the stochastic sampling decisions in VBSS is inspired by a computational model of wasp behavior (recall Section 3.3.6). This novel stochastic sampling algorithm, known as WHISTLING (Wasp beHavior Inspired STochastic sampLING), is presented in Section 4.3. Section 4.4 provides a proof of the equivalence of the wasp-inspired dominance tournament to the roulette wheel decisions of VBSS. Section 4.5 discusses some issues associated with choosing an appropriate bias function for a given problem and for the choice of heuristic function. A summary concludes the Chapter in Section 4.6.

4.2 Value-Biased Stochastic Sampling

A fundamental flaw of the HBSS framework of Bresina is that it ignores the discriminatory power inherent in the heuristic. Its stochastic decisions rank-order the choices from the choice with the highest value of the heuristic to the choice with the lowest value of the heuristic. It then chooses choice c_i according to a roulette wheel with probability:

$$P(c_i) = \frac{\text{bias}(\text{rank}(c_i, \text{heuristic}))}{\sum_j \text{bias}(\text{rank}(c_j, \text{heuristic}))} \tag{4.1}$$

	Context 1	Context 2
	Choice 1 (H1 = 10) / Choice 2 (H2 = 11)	Choice 1 (H1 = 10) / Choice 2 (H2 = 100)
HBSS with bias(p) = 1/p	Rank(c1) = 2 Rank(c2) = 1 Bias(Rank(c1)) = 0.5 Bias(Rank(c2)) = 1 P(choosing c2) = 1/(1+0.5) = 0.67	Rank(c1) = 2 Rank(c2) = 1 Bias(Rank(c1)) = 0.5 Bias(Rank(c2)) = 1 P(choosing c2) = 1/(1+0.5) = 0.67
VBSS with bias(p) = p	Bias(H1) = 10 Bias(H2) = 11 P(choosing c2) = 11/(11+10) = 0.52	Bias(H1) = 10 Bias(H2) = 100 P(choosing c2) = 100/(100+10) = 0.91

Figure 4.1: Two example decision contexts – one more discriminating than the other.

where bias() is a bias function and rank() returns the rank assigned to a given choice when the set of choices is sorted by the given heuristic. Other than to sort the choices, the heuristic values are not used and thus not utilized to their full potential. There are, however, occasions when a rank-based approach may be more beneficial. For example, genetic algorithms sometimes benefit from a rank-based selection strategy [91].

In order to fully utilize the discriminatory power of the heuristic inherent in the heuristic values, we now define what we call value-biased stochastic sampling (VBSS). In VBSS, decisions are again made in a manner analogous to a roulette wheel, but choice c_i is now made with probability:

$$P(c_i) = \frac{\text{bias}(\text{heuristic}(c_i))}{\sum_j \text{bias}(\text{heuristic}(c_j))} \tag{4.2}$$

VBSS is shown in Algorithm 4.1.

Consider one decision context in which you have two choices that are preferred almost equivalently by the heuristic (i.e., they have almost, but not quite, equal heuristic values). Now consider a second decision context in which you have two choices, but where one of the choices is much more heavily preferred (i.e., it has a much higher heuristic value than the other choice). In HBSS, both of these decision contexts are regarded equivalently. That is, the choice with the higher heuristic value gets ranked first, the other choice gets ranked second, the bias function is applied, and the decision is made. The first ranked choice has the same probability of being made in both contexts. Whereas, in VBSS, since the heuristic

Algorithm 4.1: Value Biased Stochastic Sampling (VBSS)
Input: Number of iterations I; a "heuristic" function; a "bias" function; an "objective" function; and a search-tree T.
Output: A solution S.
VBSS(I, heuristic, bias, objective, T)

```
(1)   bestsofar ← solution S obtained if "heuristic" is followed from T
(2)   repeat I times
(3)     S ← root search-node of T
(4)     while S is a decision node of T
(5)       foreach choice C from S
(6)         score[C] ← heuristic(C, S)
(7)       totalweight ← 0
(8)       foreach choice C from S
(9)         weight[C] ← bias(score[C])
(10)        totalweight ← totalweight + weight[C]
(11)      foreach choice C from S
(12)        prob[C] ← weight[C] / totalweight
(13)      select randomly the choice C biased according to prob[C]
(14)        S ← Successor(S, C)
(15)    if objective(S) is superior to objective(bestsofar)
(16)      bestsofar ← S
(17)  return bestsofar
```

values are used explicitly in the stochastic decisions rather than a rank imposed by them, the first ranked choice in the more discriminating context is chosen with a much higher probability than it is in the less discriminating context. This can be seen in Figure 4.1 where we see two example decision contexts: one in which the heuristic values are almost the same and a second in which the heuristic values are drastically different. In this second decision context, using a value-biased approach, there is a much higher probability of choosing the heuristic preferred choice as compared to the other decision context; while the rank-biased approach treats both decision contexts in the exact same way.

4.3 WHISTLING

Our stochastic search framework[1] derives from the naturally-inspired computational model of the self-organization that takes place within a colony of wasps [196, 194]. In nature,

[1]The WHISTLING algorithm is a stochastic search extension to our *scheduling wasps* [47, 45]. The WHISTLING algorithm was presented originally at CP-2002 [48].

a hierarchical social order among the wasps of the colony is formed through interactions among individual wasps of the colony. This emergent social order is a succession (or prioritization) of wasps from the most dominant to the least dominant. In the model of Theraulaz *et al.*, the results of these interactions are determined stochastically based on the *force* variables of the wasps involved. The probability of wasp 1 winning a dominance contest against wasp 2 is defined based on the force variables, F_1 and F_2, of the wasps as:

$$P(F_1, F_2) = \frac{F_1^2}{F_1^2 + F_2^2} \tag{4.3}$$

After such an interaction, the value of the force variable of the winner is increased and the value of the force variable of the loser is decreased. Over time, through many such interactions, a hierarchical social order is formed from the most dominant down to the least dominant wasp.

We can map the above model to the problem of randomizing heuristic choices by associating a wasp as a proxy for each possible choice in a decision context, and defining the force of a given wasp to be a function of the heuristic value assigned to its corresponding choice.

More precisely, we define the force of a wasp as:

$$F_w = \text{bias}(\text{heuristic}(\text{Choice}_w)) + D \tag{4.4}$$

where Choice_w is the choice represented by wasp w, $\text{heuristic}()$ is a function that returns the heuristic value of the given choice, and $\text{bias}()$ is a bias function (as in the HBSS framework). D is a variable (non-negative) that is initialized to 0 and varies according to the results of dominance contests during a run of the algorithm (see below). It mimics the fluctuations that occur in the force variable values as dominance contests between real wasps are won and lost in nature.

Given that our definition of F_w includes an explicit bias factor, we choose to simplify the stochastic rule for choosing the winner of a dominance competition (Equation 4.3) as follows:

$$P(F_1, F_2) = \frac{F_1}{F_1 + F_2} \tag{4.5}$$

By coupling this new definition in Equation 4.5 with an appropriate bias function in the force definition of Equation 4.4 we can express the original rule given in Equation 4.3. However, like HBSS, our reformulation allows the expression of a range of bias functions.

Algorithm 4.2: Wasp beHavior Inspired STochastic sampLING (WHISTLING)
Input: Number of iterations I; a "heuristic" function; a "bias" function; an "objective" function; and a search-tree T.
Output: A solution S.

```
WHISTLING(I, heuristic, bias, objective, T)
(1)   bestsofar ← solution S obtained if "heuristic" is followed from T
(2)   evaluate[bestsofar] ← objective(bestsofar)
(3)   repeat I times
(4)     S ← root search-node of T
(5)     while S is a decision node of T
(6)       foreach choice C from S
(7)         force[C] ← bias(heuristic(C, S))
(8)       WinnerSoFar ← arbitrary choice C from S
(9)       Challengers ← the set of choices from S − WinnerSoFar
(10)      foreach choice C in the set Challengers
(11)        with probability P(force[C], force[WinnerSoFar]) (see Eq. 4.5)
(12)          force[C] ← force[C] + force[WinnerSoFar]
(13)          WinnerSoFar ← C
(14)        otherwise
(15)          force[WinnerSoFar] ← force[WinnerSoFar] + force[C]
(16)      S ← Successor(S, WinnerSoFar)
(17)    evaluate[S] ← objective(S)
(18)    if evaluate[S] is superior to evaluate[bestsofar]
(19)      bestsofar ← S
(20)  return bestsofar
```

Bonabeau *et al.* generalize their definition as $P(F_1, F_2) = \frac{F_1^a}{F_1^a + F_2^a}$ where a is a parameter [14]. However, their motivation is somewhat different. They discuss how this formula models the behavior of real insect societies and how different values for a may more closely model a particular society's behavior. Furthermore, force is a variable in their model that adapts according to wins and losses of dominance contests and is not explicitly defined functionally as we do here. Our definition allows for the spectrum of polynomials as does Bonabeau *et al.*'s; but it also allows expression of many others (e.g., exponentials and logarithms) that cannot be expressed by Bonabeau *et al.*'s generalization.

Given the above definitions of $P(F_1, F_2)$ and F_w, we can make the sampling algorithm concrete by defining a specific competition structure. In what follows, we assume the following style of tournament. An initial wasp (choice) is selected arbitrarily and proceeds to engage in successive competitions with all other wasps (other choices). The initial wasp continues to compete as long as it wins. If it is defeated, the new winner takes its place

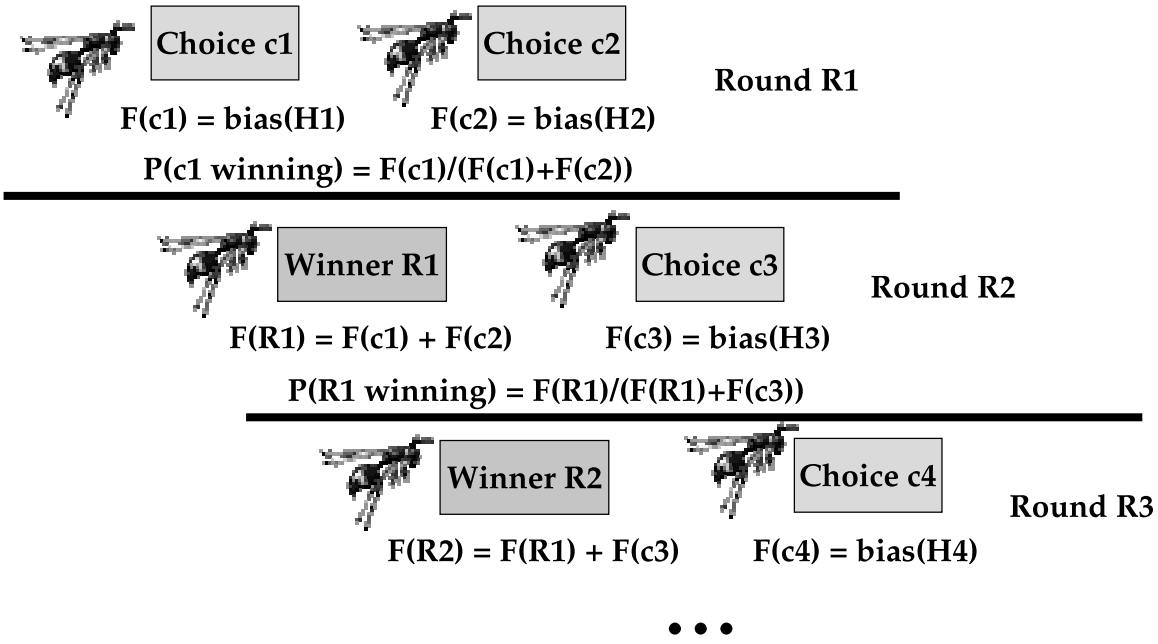


Figure 4.2: A tournament of wasp dominance contests.

and proceeds to face the remaining candidates. Upon winning a competition, the winning wasp's D variable is updated to accumulate the force variable value of the loser, and the loser drops out. The wasp (choice) remaining at the end of the tournament is returned as the final selection. This tournament structure is illustrated in Figure 4.2. It can be shown that this computation is equivalent to selecting according to a standard roulette wheel decision, with each possible choice Choice_w taking a chunk of the wheel proportional to $\text{bias}(\text{heuristic}(\text{Choice}_w))$. The advantage of this method of computing a roulette wheel decision over the usual one is that a single pass through the set of choices is required. The alternative would be to make one initial pass to compute $\sum_j \text{bias}(\text{heuristic}(\text{Choice}_j))$ followed by a second pass to choose w with probability $\frac{\text{bias}(\text{heuristic}(\text{Choice}_w))}{\sum_j \text{bias}(\text{heuristic}(\text{Choice}_j))}$. A proof of the equivalence of the tournament of dominance contests to a roulette wheel decision is provided in Section 4.4.

The WHISTLING algorithm is presented in Algorithm 4.2. Line 7 of the algorithm is the initial definition of the force variables of the wasps with $D = 0$ (see Equation 4.4). Line 12 and line 15 represent the increase in the value of D in Equation 4.4 for the winning wasp. The dominance contests (see Equation 4.5) take place in line 11 of the algorithm.

Computationally, the WHISTLING algorithm (as well as the general VBSS algorithm) selects a choice in $O(C)$ time where there are C choices from the current search-node. Since it must do this C times (in a sequencing problem), the core sampling procedure has

an overall algorithmic complexity of $O(C^2)$. In fact, the complexity of a corresponding deterministic dispatch scheduling procedure is also $O(C^2)$. Hence, WHISTLING (or VBSS) adds only a constant factor to the computational time required for strict deterministic application of the heuristic. If we compare this complexity to that of the HBSS algorithm of Algorithm 2.2, we can see that WHISTLING can be asymptotically more efficient. Since HBSS biases its stochastic decisions according to a rank-ordering of the choices, the obvious implementation sorts the choices according to their heuristic values each time a search decision is to be made – an $O(C \log C)$ operation. And with C decisions to be made (in a sequencing problem), the algorithmic complexity of HBSS is $O(C^2 \log C)$.[2] Later, we will see how WHISTLING and HBSS compare experimentally in a particular scheduling domain.

4.4 Proof: Dominance Tournament = Roulette Wheel Decision

Dominance Tournament = Roulette Wheel Decision: The structure of the tournament of dominance contests within the WHISTLING algorithm is such that Choice_w is chosen with probability $\frac{\text{bias}(\text{heuristic}(\text{Choice}_w))}{\sum_{j=1}^{k} \text{bias}(\text{heuristic}(\text{Choice}_j))}$. That is, the tournament is such that each Choice_w is chosen according to a roulette wheel decision where each Choice_w takes a chunk of the wheel proportional to $\text{bias}(\text{heuristic}(\text{Choice}_w))$.

Proof (by induction): In this proof, $F_w = \text{bias}(\text{heuristic}(\text{Choice}_w))$ is the initial value of the force of Choice_w. Further F'_w is the force of Choice_w including the accumulation of the force variables of other choices which it has defeated in the tournament.

[2] It has been pointed out that the worst-case complexity of choosing the i-th largest element from an unsorted list is $O(n)$. Thus, in theory, this stochastic selection operation can also be done in linear time. Under an assumption that the n choices have ranks 1 through n, we can select the winning rank without looking at the actual elements themselves. And then use the winning rank and the linear time selection algorithm to make the decision. However, the linear time algorithm for the selection operation itself is likely to be more of a theoretical interest than of practical interest. It has a large constant factor that results in the $O(n \log n)$ sort-first-then-select algorithm dominating for all but very large problem instances [52]. A second problem is that the assumption that the n elements have ranks 1 through n does not hold if more than one element may have the same heuristic value and thus the same rank. It may be possible (if one is clever enough) to devise a linear time method for finding all ties in heuristic value. However, such a method is certainly non-obvious and probably also non-trivial, adding to the already significant constant factor. Thus, although it may be possible to compute the HBSS decisions in linear time, it is probably not desirable to actually do so unless your search-space has a particularly large branching factor.

Base Case: Dominance tournament of two choices.

- Choice$_1$ wins with probability $\frac{F_1}{F_1+F_2}$. If it wins, its new force is $F_1' = F_1 + F_2$.
- Choice$_2$ wins with probability $\frac{F_2}{F_1+F_2}$. If it wins, its new force is $F_2' = F_1 + F_2$.
- Therefore, for the base case of two choices we have the equivalent of a roulette wheel decision.

Inductive Step:

- Assume that for the case of a tournament of dominance contests of $k-1$ choices that this tournament is equivalent to a roulette wheel decision.
- Further assume that upon the completion of this tournament that the winning choice Choice$_w$ has a force value of $F_w' = \sum_{j=1}^{k-1} F_j$.
- Now consider the addition of a k-th choice Choice$_k$ at the end of the list of choices with an initial force F_k. The winner Choice$_w$ of the sub-tournament consisting of the first $k-1$ choices would then compete in a dominance contest against Choice$_k$.
- Choice$_k$ wins this dominance contest and thus the tournament of contests among the k choices with probability $\frac{F_k}{F_k+F_w'} = \frac{F_k}{F_k+\sum_{j=1}^{k-1} F_j} = \frac{F_k}{\sum_{j=1}^{k} F_j}$. If it wins, its new force is $F_k' = F_k + F_w' = \sum_{j=1}^{k} F_j$.
- Choice$_w$ wins this dominance contest against Choice$_k$ with probability $\frac{F_w'}{F_k+F_w'}$. Further, since it had a probability of $\frac{F_w}{F_w'}$ of winning the sub-tournament of the first $k-1$ choices, it therefore wins the overall tournament of k choices with probability $\frac{F_w}{F_w'}\frac{F_w'}{F_k+F_w'} = \frac{F_w}{\sum_{j=1}^{k} F_j}$. Its new force if it wins is $F_w'' = F_k + F_w' = \sum_{j=1}^{k} F_j$.
- Therefore, for the case of a tournament of k choices, we have the equivalent of a roulette wheel decision.

4.5 Choosing a Bias Function

With VBSS (as well as with HBSS), it is necessary to specify a bias function that is applied to the heuristic values (or in the case of HBSS to the ranks). This bias function can be

viewed as a system parameter that requires some amount of tuning. There are a couple of issues to consider when selecting a bias function for VBSS:[3]

- First, stronger and/or knowledge-rich heuristics should probably be used in conjunction with stronger bias functions, such as polynomials of high degree. The rationale is that if a large amount of effort and expert knowledge went into the heuristic's design, then the search might benefit by following its advice more often. Similarly, weaker and/or knowledge-poor heuristics should probably be used with weaker bias functions, such as low degree polynomials.

- Second, heuristics that generally give values in some very small range, such as very small positive real values less than one, might require a strong bias function to spread out the values used in the roulette wheel decisions. Heuristics that give values in a wider range might already be sufficiently spread to require only a weaker bias function.

With these issues in mind, little (if any) fine-tuning of the bias function choice is generally needed. It is generally sufficient to try out a few in some range decided upon by considering the types of values given by the heuristic and the strength of the heuristic itself. Alternatively, one might employ a more computationally heavy approach to tuning the bias function such as with a meta-optimization approach as is sometimes done to tune genetic algorithms. In the experiments of this thesis, no such computational method has been employed. In most cases, a handful of bias functions have been tried on a small set of problem instances for a small number of iterations. The bias functions tried in these preliminary tuning runs are chosen according to the rationale presented above, and in most cases give nearly numerically indistinguishable results.

4.6 Summary

In this Chapter, VBSS was defined and we saw how by using the heuristic values themselves within our stochastic decisions, rather than a rank-order imposed by the heuristic values, that we are much better able to use the full discriminatory power inherent in our heuristic function. Furthermore, since we no longer require ranking of the choices, we gain computational efficiency as an added bonus – in the best case, eliminating the $O(n \log n)$ ranking step of HBSS; and in the worst case, simplifying and streamlining the multiplier

[3]These issues are not necessarily the same as those involved with selecting a bias function for HBSS.

of the linear time operation (if the HBSS decision can be computed in linear time). We further defined an efficient approach to computing roulette wheel decisions inspired by a computational model of wasp social hierarchy formation and the resulting novel stochastic sampling algorithm, which we call WHISTLING, is defined.

Chapter 5

Application: Weighted Tardiness Scheduling with Sequence-Dependent Setups

5.1 Overview

To demonstrate and experimentally evaluate the VBSS framework, we turn to the domain of factory scheduling and the fairly broad body of work in the area of dispatch scheduling heuristics (c.f., [145]). Specifically, in this Chapter, we explore the possibility of amplifying dispatch scheduling heuristic performance through the addition of a search component – namely our VBSS search framework. The scheduling problem tackled in this Chapter is that of the weighted tardiness scheduling problem with sequence-dependent setups and is formalized in Section 5.2. Existing methods for the problem from the literature are presented in Section 5.3. The search space is discussed in Section 5.4. The description of the specific set of problem instances and how they were generated can be found in Section 5.5. Performance criteria that are used in the experimental comparisons of this Chapter are defined in Section 5.6. In Section 5.7, we compare the performance of HBSS and VBSS and demonstrate the power of value-biasing over rank-biasing. We go on to illustrate the effective use of value-biasing as a means of generating initial configurations within a multi-start local hill-climbing framework in Section 5.8. The difficulties faced by related systematic search procedures such as LDS and DDS in this domain are considered in the comparison of Section 5.9. Finally, a summary concludes the Chapter in Section 5.10.

5.2 Problem Formalization

The Weighted Tardiness Scheduling Problem with Sequence-Dependent Setups is a sequencing problem encountered in a number of real-world problem applications (e.g., turbine component manufacturing [36], the packaging industry [1], among others [145]). Specifically, we are given a set of jobs $J = \{j_0, j_1, \ldots, j_N\}$. Each of the N jobs j has a weight w_j, duedate d_j, and process time p_j. Furthermore, $s_{i,j}$ is defined as the amount of setup time required immediately prior to the start of processing for job j if it is to follow job i on the machine. It is not necessarily the case that $s_{i,j} = s_{j,i}$. The 0-th job is a non-physical job representing the starting point of the problem ($p_0 = 0$, $d_0 = 0$, $s_{i,0} = 0$, $w_0 = 0$). Its only purpose is to allow for the specification of the setup time of each of the jobs if sequenced in position 1. The sequence-dependent nature of the setup times is a primary source of problem difficulty. The particular version of the problem that we concern ourselves with here is the case where the N jobs must be sequenced on a single machine and where preemption of a job during setup or processing is not permitted. Furthermore, if a machine is processing or setting up, then it cannot do anything else until that operation is completed.

The objective of this problem is to sequence the set of jobs J on a machine to minimize the total weighted tardiness:

$$T = \sum_{j \in J} w_j T_j = \sum_{j \in J} w_j \max(c_j - d_j, 0), \tag{5.1}$$

where T_j is the tardiness of job j; and c_j, d_j is the completion time and duedate of job j. The completion time of job j is equal to the sum of the process times and setup times of all jobs that come before it in the sequence plus the setup time and process time of job j itself. Specifically, let $\pi(j)$ be the position in the sequence of job j. We can now define c_j as:

$$c_j = \sum_{i,k \in J, \pi(i) <= \pi(j), \pi(i) = \pi(k)+1} p_i + s_{k,i} \tag{5.2}$$

5.3 State-of-the-(Ad-Hoc)-Art Solution Methods

Most of the work that has been done on the weighted tardiness scheduling problem has been for versions of the problem without sequence-dependent setup times. For the setup-free problem, there exist libraries of benchmark instances and a large number of algorithms with which to compare new approaches.

Unfortunately, although sequence-dependent setups commonly appear in real-world scheduling problems (e.g., [1, 144, 143]), they are often ignored during the development of algorithms and solution procedures. It is common practice to assume that setup time can be reintroduced to the problem after solving the setup-free version. For example, the dispatch heuristics for the sequence-dependent setup case that we will discuss below are variations of a well-known and state-of-the-art heuristic for the setup-free version of the problem. Terms including setup time have been added to the heuristic in a seemingly ad hoc manner. Yet, at the same time, these ad hoc heuristic procedures (as well as local improvement search algorithms applied to their results) seem about the best one can do for the sequence-dependent setup version of the problem given the current state-of-the-art. For example, complete, guaranteed optimal algorithms are currently limited to solving problems of at most 20-25 jobs in size [181], necessitating the use of heuristic-guided search, or meta-heuristic approaches for larger problems.

In the following Subsections, we overview the existing dispatch policies for the sequence-dependent setup, weighted tardiness scheduling problem as well as a local hill-climber designed specifically with the intentions of improving upon the solutions produced by one of these heuristics. Later in this Chapter, we use VBSS to search a stochastic neighborhood of the heuristic's prescribed solution path using VBSS as an alternative to this hill-climber. We also define a few systematic heuristic-guided alternatives that also use the dispatch policies of this Section for guidance.

5.3.1 Dispatch Scheduling Policies

In dynamic factory environments, dispatch scheduling heuristics (c.f., [145]) provide a practical, robust basis for managing execution in environments that are often plagued by a large number of dynamic characteristics that can be difficult to accurately model (e.g., McKay provides a model of such a complex dynamic factory environment [140]). Scheduling decisions such as which job to assign to a machine next are made in an online manner only as needed, based on the current state of the factory. Dispatch heuristics make use of information about jobs such as expected processing time, setup time, due date, priority, etc., and are typically designed to optimize a given performance objective. Their virtue is their simplicity and insensitivity to environmental dynamics and for these reasons they are commonly employed. At the same time, the localized and myopic nature by which decisions are made under such schemes make them inherently susceptible to sub-optimal decision-making and they can even exhibit formally chaotic tendencies [116, 8].

For the problem domain of weighted tardiness scheduling with sequence-dependent setups, the *Apparent Tardiness Cost with Setups (ATCS)* dispatch heuristic [134] is perhaps the strongest heuristic available. ATCS builds on earlier research into the weighted tardiness problem and is arguably the current best performing dispatch policy for this class of scheduling problem. ATCS is defined as follows:

$$\text{ATCS}_j(t,l) = \frac{w_j}{p_j} \exp\left(-\frac{\max\left(d_j - p_j - t, 0\right)}{k_1 \bar{p}} - \frac{s_{l,j}}{k_2 \bar{s}}\right), \tag{5.3}$$

where t is the current time (or the sum of the process and setup times of the already sequenced jobs); l is the index of the job just completed (or the last job added to the schedule); $\bar{p}$ is the average processing time of all jobs; $\bar{s}$ is the average setup time. The k_1 and k_2 are parameters for tuning the heuristic. Lee *et al.* defines values for these parameters according to problem instance characteristics [134], specifically:

$$k_1 = \begin{cases} 4.5 + R & \text{if } R \leq 0.5 \\ 6.0 - 2R & \text{otherwise} \end{cases}, \tag{5.4}$$

and

$$k_2 = \frac{\tau}{2\sqrt{\eta}}. \tag{5.5}$$

R is the due-date range factor; τ is the due-date tightness factor; and η is the setup time severity factor. These are defined in Section 5.5. The next job j added to the schedule using the ATCS heuristic is simply:

$$j = \arg\max_j \text{ATCS}_j(t,l). \tag{5.6}$$

Before the ATCS heuristic's presentation by Lee *et al.*, the best performing dispatch heuristic for this problem had been that of Raman *et al.* [161]:

$$\text{Raman}_j(t,l) = \frac{w_j}{p_j + s_{l,j}} \exp\left(-\frac{\max\left(d_j - p_j - s_{l,j} - t, 0\right)}{k\bar{p}}\right) \tag{5.7}$$

where the next job j is chosen according to:

$$j = \arg\max_j \text{Raman}_j(t,l). \tag{5.8}$$

The k is a parameter that again requires tuning. In their comparison of ATCS and Raman,

Lee *et al.* suggest setting $k = 5.5 - \tau - R + \eta$.

One thing that should be noted is that both Raman *et al.*'s dispatch policy as well as the ATCS dispatch policy are modifications of the R&M dispatch policy [160] that was developed for the variation of the problem without sequence-dependent setup constraints:

$$\text{R\&M}_j(t) = \frac{w_j}{p_j} \exp(-\frac{\max(d_j - p_j - t, 0)}{k\bar{p}}). \tag{5.9}$$

This heuristic will be discussed again in a later chapter that considers the problem without sequence-dependent setup times.

5.3.2 Dispatch Policy as Starting Configuration for Local Search

In Lee *et al.*'s original paper describing the ATCS heuristic, they also present a local hill-climber designed specifically with the intentions of being applied to the solution that results directly from the deterministic application of the dispatch policy ATCS [134]. The hill-climber (that we will refer to later in our experimental study as "LEE") assumes that the starting configuration is a good one (i.e., in the vicinity of the optimal or a near-optimal solution). Given this assumption, Lee *et al.*'s hill-climber uses a fairly small operator set. The operator set includes two types of local moves:

1. Swaps: First, choose the job j' that adds the most to the total weighted tardiness objective (i.e., $j' = \arg\max_j w_j \max(c_j - d_j, 0)$). Next consider swapping job j' with each of the 20 nearest jobs in the current sequence.

2. Insertions: First, choose the job j' that adds the most to the total weighted tardiness objective (i.e., $j' = \arg\max_j w_j \max(c_j - d_j, 0)$). Next consider the removal of job j' followed by the insertion of job j' before each of the 20 nearest jobs in the current sequence.

A couple things to note about this operator set is that it does not consider moving jobs large distances and that one of the jobs is fixed to be the job which adds the most to the objective function value. These fall out from the assumption of having a good initial solution. It is assumed that jobs are near their optimal location in the sequence (i.e., moving jobs large distances in the sequence is not considered) and it is assumed that most jobs do not need to be moved at all (i.e., fixing one of the jobs to be the one that adds the most to the objective function). Given this operator set, the hill-climber at each step makes the move that gives

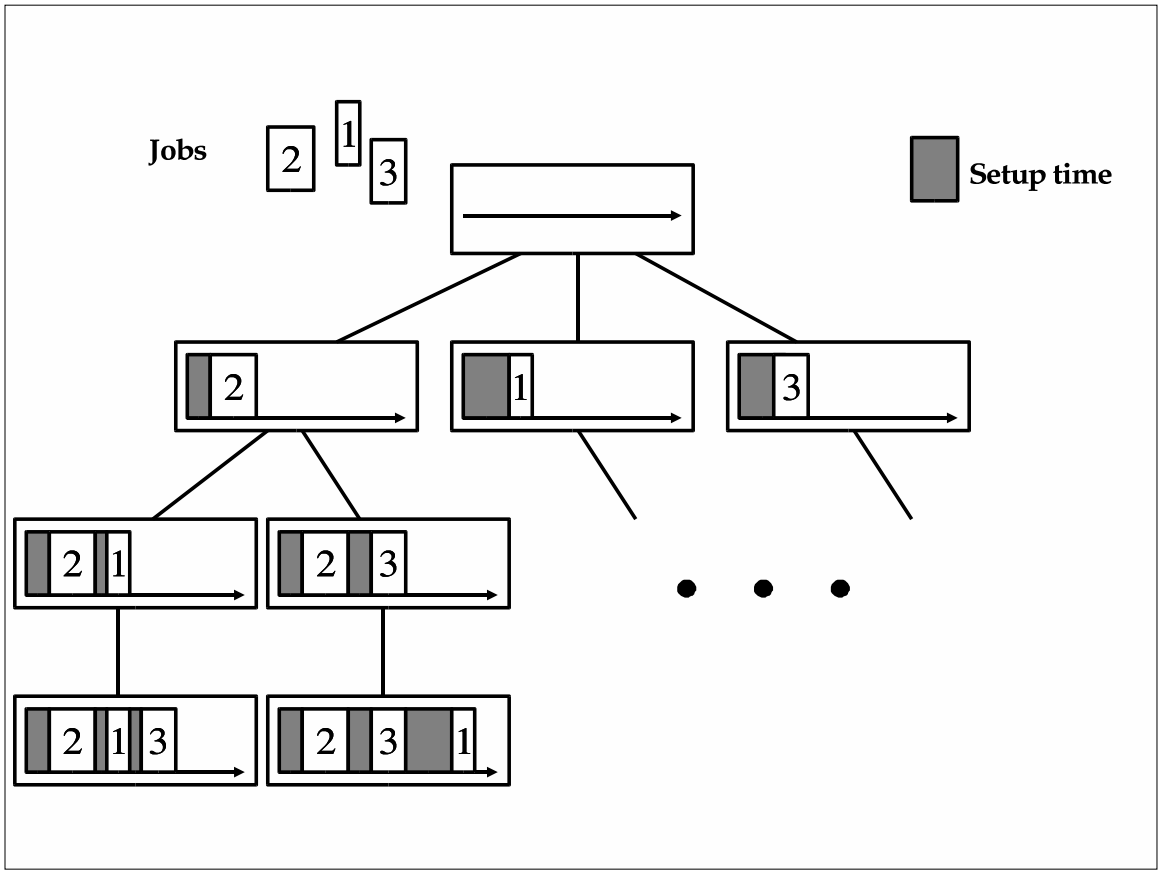


Figure 5.1: Illustration of the search space for the weighted tardiness scheduling problem. Particularly note the sequence-dependent size of the setup times that are indicated by the size of the gray boxes in the figure.

the greatest improvement in the objective function's evaluation and ends the search when no further improvement can be made given this operator set. The ATCS dispatch policy coupled with the use of this hill-climber is currently in operation within the scheduling system of a number of factories in the packaging industry [1].

5.4 The Search-Space

The search-space that we will explore in the experiments in the remainder of this Chapter can be viewed as a tree. The root node of this tree represents an empty sequence (or empty schedule). Each search-node has a child for each of the jobs not yet in the sequence. Such a child of a search-node represents adding the given job to the end of the sequence. If a problem instance has N jobs, then the root node has N children (i.e., each represents

beginning the sequence with one of the N jobs). Each of the children of the root node has $N-1$ children, and so forth and so on. Each leaf node of the tree represents one of the $N!$ possible sequences of jobs. Every leaf node represents a feasible solution and thus every path from the root is guaranteed to find a feasible solution. An illustration of this search-space is shown in Figure 5.1. Particularly note that the size of the setup times (indicated by the gray boxes) is dependent upon the sequence of jobs.

5.5 The Problem Set

The problem instances that we consider are generated according to a procedure described by Lee *et al.* [134] and used in the analysis of their dispatch scheduling policy ATCS. Unfortunately, they did not make their actual problem set available so we are unable to compare to the exact problem instances that were used in their study. Our problem set is however generated as they prescribe. We have made these problem instances available on the Internet. Appendix A provides the location, details of the file format, and best known solutions.

Each problem instance is characterized by three parameters: the due-date tightness factor τ; the due-date range factor R; and the setup time severity factor η. These parameters are defined as follows:

$$\tau = 1 - \frac{\bar{d}}{C_{\max}} \tag{5.10}$$

$$R = \frac{d_{\max} - d_{\min}}{C_{\max}} \tag{5.11}$$

$$\eta = \frac{\bar{s}}{\bar{p}} \tag{5.12}$$

where $\bar{d}$, $\bar{p}$, and $\bar{s}$ are the average duedate, average process time, and average setup time, $d_{\max}$, $d_{\min}$ are the maximum and minimum duedates, and $C_{\max}$ is the makespan (or completion time of the last job). Given that the makespan depends on the sequence and the $s_{i,j}$, the estimator suggested by Lee *et al.* is used: $\tilde{C}_{\max} = n(\bar{p} + \beta\bar{s})$ where n is the number of jobs in the problem instance. We, specifically, consider problem sets characterized by the following parameter values: $\tau = \{0.3, 0.6, 0.9\}$; $R = \{0.25, 0.75\}$; and $\eta = \{0.25, 0.75\}$. For each of the twelve combinations of parameter values, we generate 10 problem instances with 60 jobs each. Generally speaking, these 12 problem sets cover a spectrum from loosely to tightly constrained problem instances. The processing times are uniformly distributed

over the interval $[50, 150]$, with $\bar{p} = 100$. The mean setup time $\bar{s}$ is then determined from η and the setup times are uniformly distributed in the interval $[0, 2\bar{s}]$. The duedate of a job is uniformly distributed over $[\bar{d}(1-R), \bar{d}]$ with probability τ and uniformly distributed over $[\bar{d}, \bar{d} + (C_{\max} - \bar{d})R]$ with probability $1 - \tau$. The weights of the jobs are distributed uniformly over $[0, 10]$.

5.6 Performance Criteria

In the experimental results that follow, we use the following performance criteria:

- Average Percent Improvement (API) over the solution given by use of the deterministic heuristic policy. Percent improvement is defined as: $100 * \frac{h-a}{h}$ where h is the objective value for the problem instance given by the deterministic use of the dispatch heuristic ATCS, and a is the objective value given by the search algorithm. This is then averaged across all 120 problem instances to get API. For all stochastic algorithms considered, the API given is also averaged across the results of 10 independent runs on each problem instance. In parentheses, after the average of the 10 runs, is the API where the best run for each problem instance is used.

  The ATCS dispatch policy is currently the best known for this problem. Lee *et al.*'s hill-climber which uses ATCS to determine the initial starting configuration is currently the best known heuristic search algorithm for this problem and Lee *et al.* report their results as the API over ATCS. Ideally, we would like to have a set of problems for which the optimal solutions are known, for which we have bounds, or for which we at least have the current best known solutions. Unfortunately, for the sequence-dependent setup problem, there is not such a benchmark set of problems in existence currently. For this reason, we present comparative results using the criteria set forth in the original discussion of the current best known algorithm for the problem. In a later Chapter where we discuss the setup-free version of the problem, we do use a benchmark library of problems for which optimal solutions are known for a number of instances, best known are given for the others, and for which several algorithmic results are available for comparative purposes.

- Average CPU Time (TIME) in seconds on a Sun Ultra 10, 300MHz.

- The average number of search nodes generated (Gen) by the search.

Table 5.1: HBSS Preliminary Results: A sampling of the results from applying HBSS with various bias functions and 100 iterations. This is a snapshot of the data that was used to choose a bias function for the HBSS algorithm for further experiments.

p	API
1	0.0 (0.0)
2	4.6 (8.3)
3	18.3 (22.1)
4	21.5 (24.0)
5	21.6 (23.9)
6	20.8 (23.2)
7	19.7 (22.2)
8	17.9 (21.2)

- The average number of search nodes visited (Visits) by the search.
- The average number of solution nodes considered (Sols) by the search.

5.7 To Value-Bias or to Rank-Bias?

To value-bias or to rank-bias: that is the question. Whether tis nobler in the search to rank-order choices ignoring utterly the context of decisions, or to take arms against a sea of troubles, and by value-biasing end them?

In Chapter 4, we saw an example of two decision contexts in which a value-bias approach made better use of the discriminatory power inherent in the heuristic values as compared to a rank-bias approach to stochastic sampling. Here now, we compare VBSS and HBSS experimentally on the weighted tardiness scheduling problem with sequence-dependent setups. The goal is to see how much can be gained by using the heuristic values for bias rather than biasing by rank-order alone.

In order to properly compare HBSS and VBSS on the problem set, we begin by applying each of them with a wide range of bias functions and the ATCS dispatch heuristic presented earlier. Table 5.1 shows the results of the HBSS algorithm for polynomial bias functions of degrees 1 through 8 (bias $= \frac{1}{\text{rank}^p}$ where p is the degree of the polynomial). Similarly, Table 5.2 shows the results of VBSS with polynomial bias functions (bias $= \text{value}^p$ where p is the degree of the polynomial and value is the heuristic value). Many more bias functions were also considered but for brevity have been excluded from these tables. The values in the tables are averages across all problem instances and for 10 runs each. Values in

Table 5.2: VBSS Preliminary Results: A sampling of the results from applying VBSS with various bias functions and 100 iterations. This is a snapshot of the data that was used to choose a bias function for the VBSS algorithm for further experiments.

p	API
1	7.3 (9.7)
2	16.7 (19.7)
3	20.0 (22.9)
4	21.6 (24.5)
5	22.7 (25.0)
6	22.9 (25.5)
7	23.2 (25.8)
8	23.1 (25.3)

Table 5.3: VBSS (with $p = 5$) vs HBSS (with $p = 5$) for various numbers of iterations. ATCS is the deterministic heuristic result. LEE is the hill-climber (non-randomized) of Lee *et al.*

Algorithm	API	TIME	Gen	Visits	Sols
ATCS	0.0	0.009	1,830	60	1
LEE	12.4	0.012	1,909	61	2
VBSS (1)	5.0 (11.7)	0.021	3,660	120	2
HBSS (1)	4.7 (12.4)	0.159	3,660	120	2
VBSS (10)	16.4 (21.1)	0.123	20,130	660	11
HBSS (10)	15.5 (20.4)	1.585	20,130	660	11
VBSS (100)	22.7 (25.0)	1.122	184,830	6,060	101
HBSS (100)	21.6 (23.9)	15.467	184,830	6,060	101
VBSS (200)	23.8 (26.1)	2.234	367,830	12,060	201
HBSS (200)	22.6 (24.4)	30.932	367,830	12,060	201

parentheses are the results of the best run of the 10 runs for each problem instance.

For the HBSS algorithm, the degree 5 polynomial gives highest average percent improvement over the deterministic heuristic solution (API), though is not statistically significant as compared to either the degree 4 or 6 polynomials. All further comparisons involving HBSS will use the results given by the degree 5 polynomial bias function. For the VBSS results, the best bias function is even less clear. No statistical significance was found among polynomial bias functions of degrees 4 through 8. Further comparisons will use the results given by the degree 5 polynomial bias function.

We now turn to a direct comparison of VBSS and HBSS in Table 5.3. We compare the algorithms with several numbers of iterations. We also compare to the deterministic heuristic solution listed in the row labeled ATCS and to the hill-climber proposed by Lee

et al. [134] applied to the result of the deterministic heuristic solution listed in the table as LEE. First note that all rows in the table should necessarily give improved performance as compared to ATCS. This is due to the fact that both HBSS and VBSS, as implemented, first compute the ATCS solution; and LEE hill-climbs on the ATCS solution. We can now make the following observations:

- Both VBSS and HBSS begin outperforming LEE when 10 or more iterations are computed. But, for a single iteration of either HBSS or VBSS, the solutions are not as good as that of LEE and require more time than LEE. Therefore, HBSS and VBSS both appear useful for improving upon solutions provided the CPU time is available. If you desire a good solution almost instantaneously, then LEE might be the better choice. Yet in slightly more than 2 seconds using 200 iterations of VBSS, you can achieve almost twice the API as compared to the API of LEE.

- Consider any number of iterations (1, 10, 100, or 200) and note that VBSS using the heuristic values as bias gives better results than HBSS using rank bias with that same number of iterations.

- Upon examining CPU Times, it should be noted that for these 60 job problems, VBSS exhibits a greater than 10-fold speedup over HBSS. This is due to the need of HBSS to sort the choices according to the heuristic.[1] Given this, one can compare 100 iterations of VBSS to 10 iterations of HBSS; and similarly 10 iterations of VBSS to 1 iteration of HBSS. With this in mind, you can clearly find significantly better results by value-biasing in a fraction of the time required to rank-bias the stochastic search.

Value-biasing is a clear winner over rank-biasing in terms of solution quality and in terms of the computational efficiency of the search in this problem domain. One may wonder if there are circumstances where it is better to abandon the discriminatory advice of the heuristic values in favor of a rank-biased approach. Genetic algorithms can sometimes benefit from using a rank-based selection strategy [91]. Also, in Bresina's original telescope scheduling problem, rank-biased stochastic sampling performed very well (although he does not mention ever considering the more obvious value-biased alternative).

[1] Note, however, that if it is possible to compute the HBSS decisions in linear time (i.e., if sorting is not necessary) then the speedup may be less than the 10-fold seen in this experiment. However, depending on the significance of the constant factor in such a linear time version in this problem domain with a branching factor of 60, the best HBSS can do in terms of CPU time might still be the $O(n \log n)$ results reported here.

Table 5.4: VBSS plus a local hill-climb (VBSS-HC) vs Iterative Sampling plus a local hill-climb (IS-HC). VBSS-HC uses a polynomial bias function of degree 5. Both algorithms perform the hill-climb on the results of each of its iterations. ATCS is the deterministic heuristic result. LEE is the hill-climber (single-start, non-randomized) of Lee *et al.*SA is simulated annealing for the specified number of restarts beginning with ATCS solution.

Algorithm	API	TIME	Gen	Visits	Sols
ATCS	0.0	0.009	1,830	60	1
LEE	12.4	0.012	1,909	61	2
SA (1)	25.3 (27.8)	224.60	20,001,830	7,838,400	7,838,341
SA (3)	28.1 (29.3)	673.10	60,001,830	23,515,080	23,515,021
VBSS-HC (1)	15.1 (19.6)	0.021	3,791	122	4
VBSS-HC (10)	21.1 (23.7)	0.123	20,725	667	22
VBSS-HC (100)	24.7 (26.6)	1.122	190,076	6,122	202
VBSS-HC (200)	25.5 (27.2)	2.236	378,243	12,183	402
VBSS-HC (500)	26.6 (28.0)	5.570	942,744	30,366	1,002
VBSS-HC (1000)	27.4 (28.7)	11.121	1,883,579	60,671	2,002
VBSS-HC (2500)	28.1 (29.4)	27.742	4,706,084	151,586	5,002
VBSS-HC (5000)	28.7 (29.9)	55.674	9,410,259	303,111	10,002
VBSS-HC (10000)	29.3 (30.4)	114.140	18,818,609	606,161	20,002
IS-HC (1)	0.0 (0.0)	0.023	716	75	16
IS-HC (10)	0.0 (0.0)	0.226	7,163	754	164
IS-HC (100)	0.0 (0.0)	2.272	71,633	7,541	1,641
IS-HC (200)	0.0 (0.0)	4.543	143,266	15,082	3,282
IS-HC (500)	0.0 (0.0)	11.351	358,165	37,305	8,205
IS-HC (1000)	0.0 (0.0)	22.700	716,330	75,410	16,410
IS-HC (2500)	0.0 (0.0)	56.713	1,790,825	186,525	41,025
IS-HC (5000)	0.0 (0.0)	113.446	3,581,650	373,050	82,050
IS-HC (10000)	0.0 (0.0)	226.902	7,163,300	754,100	164,100

5.8 Value-Biasing Local Search Starting Configurations

In the previous Section, we observed that LEE requires a tiny fraction of the CPU time required to perform multiple iterations of VBSS, but also observed that if we had the extra time, we could find significantly better solutions than LEE with VBSS. Perhaps, VBSS could be used as a means of extending the Lee *et al.* hill-climber to a multi-start algorithm. Here we examine the use of VBSS as a mechanism for seeding the starting solutions of Lee *et al.*'s local hill-climber. As an added point of comparison, we also consider beginning a simulated annealing search at the solution given by the ATCS dispatch policy. We have implemented a computationally intense simulated annealing procedure that uses a modified

Lam schedule (see [19] for description of the modified Lam schedule) and that is allowed to run for a total of 20 million evaluations.

In Table 5.4 we compare this VBSS seeded hill-climber (VBSS-HC), the hill-climber with unbiased random starting configurations (IS-HC), LEE, and the deterministic dispatch policy ATCS. We make the following observations:

- The unbiased hill-climber never finds a solution better than ATCS. In fact, in most cases (though not shown here) the best solution given by the unbiased hill-climber on any problem instance is far worse than that of the ATCS solution. The primary reason for this is that the Lee *et al.* hill-climber assumes a good starting configuration with its very limited operator set. Unbiased random starting solutions are highly unlikely to be such good starting solutions for the problem instances of this problem set.

- Using VBSS to seed the starting configurations requires approximately half the CPU time for the search as compared to using unbiased random starting configurations. Though it takes more time to generate starting solutions with VBSS (evident in the higher number of search nodes generated by VBSS-HC), the underlying assumption of Lee *et al.*'s hill-climber, that it starts at a good solution holds in the case of VBSS. Therefore, the hill-climb is short (note the lower numbers of search nodes and solutions nodes visited by VBSS-HC as compared to IS-HC). In contrast, it takes nearly negligible time to generate unbiased random starting solutions, but those solutions are far from local optima, so the hill-climb takes significantly more time.

5.9 Comparison to Systematic Heuristic Search Methods

We now consider a computational comparison of VBSS and VBSS-HC with various versions of discrepancy search. The purpose of this comparison is to explore the power of stochastic sampling in combinatorial domains where systematic search gets bogged down considering a large number of un-promising solutions. For example, limited discrepancy search is required to systematically exhaust every solution path containing a single discrepancy from the heuristic's advice before it ever considers a solution path with two discrepancies. During this search, there might be several decision contexts in which the heuristic very strongly prefers one choice over the others, or very strongly prefers against one or more choices in some decision contexts. LDS has no way of skipping over the single discrepancy solution paths where it is clear to the heuristic that one or more choices are

bad and thus systematically considers them anyway. Similarly, depth-bounded discrepancy search is unable to consider a solution with discrepancies at depth 4 or greater before it exhausts all solution paths with all possible discrepancy combinations from the root through and including depth 3. Although its assumption of a heuristic's tendency to be most fallible at the start of the search is a good one in most cases, at the same time if the search-space has a very large branching factor, then it is still likely to be considering a large number of solution paths that might be clearly un-promising to the heuristic. Our hypothesis is that stochastic sampling approaches are able to better tune the heuristic's advice to the context and though incomplete are capable of finding better solutions given limited time than the systematic approaches of LDS and DDS in domains where the branching factor is very high.

Specifically, we will consider the following variations of discrepancy search:

- LDS-all-single: Limited Discrepancy Search[2] considering all single discrepancy (or less) solution paths. There are $1,771$ such solution states in each of these 60 job problem instances.

- LDS-all-two: Limited Discrepancy Search[3] considering all two-discrepancy (or less) solution paths. There are $1,533,116$ such solution states.

- DDS-depth-2: Depth-Bounded Discrepancy Search considering all discrepancies through depth 2. There are $3,540$ solution states considered.

- DDS-depth-3: Depth-Bounded Discrepancy Search considering all discrepancies through depth 3. There are $205,320$ solution states considered.

The results of this comparison can be found in Table 5.5. The following observations are made:

- VBSS on average finds better solutions than LDS-all-single in less time (with 500 iterations of VBSS) and even better solutions with slightly more time than LDS (1000 iterations of VBSS).

- In approximately half the time, the VBSS seeded hill-climber (VBSS-HC with 1000 iterations) finds solutions better than DDS (DDS-depth-2).

[2]The improved version (ILDS) of Korf [126] is used.
[3]Again, the improved version (ILDS) of Korf [126] is used.

Table 5.5: VBSS and VBSS-HC as compared to discrepancy search procedures.

Algorithm	API	TIME	Gen	Visits	Sols
VBSS (100)	22.7 (25.0)	1.12	184,830	6,060	101
VBSS (500)	25.1 (27.0)	4.44	916,830	30,060	501
VBSS (1000)	26.0 (27.8)	8.87	1,831,830	60,060	1001
VBSS-HC (100)	24.7 (26.6)	1.12	190,076	6,122	202
VBSS-HC (500)	26.6 (28.0)	5.57	942,744	30,366	1,002
VBSS-HC (1000)	27.4 (28.7)	11.12	1,883,579	60,671	2,002
VBSS-HC (10000)	29.3 (30.4)	114.14	18,818,609	606,161	20,002
LDS-all-single	24.3	6.04	1,601,615	70,270	1,771
LDS-all-two	28.8	3902.04	912,864,816	48,219,521	1,533,116
DDS-depth-2	23.6	20.94	6,056,940	205,320	3,540
DDS-depth-3	26.5	911.72	339,393,960	11,703,240	205,320

- In less than two minutes, the VBSS seeded hill-climber (VBSS-HC with 10000 iterations), finds solutions that are on average better than solutions found by allowing LDS to run for over an hour per instance (LDS-all-two) and that are on average better than allowing DDS to run for 15 minutes (DDS-depth-3).

- VBSS-HC can find better solutions while generating and visiting far fewer search nodes than LDS or DDS and while visiting far fewer solution nodes than LDS or DDS. In particular compare VBSS-HC with 10,000 iterations to LDS-all-two and DDS-depth-3. For example, in considering all solution paths with two or less discrepancies from the heuristic's advice, LDS generates nearly a billion search nodes, visiting almost 50 million of them, and evaluating over a million and a half solution nodes; while 10,000 iterations of VBSS-HC generates just under 19 million search nodes visiting approximately 600 thousand, and evaluating only 20 thousand solution nodes.

The overall theme of these observations is that in far less CPU time, VBSS can find significantly better solutions on average as compared to the systematic discrepancy search procedures. This would clearly not be the case in all domains, but for this problem domain with the large branching factor, it is. Though often used in problems of optimization, LDS and DDS are better suited to constrained problem domains – for example, constraint satisfaction domains where we are instead looking for any feasible solution; but also in constrained optimization domains where there is some objective criteria, but which the problem's search-space is not as cumbersome in size as a large sequencing problem.

5.10 Summary

In this Chapter, we considered the problem of weighted tardiness scheduling under the constraint of sequence-dependent setups. This is a problem for which there is little prior work. In dealing with the weighted tardiness scheduling problem, many assume away the setup costs and later reintroduce the sequence-dependent setups with ad hoc modifications. The dispatch policy of ATCS coupled with a local hill-climber was until now the best available heuristic method for this problem. Given this problem, we experimentally compared a number of heuristic-guided search algorithms using the ATCS heuristic within the search.

In the first comparison, we showed that the value-biased approach of VBSS does make better use of the inherent discriminatory power of the heuristic as compared to the rank-biased approach of HBSS. VBSS exhibited solutions with better objective values; and further, was able to find these solutions in significantly less CPU time. Neither of these findings are surprising. Given that we made the assumption that our heuristic is a good one, then it only makes sense that we should be able to do better with value-biasing since we are making use of more of the information contained therein; while the rank-biased approach essentially throws away heuristic information. Furthermore, VBSS dominating HBSS in terms of CPU time falls directly out of the need of HBSS to rank-order the choices at each decision step.

In the second comparison, we considered the idea of using VBSS as a seeding mechanism for the starting configurations of a multi-start extension to Lee *et al.*'s hill-climber. Lee *et al.*'s hill-climber is designed with the assumption of beginning at a good search-state and previously applied only to the solution given by the ATCS dispatch policy. This procedure gives good results with negligible computational overhead. However, given more CPU time, we showed that it is possible to further improve upon the results by using VBSS to generate starting configurations and applying Lee *et al.*'s hill-climber beginning at the VBSS generated solutions.

In the final comparison, we considered the hypothesis: given the large branching factor of un-constrained sequencing problems, VBSS could make more effective use of heuristic guidance as compared to truncated systematic approaches such as LDS and DDS. LDS and DDS, as they are defined, systematically exhaust the search space considering solution nodes in order of increasing discrepancy with the heuristic's prescribed path. In problem domains where there are a large number of choices at each decision point, it is likely that many of these choices will be deemed as un-promising by the heuristic values; but for example, LDS is forced to systematically consider one discrepancy solutions where the

single discrepancy is each of these un-promising choices before ever considering any solution with two discrepancies. As was seen through the number of nodes generated and visited and the number of solution states evaluated, LDS and DDS become overwhelmed by the search-space size. Consequently, VBSS and VBSS-HC, with the power to ignore un-promising discrepancies are better able to reach more productive regions of the search-space in domains such as this one with the large branching factor.

Chapter 6

AQDF: Algorithm Quality Density Function

6.1 Overview

In this Chapter, we define a descriptive tool that we call the Algorithm Quality Density Function (AQDF). At the foundation of the AQDF is Bresina's concept of a quality density function (QDF). More importantly, the study and analysis of the AQDF is motivated by the work of Gomes *et al.* They studied the runtime distributions of backtrack search in constraint satisfaction domains and offered as rationale for rapid randomized restarts the heavy-tailed nature of these runtime distributions. In a similar way, I examine the quality distributions of stochastic sampling search in combinatorial optimization domains and examine what if anything these AQDFs may tell us about the search behavior. Section 6.2 describes Bresina's concept of a quality density function upon which the AQDF is based. Section 6.3 defines and discusses our AQDF. Section 6.4 relates the AQDF to the concept of a performance profile from the body of work that has been done with Anytime algorithms. A summary concludes the Chapter in Section 6.5.

6.2 Quality Density Function

Bresina *et al.* introduce the idea of a *quality density function* (QDF) [23, 24]. The QDF is the distribution of the quality of solutions obtained from sampling uniformly from a solution space (see Figure 6.1 for examples of QDFs). Bresina's QDF characterizes the feasible-solution space of a problem instance in terms of the distribution of solution qual-

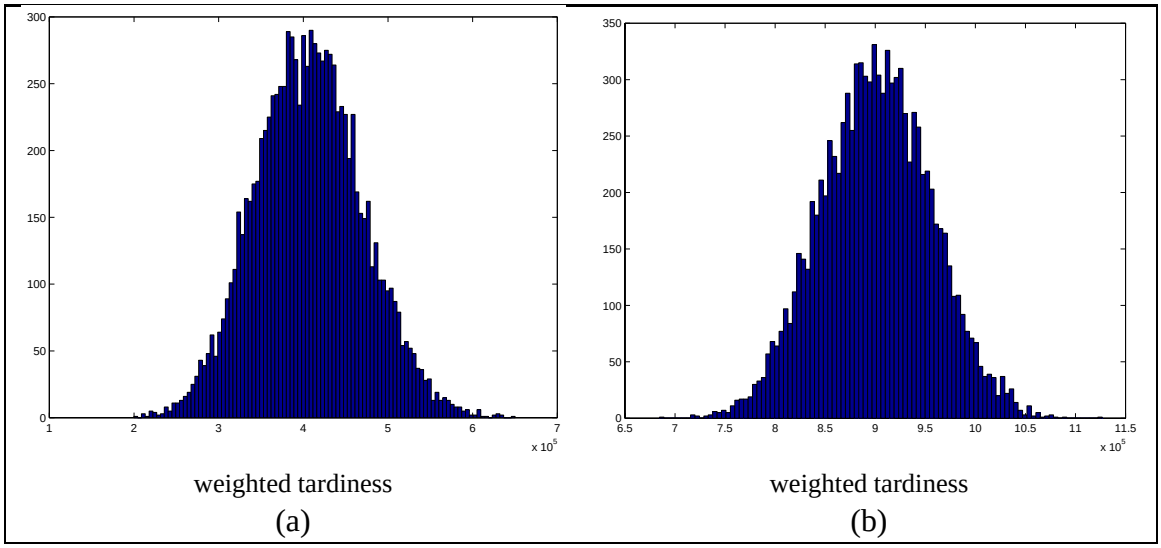


Figure 6.1: Histogram approximations of QDFs for two instances of a weighted tardiness scheduling problem: (a) QDF of a loose duedate problem; (b) QDF of a tight duedate problem.

Table 6.1: Descriptive comparison of the QDF and AQDF.

	QDF	AQDF
Sampling is	uniform	according to particular algorithm
Characterizes	problem space	algorithm's performance
Is algorithm	independent	dependent

ities. Using such a QDF, Bresina computes the number of standard deviations away from the mean is a solution produced by a particular algorithm. He then compares this measure to that given by other algorithms for his scheduling problem to benchmark the performance of the algorithms in terms of solution quality for a given problem instance. This is incorporated into a broader measure that he calls *expected solution quality* (ESQ) that includes computation time in the comparison [24].

6.3 Algorithm Quality Density Function

For the basis of our descriptive analysis of the WHISTLING algorithm, we now define an *algorithm quality density function* (AQDF) as the distribution of the quality of solutions obtained by sampling from a solution space via a particular stochastic sampling algorithm rather than uniform samples. So for example, given a heuristic for a particular scheduling problem and a bias function, we can compute the AQDF of the WHISTLING algorithm for

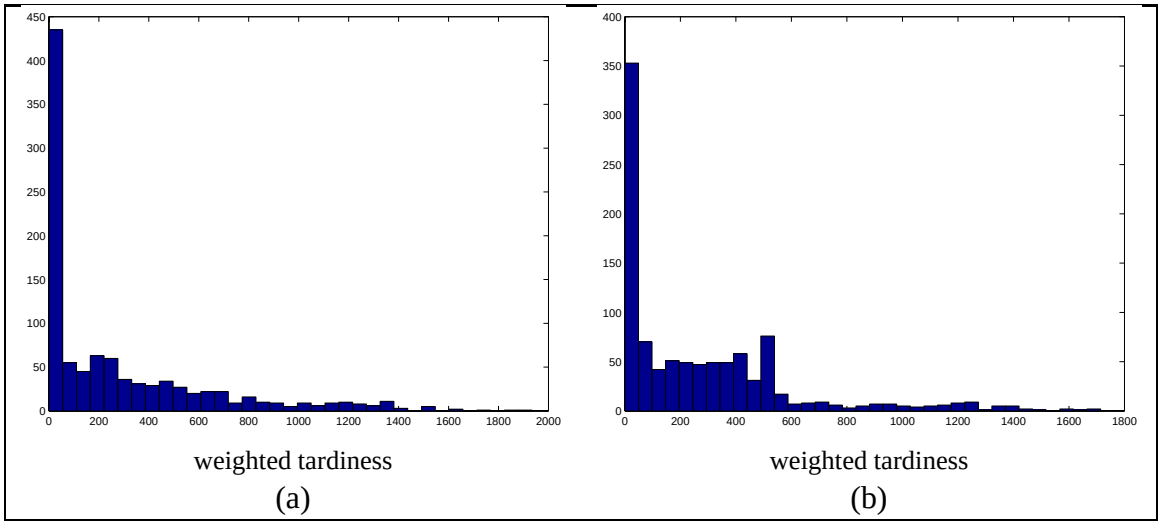


Figure 6.2: Histogram approximations of AQDFs for an instance of a weighted tardiness scheduling problem with sequence-dependent setups and with loose duedates and the WHISTLING algorithm using the ATCS heuristic and: (a) "weaker" bias function; (b) "strong" bias function.

an instance of that scheduling problem. Figure 6.2 and Figure 6.3 show example AQDFs of the WHISTLING algorithm for two such problem instances – a loose duedate instance and a tight duedate instance of a weighted tardiness scheduling problem, respectively. Our AQDF characterizes the performance of a particular stochastic sampling algorithm for a problem instance in terms of the distribution of solution qualities obtained by that algorithm. Table 6.1 compares and contrasts the AQDF and QDF.

Figure 6.2(b) illustrates an approximation of the AQDF of WHISTLING using a "strong" bias; while Figure 6.2(a) shows an approximation of the AQDF of WHISTLING using a somewhat "weaker" bias. For this problem instance, there are several optimal solutions where the weighted tardiness is 0. Using the stronger bias, WHISTLING tends to favor the suboptimal heuristic solution more heavily, although it does find the optimal solution very frequently and would require very few iterations until it does so; while the somewhat "weaker" bias allows WHISTLING to widen the search and finds one of the nearby optimal solutions more frequently for this problem instance.

Contrast this with Figure 6.3 where the stronger bias (see the approximation of the AQDF of Figure 6.3(b)) allows WHISTLING to reach deeper into the "better" region of the search space concentrated very closely around the heuristic solution; while WHISTLING with the "weaker" bias seems to wander a bit too far from the heuristic solution. For this problem instance, the heuristic appears more informed than it did for the other problem

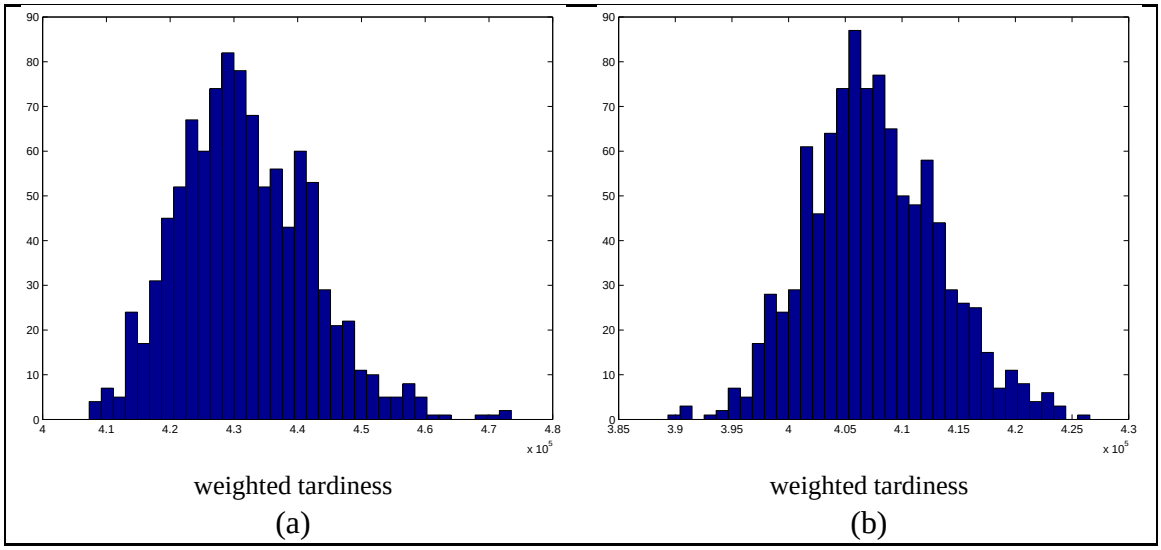


Figure 6.3: Histogram approximations of AQDFs for an instance of a weighted tardiness scheduling problem with sequence-dependent setups and with tight duedates and the WHISTLING algorithm using the ATCS heuristic and: (a) "weaker" bias function; (b) "strong" bias function.

instance. If we had this AQDF data ahead of time it would have allowed us to make a more informed choice for the bias function and we could have chosen the stronger bias function for a more productive search.

Figure 6.4 shows approximations of the AQDFs of the WHISTLING algorithm using an earliest duedate (EDD) heuristic (Figure 6.4(a)) and a weighted shortest processing time (WSPT) heuristic (Figure 6.4(b)) for a weighted tardiness problem instance with very loose duedates and a wide duedate range (no setup times in this instance). WHISTLING with EDD finds better solutions for this problem instance than does WHISTLING with WSPT. Figure 6.5 shows the equivalent AQDFs for another instance of the problem with tight duedates. This example is in contrast with that of Figure 6.4 in that we see that WSPT (Figure 6.5(b)) does significantly better than EDD (Figure 6.4(a)). This should serve as an example that for some problems there may be instances where WHISTLING with one heuristic may perform better than another while at the same time there may be instances where this second heuristic is superior. Later in this thesis, we develop algorithms for computing approximations of an AQDF online with the search to help make such heuristic choice decisions in a more informed way.

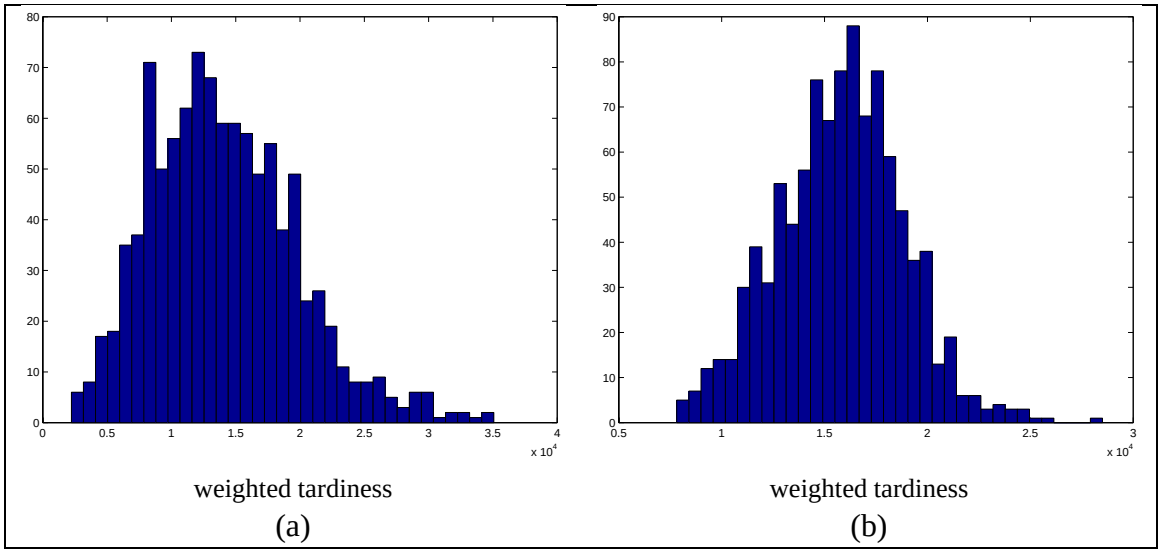


Figure 6.4: Histogram approximations of AQDFs for an instance of a weighted tardiness scheduling problem with very loose duedates and a wide duedate range and the WHISTLING algorithm using the heuristics: (a) earliest duedate; (b) weighted shortest processing time.

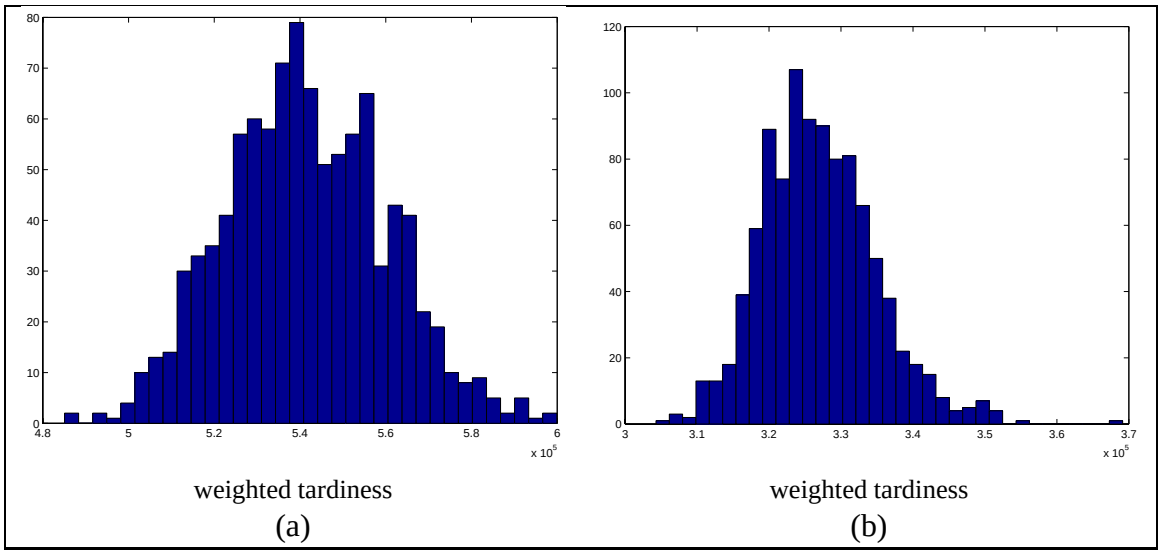


Figure 6.5: Histogram approximations of AQDFs for an instance of a weighted tardiness scheduling problem with tight duedates and the WHISTLING algorithm using the heuristics: (a) earliest duedate; (b) weighted shortest processing time.

Table 6.2: Descriptive comparison of a Performance Profile and the AQDF.

	Performance Profile	AQDF
Problem	class dependent	instance dependent
Generated for	several random instances	single instance
Generated	ahead of time	online
Quality means	percent improvement	objective value
Models	quality as function of time	quality of single iterations

6.4 Relation to Performance Profiles

To someone familiar with the concept of a performance profile, it may be desirable to consider the relationship of the AQDF and a performance profile. Recall from Chapter 3 that a performance profile is a function that maps the runtime of an algorithm to the expected quality of the result obtained by executing the algorithm for that amount of time. Furthermore, a performance profile is often (if not always) a continuous function. Often, "quality" specifies the amount of improvement over some initially generated solution.

Now consider an anytime algorithm that repeatedly generates solutions and saves the best of those solutions encountered along the way. Further, consider that partial solutions are not valued and that the time required to generate each complete solution is invariant. For example, the time to construct a single solution to a sequencing problem by the WHISTLING algorithm doesn't vary across iterations for a single problem instance. The amount of computational effort to choose the first element in the sequence is $O(n)$, to choose the second element in the sequence is $O(n-1)$, and so forth until a complete sequence is obtained.[1] This computational effort does not vary from iteration to iteration. A performance profile for this example, would track the expected quality of the best found solution as a function of time. But the expected quality of this best found solution should only change at discrete time intervals (i.e., every X time units where X is the length of time to generate a single solution). The rest of the performance profile, though it would likely be smoothed out as a continuous function, would for all intents and purposes be completely irrelevant. Lastly, if you had a detailed model of the expected solution quality at the first such time interval, then you would have all that is necessary for successive numbers of iterations. And thus, for such an algorithm, there is but a single time slice of the performance profile that is at all relevant. This is illustrated in Figure 6.6.

The AQDF acknowledges this and is most relevant to algorithms where the approxi-

[1]This is under the assumption that the heuristic value function used is linear in the number of choices. The same argument can be made for heuristics of greater complexity.

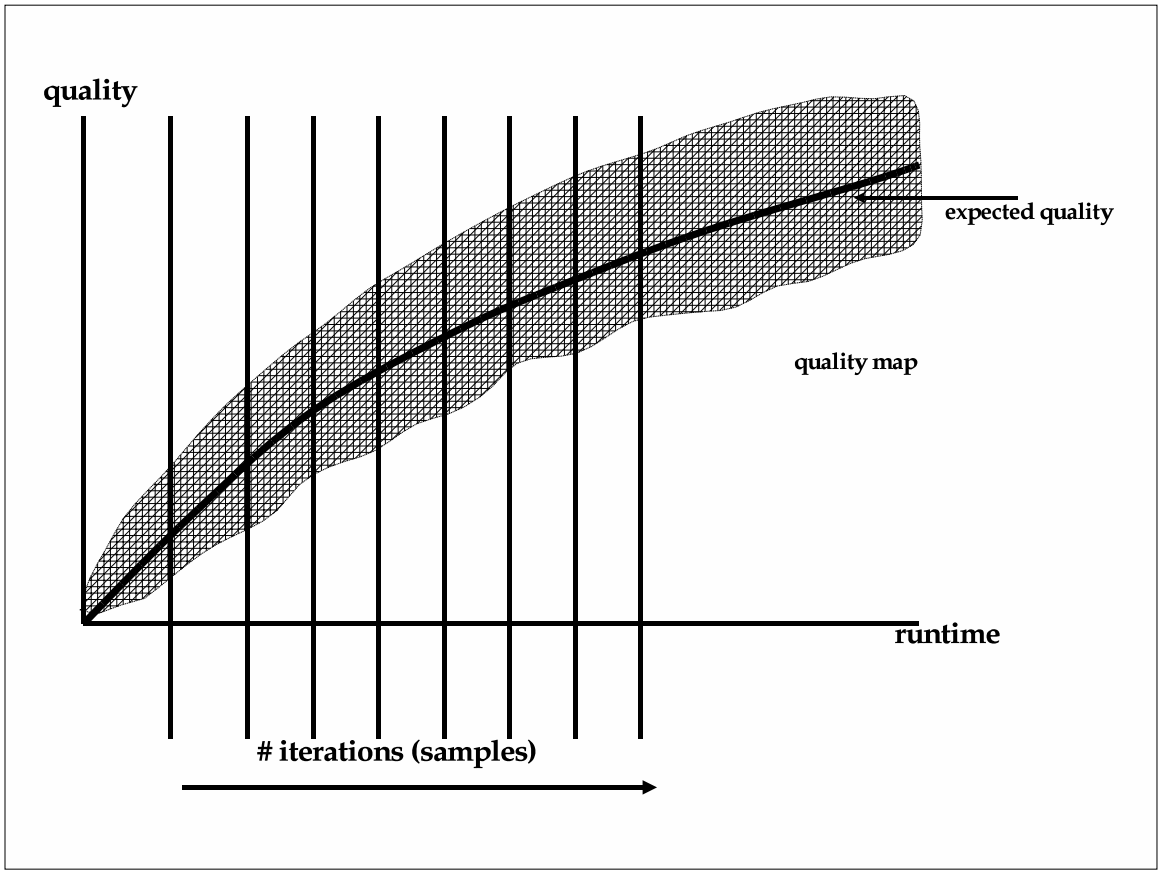


Figure 6.6: Relationship of the AQDF with the concept of a performance profile. The AQDF is a problem instance dependent, detailed model of the expected quality of solutions generated by single iteration runs of a stochastic sampling algorithm – essentially the first time slice indicated in the figure.

mate computation time of a single iteration is invariant. It redefines "quality" as simply the objective value rather than in terms of improvement over some initial solution. It also is generated using a single problem instance, rather than a set of random instances deemed indicative of future problems as is used to generate a PP. Given this, the AQDF can be described as a problem instance dependent, detailed model of the expected quality of solutions generated by single iteration runs of the stochastic sampling algorithm. Table 6.2 summarizes the similarities and differences between the AQDF and the PP.

6.5 Summary

In this Chapter, the descriptive tool of the Algorithm Quality Density Function was presented. The AQDF is a model of the distribution of the quality of solutions obtained by single iteration runs of a given stochastic sampling algorithm. It is a problem instance dependent and algorithm dependent descriptive tool. The AQDF is related to the concept of a performance profile. The AQDF can be described as a detailed model of the time slice of a performance profile representing single iteration runs of a stochastic sampling algorithm. Finally, the AQDF is a problem instance dependent detailed model; while the performance profile is a problem class dependent model.

Chapter 7

QD-BEACON: Quality Distribution Based sEArch CONtrol

7.1 Overview

Given the definition of the AQDF of the previous Chapter, consider that if we had a method of estimating an AQDF with relatively few samples, then we could perhaps use such an estimation for effective search control. For example, we could approximate the AQDFs of the WHISTLING algorithm for a problem instance using some number of heuristics. Then given estimates of the AQDFs, we could choose which heuristic to use for further iterations according to the probability of finding a better solution than the best found so far. The estimates of the AQDFs can continue to be refined as we iterate the search. In this Chapter, we define the QD-BEACON (Quality Distribution Based Search Control) framework. QD-BEACON provides the functionality to compute models of AQDFs online with the search as well as the functionality to exploit those models for effective search control guidance.

The functionality that is provided to search algorithms by QD-BEACON is summarized in Section 7.2. To implement this framework, a method for estimating an AQDF is needed. Section 7.3 presents three methods for this estimation using: 1) normal distributions; 2) kernel density estimators; and 3) generalized extreme value distributions. Within the QD-BEACON framework, a method is needed for balancing the tradeoff between exploiting the current estimates of the AQDFs and exploring to improve upon those estimates. Section 7.4 first discusses this tradeoff theoretically in terms of the k-armed bandit problem and then presents our exploration strategy based upon our analysis of a new problem that we call the "max k-armed bandit problem". A summary concludes the Chapter in Section 7.5.

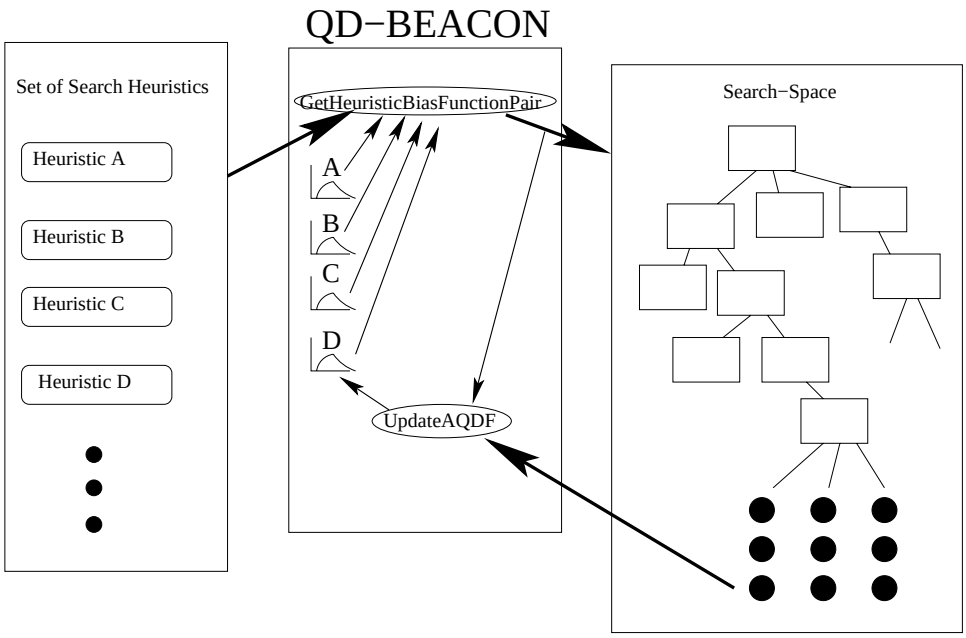


Figure 7.1: The QD-BEACON framework provides a methodology for choosing from among a set of search heuristics based on learned statistical models of their performance on the problem instance at hand. The result of each iteration of the search provides feedback to QD-BEACON used to refine the statistical models.

7.2 QD-BEACON

The QD-BEACON framework provides the functionality necessary to model AQDFs on-line within the search and to exploit those models for effective search control guidance. For example, given a set of search heuristics for a problem, QD-BEACON provides a method for learning statistical models of the distributions of solution qualities given by each of the heuristics in the set. Given these statistical models, QD-BEACON allows a stochastic sampling algorithm to make a more informed choice of search heuristic. This example is illustrated in Figure 7.1.

The QD-BEACON framework primarily provides the following functionality for modeling AQDFs and for using those models to select a search heuristic for each iteration of the search:

- UPDATEAQDF(Heuristic,BiasFunction,Quality): This function allows a stochastic sampling algorithm to give feedback on the quality of the solution found by the heuristic / bias function pair and is used to update the appropriate AQDF.

- GETHEURISTICBIASFUNCTIONPAIR(): This function returns the heuristic and bias

function to use for the next iteration of the algorithm. This pair is chosen according to the AQDF data, the probability of finding a solution better than the best found so far with each of the heuristic/bias function pairs, and the exploration strategy described in Section 7.4.

QD-BEACON also provides the following functionality for a stopping criterion, for considering infeasible search trajectories in constrained optimization domains, and for setting up the set of heuristics to be combined by the framework:

- CONTINUE(P): This function returns *true* if the probability of finding a better solution than the best found so far given the AQDF data (of the observed best) is at least P and *false* otherwise. It allows an alternative stopping criteria in addition to a maximum number of iterations.

- REPORTINFEASIBLESOLUTIONSTATE(Heuristic,BiasFunction): This function allows a stochastic sampling algorithm to report that an infeasible solution state was found during the current iteration by the heuristic / bias function pair. For some problems, this function may not be necessary. But for other problems, such as constraint satisfaction problem domains, where not every probe from the root of the search-tree to a leaf node is guaranteed to find a feasible solution, it is necessary to consider such failed samplings in future heuristic / bias function choices. For each heuristic / bias function pair i, this function is used to maintain a variable Infeasible$_i$ which is a count of the number of infeasible solution nodes found using i.

- QD-BEACONINIT(Heuristics, Biases): This function performs any necessary initialization required by the QD-BEACON framework. It takes as input an array of heuristic functions and an array of bias functions. These lists should be paired – meaning that the i-th element of the list of heuristics is always used with the i-th element of the list of bias functions (i.e., if you wish to consider one heuristic with multiple bias functions, then that heuristic should be in the list of heuristics multiple times).

Given the QD-BEACON framework described above, an extended version of the WHISTLING algorithm that incorporates QD-BEACON is presented in Algorithm 7.1.

Algorithm 7.1: Integrated WHISTLING/QD-BEACON
Input: Number of iterations I; an array of "heuristic" functions; an array of "bias" functions; an "objective" function; a probability P; and a search-tree T.
Output: A solution S.

```
WHISTLING-QD-BEACON(I, Heuristics, Biases, objective, P, T)
(1)   QD-BEACONINIT(Heuristics, Biases)
(2)   bestsofar ← solution S obtained if "heuristic" h_1 is followed from T
(3)   evaluate[bestsofar] ← objective(bestsofar)
(4)   foreach heuristic h ∈ Heuristics - h_1
(5)     S ← solution obtained if heuristic h is followed from T
(6)     evaluate[S] ← objective(S)
(7)     if evaluate[S] is superior to evaluate[bestsofar]
(8)       bestsofar ← S
(9)   repeat I times or until not CONTINUE(P)
(10)    {heuristic, bias} ← GETHEURISTICBIASFUNCTIONPAIR()
(11)    S ← root search-node of T
(12)    while S is a decision node of T
(13)      foreach choice C from S
(14)        force[C] ← bias(heuristic(C, S))
(15)      WinnerSoFar ← arbitrary choice C from S
(16)      Challengers ← the set of choices from S − WinnerSoFar
(17)      foreach choice C in the set Challengers
(18)        with probability P(force[C], force[WinnerSoFar]) (see Eq. 4.5)
(19)          force[C] ← force[C] + force[WinnerSoFar]
(20)          WinnerSoFar ← C
(21)        otherwise
(22)          force[WinnerSoFar] ← force[WinnerSoFar] + force[C]
(23)      S ← Successor(S, WinnerSoFar)
(24)    if S is not a feasible solution state
(25)      REPORTINFEASIBLESOLUTIONSTATE(heuristic, bias)
(26)    else
(27)      evaluate[S] ← objective(S)
(28)      UPDATEAQDF(heuristic, bias, evaluate[S])
(29)      if evaluate[S] is superior to evaluate[bestsofar]
(30)        bestsofar ← S
(31)  return bestsofar
```

7.3 Estimating an AQDF

The example AQDFs of the previous Chapter were generated using a relatively large number of iterations – 1000. A design criteria of the QD-BEACON framework is that it should be capable of roughly estimating the AQDF of a stochastic search algorithm online during the search using very few samples (e.g., 10 samples). So for example, consider a case where you have some scheduling problem to solve and you have sufficient computation time available for approximately 100 iterations of the WHISTLING algorithm. Also consider that you have two heuristics for the problem, but you do not know which heuristic is more appropriate for the given problem instance nor do you have enough time to do the necessary computation to make such a determination. It is desirable that QD-BEACON be capable of estimating the AQDF for each heuristic given relatively few samples from the WHISTLING algorithm using each. Then using these AQDFs, it is also desirable that QD-BEACON be capable of deciding which heuristic to continue using for the remaining samples of the WHISTLING algorithm based on the probabilities of finding a better solution than the best found so far. It is also important that QD-BEACON be capable of estimating the AQDFs efficiently as well as capable of efficiently choosing among competing heuristics (and possibly bias functions) for successive iterations. In the sections that follow describing three methods for estimating an AQDF, we assume that we are dealing with a problem domain in which smaller objective values are better.

7.3.1 Normal Estimates

One possibility for estimating the AQDF is to fit a normal distribution to the data. The advantage of this approach is that it can be computed quickly and online during the search with little difficulty. The disadvantage is that if the AQDF is very much unlike a Gaussian (e.g., Figure 6.2) then the estimated probability of finding a better solution will be very inaccurate and QD-BEACON may suggest using a less well-suited heuristic for the remainder of the search.

To estimate an AQDF by a normal distribution within the QD-BEACON framework, we do the following. For each AQDF i required by QD-BEACON for our problem, we maintain the following:

- N_i is the number of samples for the estimation of AQDF i (i.e., the number of samples using the i-th heuristic / bias function pair).

Algorithm 7.2: Normal Estimation of the AQDF for QD-BEACON
Input: A heuristic, a bias function, and the quality of the solution just obtained
Output:
Description: Updates the N_i, X_i, Y_i, P_i appropriately. Note that it assumes smaller objective values, or qualities, are better. BestSolution is the global best solution found so far, the value of which persists across calls to UPDATEAQDF().

UPDATEAQDF(Heuristic,BiasFunction,Quality)
(1) $i \leftarrow \{\text{Heuristic}, \text{BiasFunction}\}$
(2) $N_i \leftarrow N_i + 1$
(3) $X_i \leftarrow X_i + \text{Quality}$
(4) $Y_i \leftarrow Y_i + \text{Quality}^2$
(5) **if** Quality $<$ BestSolution
(6) BestSolution $\leftarrow$ Quality
(7) $\mu_i \leftarrow \frac{X_i}{N_i}$
(8) $\sigma_i \leftarrow \sqrt{\frac{Y_i - N_i \mu_i^2}{N_i - 1}}$
(9) **foreach** $j =$ heuristic/bias function pair
(10) $P_j \leftarrow N(\frac{\text{BestSolution} - \mu_j}{\sigma_j})$
(11) **else**
(12) $\mu_i \leftarrow \frac{X_i}{N_i}$
(13) $\sigma_i \leftarrow \sqrt{\frac{Y_i - N_i \mu_i^2}{N_i - 1}}$
(14) $P_i \leftarrow N(\frac{\text{BestSolution} - \mu_i}{\sigma_i})$

- $X_i = \sum_{j=1}^{N_i} S_{i,j}$ is the sum of these N_i samples ($S_{i,j}$ is the objective value of the j-th solution obtained by the i-th heuristic / bias function pair).
- $Y_i = \sum_{j=1}^{N_i} S_{i,j}^2$ is the sum of the squares of these N_i samples.

The N_i, X_i, and Y_i are initialized to 0 by QD-BEACONINIT(Heuristics, Biases). At the beginning of UPDATEAQDF(Heuristic,BiasFunction,Quality), N_i, X_i, and Y_i are updated appropriately for $i = \{\text{Heuristic}, \text{BiasFunction}\}$.

Upon updating the N_i, X_i, and Y_i, UPDATEAQDF(Heuristic,BiasFunction,Quality) continues by updating the probability of each heuristic / bias function pair finding a better solution than the best found so far. If "Quality" is a new best found solution, then for all heuristic / bias function pairs i, the probability P_i of finding a better solution than the best found so far, B, given AQDF i is updated according to:

$$P_i = N(\frac{B - \mu_i}{\sigma_i}). \tag{7.1}$$

The mean μ_i and standard deviation σ_i are computed according to:

$$\mu_i = \frac{X_i}{N_i} \tag{7.2}$$

and

$$\sigma_i = \sqrt{\frac{Y_i - N_i \mu_i^2}{N_i - 1}}. \tag{7.3}$$

If "Quality" is not a new best found solution, then P_i need only be updated for the given heuristic / bias function pair $i = \{\text{Heuristic}, \text{BiasFunction}\}$. A lookup table is used for the cumulative distribution of the standard normal, $N(\cdot)$. Algorithm 7.2 shows the algorithm for updating the AQDFs within QD-BEACON using normal estimation.

7.3.2 Kernel Density Estimation

A second possibility for estimating the AQDF is Kernel Density Estimation (see [193, 184, 17, 200, 183, 97]). A kernel density estimator makes little, if any, assumptions regarding the underlying distribution that it models. It provides a non-parametric framework for estimating arbitrary probability densities. In terms of estimating an AQDF, it provides the QD-BEACON framework with a non-parametric method for this estimation that relies on very limited distributional assumptions. The advantage of this approach is that it should be possible to more closely estimate arbitrary AQDFs. The disadvantage is that there is additional computational overhead associated with using kernel density estimates as compared to the normal distribution of the previous section. Kernel density estimation takes local averages to estimate a density function by placing smoothed out quantities of mass at each data point. The kernel density estimator is defined as:

$$\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^{n} K(\frac{x - X_i}{h}). \tag{7.4}$$

$K(\cdot)$ is a kernel function and h is called the bandwidth (also sometimes called the scale parameter or spreading coefficient). The X_i are the n sample values.

The kernel function chosen for the QD-BEACON framework is the Epanechnikov kernel [74]:

$$K(x) = \tfrac{3}{4\sqrt{5}}(1 - \tfrac{x^2}{5}) \quad \text{for } |x| < \sqrt{5} \text{ and otherwise } 0 \ . \tag{7.5}$$

Epanechnikov showed that this is the risk optimal kernel, but estimates using other smooth kernels are usually numerically indistinguishable. Thus the form of the kernel can be cho-

sen to best address computational efficiency concerns. In our case, the Epanechnikov kernel is a clear winner computationally for the following reasons:

- It is easy to integrate. Since we are most interested in using the density estimate to ultimately compute the probability of finding a better solution than the best found so far, this kernel function choice allows us to easily compute the cumulative probability distribution for arbitrary AQDFs.
- It is bounded. Due to the condition $|x| < \sqrt{5}$, only a limited number of sample values must be considered in computing the value of the kernel function. This is useful in reducing the computational overhead.

Although the choice of kernel function is not very critical in terms of numerical results, the choice of bandwidth on the other hand can be very crucial. Epanechnikov showed that the optimal choice of bandwidth is [74]:

$$h = (\frac{L}{nM})^{1/5}, \tag{7.6}$$

where

$$L = \int_{-\infty}^{\infty} K(x)^2 \, dx, \tag{7.7}$$

$$M = \int_{-\infty}^{\infty} (f''(x))^2 \, dx, \tag{7.8}$$

and where n is the number of samples.

Unfortunately, this computation is dependent on knowing the true distribution (M depends on $f(x)$). Commonly, it is assumed that the underlying distribution is normal which, if the chosen kernel function is the Epanechnikov, results in $h = 1.05\sigma n^{-1/5}$ where σ is the sample standard deviation.[1] The standard deviation σ is usually replaced by $s = \min\{\sigma, Q/1.34\}$, where Q is the interquartile range yielding $h = 1.05sn^{-1/5}$ [184].

We choose to instead make the assumption that the underlying distribution is the Gumbel distribution:

$$P(Z \leq z) = G(z) = \exp(-\exp(-(\frac{z-b}{a}))). \tag{7.9}$$

where b is called the location parameter and a the scale parameter. The reasoning behind this assumption is motivated by extreme value theory and will be discussed in greater detail in Section 7.3.3. The Gumbel distribution is one of the three types of extreme value

[1] If you are instead using the Gaussian kernel, then under the assumption of an underlying normal distribution, $h = 1.06\sigma n^{-1/5}$.

Algorithm 7.3: Kernel Density Estimation of the AQDF for QD-BEACON
Input: A heuristic, a bias function, and the quality of the solution just obtained
Output:
Description: Updates the N_i, X_i, Y_i, P_i, $S_{i,j}$, h_i appropriately. Note that it assumes smaller objective values, or qualities, are better. BestSolution is the global best solution found so far, the value of which persists across calls to UPDATEAQDF().
UPDATEAQDF(Heuristic,BiasFunction,Quality)
(1) $i \leftarrow \{\text{Heuristic}, \text{BiasFunction}\}$
(2) $N_i \leftarrow N_i + 1$
(3) $X_i \leftarrow X_i + \text{Quality}$
(4) $Y_i \leftarrow Y_i + \text{Quality}^2$
(5) insert Quality into sorted $S_{i,j}$
(6) $\mu_i \leftarrow \frac{X_i}{N_i}$
(7) $\sigma_i \leftarrow \sqrt{\frac{Y_i - N_i \mu_i^2}{N_i - 1}}$
(8) $Q_i \leftarrow$ interquartile range of $S_{i,j}$
(9) $h_i \leftarrow 0.79 \min(Q_i, \sigma_i) N_i^{-1/5}$
(10) **if** Quality $<$ BestSolution
(11) BestSolution $\leftarrow$ Quality
(12) **foreach** $j =$ heuristic/bias function pair
(13) $P_j \leftarrow$ result of Equation 7.12
(14) **else**
(15) $P_i \leftarrow$ result of Equation 7.12

distributions. Given the Gumbel distribution assumption, $M = \frac{1}{4a^5}$. Note that the standard deviation of the Gumbel distribution is $\sigma = \frac{\pi a}{\sqrt{6}}$ [152]. From this, we have $a = \frac{\sigma\sqrt{6}}{\pi}$. We can now write M in terms of the sample standard deviation: $M = \frac{\pi^5}{4\sqrt{6}^5 \sigma^5}$. We are using the Epanechnikov kernel so $L = \frac{3}{5\sqrt{5}}$. This results in a value of h computed as: $h = 0.79sn^{-1/5}$ where again $s = \min\{\sigma, Q/1.34\}$.

Within QD-BEACON, we are interested in the cumulative distribution function for the purpose of computing the probability of finding a better solution than the best found so far. This can be obtained from integrating the kernel density estimator. Thus we have the probability P_i of finding a solution better than the best found so far given AQDF i:

$$P_i = \int_0^B \hat{f}_i(x)\, dx, \tag{7.10}$$

where B is the best solution so far, which equals[2]

$$P_i = \int_0^B \frac{1}{n_i h_i} \sum_{j=1}^{n_i} K(\frac{x - S_{i,j}}{h_i})\, dx. \tag{7.11}$$

Given our choice of the Epanechnikov kernel, this evaluates to:

$$\begin{aligned} P_i = \frac{3}{4 n_i h_i \sqrt{5}} \quad \sum_{j, |\frac{B - S_{i,j}}{h_i}| < \sqrt{5}} \quad & ((B - \frac{1}{5h_i^2}(\frac{B^3}{3} - B^2 S_{i,j} + B S_{i,j}^2)) - \\ & (S_{i,j} - h_i\sqrt{5} - \frac{1}{5h_i^2}(\frac{(S_{i,j} - h_i\sqrt{5})^3}{3} - \\ & (S_{i,j} - h_i\sqrt{5})^2 S_{i,j} + (S_{i,j} - h_i\sqrt{5}) S_{i,j}^2))) \end{aligned} \tag{7.12}$$

It should be noted, that if we maintain the samples in sorted order, then given that B must be less than or equal to the smallest value in this list[3], we compute this sum until we reach a sample $S_{i,j}$ such that $|\frac{B - S_{i,j}}{h_i}| \geq \sqrt{5}$. Once a sample for which this condition holds is reached in the list, the summation can end. Actually, rather than in a sorted list, we maintain the samples in a sorted histogram, maintaining counts of the number of samples with given discrete values. To compute $h_i = 0.79 s_i n_i^{-1/5}$, we compute σ_i as it had been in the previous section for the normal distribution and Q_i is readily accessible from the samples given that we maintain them in sorted order. The kernel density estimator version of QD-BEACON's UPDATEAQDF(Heuristic,BiasFunction,Quality) is shown in Algorithm 7.3.

7.3.3 Generalized Extreme Value Distribution

We now consider that the solutions to the problem at hand computed on each iteration of our stochastic sampling algorithm (or some other iterative stochastic search algorithm) are in fact at the extreme when the overall solution-space is considered. If you were to sample solutions uniformly at random, the probability is very low that you would find any of the solutions generated by WHISTLING with a strong heuristic. In other words, good solutions to any given problem instance from the class of problems we are most interested in are, in a sense, rare phenomena within the space of feasible solutions.

Consider for example the QDF illustrated previously in Figure 6.1 (a) for the solution-space of a weighted tardiness scheduling instance with loose duedates. The solution space for this instance has a mean objective value of $4.0935 * 10^5$ and standard deviation of

[2]This assumes a minimization problem with a lower bound of 0 on the value of the objective function.
[3]Assuming we are minimizing an objective function.

$6.3589 * 10^4$. The optimal solution to this problem instance is 0 – over 6.4 standard deviations better than the mean solution in the problem-space. The mean objective value of single iteration solutions generated by WHISTLING on this problem instance is 276 – also over 6.4 standard deviations better than the mean solution of the problem space. The optimal solution to this problem instance is "rare" considering the problem space, as are single iteration solutions generated by WHISTLING. And this is an "easy" problem instance.

Now consider a harder problem instance with QDF illustrated in Figure 6.1 (b). This problem instance has much tighter duedates. The solution space for this problem instance has a mean objective value of $9.0140 * 10^5$ and standard deviation of $5.4138 * 10^4$. We do not know the optimal solution to this problem instance, but the best known solution has objective value equal to 389354 which is over 9.4 standard deviations better than the mean solution in the problem-space. The mean objective value of single iteration solutions found by WHISTLING is 407480 which is over 9.1 standard deviations better than the mean solution in the problem space.

This is far from a proof that the solutions we are sampling with WHISTLING and other stochastic sampling algorithms are in fact rare phenomena in terms of the problem space, but it should serve to illustrate that the AQDFs we are modeling represent distributions of solution values sampled from the extremes of the solution space. With this noted, it makes sense to turn to the field of extreme value theory, which concerns itself with "techniques and models for describing the unusual rather than the usual" [50]. Specifically, we shall turn to an extreme value analog to the central limit theory. Consider, $M_n = \max\{X_1, \ldots, X_n\}$ where $X_1, \ldots, X_n$ is a sequence of independent random variables having a common distribution function F. For example, perhaps the X_i represent the mean temperatures for each of the 365 days in the year, then M_n would correspond to the annual maximum temperature. To model M_n, extreme value theorists turn to the *extremal types theorem* [50]:

Theorem 1 *If there exists sequences of constants* $\{a_n > 0\}$ *and* $\{b_n\}$ *such that* $P((M_n - b_n)/a_n \leq z) \rightarrow G(z)$ *as* $n \rightarrow \infty$*, where* G *is a non-degenerate distribution function, then* G *belongs to one of the following families:*

I: $G(z) = \exp(-\exp(-(\frac{z-b}{a}))), -\infty < z < \infty$

II: $G(z) = \exp(-(\frac{z-b}{a})^{-\alpha})$ *if* $z > b$ *and otherwise* $G(z) = 0$

III: $G(z) = \exp((\frac{z-b}{a})^{\alpha})$ *if* $z < b$ *and otherwise* $G(z) = 1$

for parameters $a > 0$*, b and in the latter two cases* $\alpha > 0$.

These are known as the extreme value distributions with types I (Gumbel), II (Fréchet), and III (Weibull). These distributions are commonly reformulated into the generalization known as the generalized extreme value distribution (GEV):

$$G(z) = \exp(-(1 + \xi(\frac{z-b}{a}))^{-1/\xi}) \tag{7.13}$$

where $\{z : 1 + \xi(\frac{z-b}{a}) > 0\}$, $-\infty < b < \infty$, $a > 0$, and $-\infty < \xi < \infty$. The case where $\xi = 0$ is treated as the limit of $G(z)$ as ξ approaches 0 to arrive at the Gumbel distribution. Under the assumption of Theorem 1, $P((M_n - b_n)/a_n \leq z) \approx G(z)$ for large enough n which is equivalent to $P(M_n \leq z) \approx G((z - b_n)/a_n) = G^*(z)$ where $G^*(z)$ is some other member of the generalized extreme value distribution family.

Following from the preceding argument regarding the rarity of the solutions produced on each iteration of WHISTLING with a strong heuristic within the larger solution space, I argue that the GEV distribution is a sensible assumption to make regarding the distribution of samples represented by an AQDF. Theorem 1 only explicitly applies to modeling the distribution of "block maxima". The assumption I now make is that the AQDF for a stochastic sampling algorithm using a strong heuristic behaves the same as (or at least similar to) the distribution of block maxima and thus that the cumulative distribution function of the AQDF can be modeled by the GEV distribution.

To use the GEV for this modeling, we must first recognize that we have been assuming throughout this thesis that we are interested in minimizing our objective function so we need a "block minima" analog to Equation 7.13. Let $M'_n = \min\{X_1, \ldots, X_n\}$. We now want $P(M'_n < z)$. Let $M''_n = \max\{-X_1, \ldots, -X_n\}$. Therefore, $M'_n = -M''_n$ and $P(M'_n < z) = P(-M''_n < z) = P(M''_n > -z) = 1 - P(M''_n \leq -z)$. Therefore, assuming that the distribution function associated with an AQDF behaves according to a GEV distribution, we have that the probability P_i of finding a better solution than the best found so far (B) can be defined as:

$$P_i = 1 - G_i(-B) = 1 - \exp(-(1 + \xi_i(\frac{-B - b_i}{a_i}))^{-1/\xi_i}) \tag{7.14}$$

where the b_i, a_i, and ξ_i are estimated from the negative of the sample values. To compute these parameters within the QD-BEACON framework, we use Hosking's maximum-likelihood estimator of the GEV parameters [109, 138, 110]. The implementation of this algorithm available through StatLib [110] has been used. The original code is in Fortran but

Algorithm 7.4: Using the Generalized Extreme Value Distribution within QD-BEACON
Input: A heuristic, a bias function, and the quality of the solution just obtained
Output:
Description: Updates the N_i, X_i, Y_i, P_i, $S_{i,j}$, and the GEV parameters appropriately. Note that it assumes smaller objective values, or qualities, are better. BestSolution is the global best solution found so far, the value of which persists across calls to UPDATEAQDF().

UPDATEAQDF(Heuristic,BiasFunction,Quality)

(1) $i \leftarrow \{\text{Heuristic}, \text{BiasFunction}\}$
(2) $N_i \leftarrow N_i + 1$
(3) $X_i \leftarrow X_i + \text{Quality}$
(4) $Y_i \leftarrow Y_i + \text{Quality}^2$
(5) add Quality to $S_{i,j}$
(6) $\mu_i \leftarrow \frac{X_i}{N_i}$
(7) $\sigma_i \leftarrow \sqrt{\frac{Y_i - N_i \mu_i^2}{N_i - 1}}$
(8) $Q_i \leftarrow$ interquartile range of $S_{i,j}$
(9) $\{b_i, a_i, \xi_i\} \leftarrow$ MLEOFGEV(μ_i,$\min(\sigma_i, Q_i)$,$-1 * S_{i,j}$)
(10) **if** Quality $<$ BestSolution
(11) BestSolution $\leftarrow$ Quality
(12) **foreach** $j =$ heuristic/bias function pair
(13) $P_j \leftarrow 1 - \exp(-(1 + \xi_j(\frac{-\text{BestSolution} - b_j}{a_j}))^{-1/\xi_j})$
(14) **else**
(15) $P_i \leftarrow 1 - \exp(-(1 + \xi_i(\frac{-\text{BestSolution} - b_i}{a_i}))^{-1/\xi_i})$

has been converted to C for our purposes using the "f2c"[4] fortran-to-c utility of Bell labs. The GEV version of QD-BEACON's UPDATEAQDF(Heuristic,BiasFunction,Quality) is shown in Algorithm 7.4. It calls MLEOFGEV(μ,σ,S) to compute a maximum likelihood estimate of the GEV parameters. The MLEOFGEV(μ,σ,S) is described by Algorithm 7.5. In computing a maximum likelihood estimate of the GEV parameters, Hosking's algorithm is called multiple times if necessary. The first call uses initial estimates of the parameters as recommended by Hosking (set assuming a Gumbel distribution). If Hosking's algorithm fails to converge, then a few additional calls are made with random initial values of the parameters. If convergence still fails, MLEOFGEV(μ,σ,S) returns the values of the parameters as estimated by assuming a type I extreme value distribution (the Gumbel distribution).

[4] http://netlib.bell-labs.com/netlib/f2c/

Algorithm 7.5: Maximum Likelihood Estimation of the parameters of the Generalized Extreme Value Distribution
Input: The sample mean and standard deviation. The list of data-points S to use for the estimation
Output: The maximum likelihood estimates of b, a, and ξ
MLEOFGEV(μ,σ,S)
(1) $\tilde{a} \leftarrow \frac{\sigma\sqrt{6}}{\pi}$
(2) $\tilde{b} \leftarrow \mu - 0.5772\tilde{a}$
(3) $\tilde{\xi} \leftarrow 0$
(4) $\{\tilde{a}, \tilde{b}, \tilde{\xi}\} \leftarrow$ HOSKING(S, $\tilde{a}$, $\tilde{b}$, $\tilde{\xi}$, ResultCode)
(5) **if** ResultCode is Success
(6) **return** $\{\tilde{a}, \tilde{b}, \tilde{\xi}\}$
(7) **else**
(8) **for** 1 **to** 5
(9) $\tilde{a} \leftarrow$ random value
(10) $\tilde{b} \leftarrow$ random value
(11) $\tilde{\xi} \leftarrow$ random value
(12) $\{\tilde{a}, \tilde{b}, \tilde{\xi}\} \leftarrow$ HOSKING(S, $\tilde{a}$, $\tilde{b}$, $\tilde{\xi}$, ResultCode)
(13) **if** ResultCode is Success
(14) **return** $\{\tilde{a}, \tilde{b}, \tilde{\xi}\}$
(15) $\tilde{a} \leftarrow \frac{\sigma\sqrt{6}}{\pi}$
(16) $\tilde{b} \leftarrow \mu - 0.5772\tilde{a}$
(17) $\tilde{\xi} \leftarrow 0$
(18) **return** $\{\tilde{a}, \tilde{b}, \tilde{\xi}\}$

7.3.4 Illustrative Comparison of AQDF Estimation Methods

In this Subsection, we take a look at the strengths and weaknesses of each of the three AQDF estimation methods presented above. Specifically, we consider a single problem instance from a set of benchmark instances for the weighted tardiness scheduling problem. This benchmark set and a comparison of algorithms applied to the problems of the set will actually be discussed in detail in a later Chapter. For now, it is sufficient to know that the problem is one of minimization and that the current best known solution to this problem instance has an objective value of 425875. In this analysis, we consider two algorithms that will be referred to as "Algorithm 1" and "Algorithm 2". All that is necessary to know about these two algorithms is that they are both iterative stochastic search algorithms and that Algorithm 1 in general tends to produce better solutions than Algorithm 2. Although at the individual problem instance level, it may not be known if this latter assumption is true a priori.

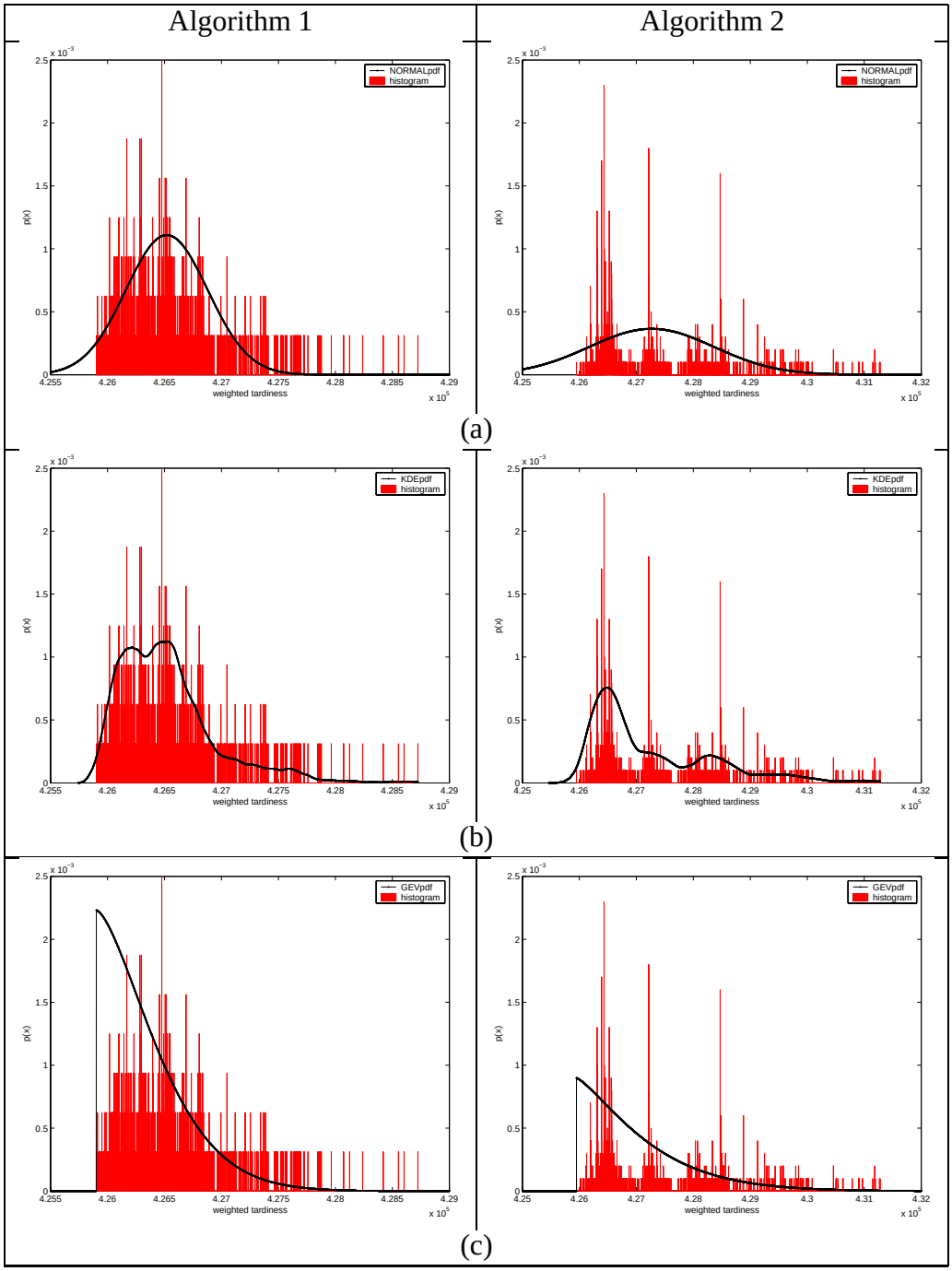


Figure 7.2: Comparison of AQDF estimation methods for an instance of a weighted tardiness scheduling problem and for two iterative stochastic search algorithms referred to simply as “Algorithm 1” and “Algorithm 2”. The estimation methods shown are: (a) Normal distribution, (b) Kernel Density Estimator, and (c) GEV distribution. Each estimation method is superimposed on a histogram estimate.

Several graphs are shown in Figure 7.2. The histogram in each graph is an estimate of the AQDF for the indicated algorithm and 1000 iterations. In Figure 7.2 (a), the Normal distribution estimates of these AQDFs are superimposed upon the histograms. Likewise, in Figure 7.2 (b), are shown the kernel density estimates of the AQDFs; and in Figure 7.2 (c), are shown the GEV distribution estimates. For the Normal distribution estimates, we see that there is a left-hand tail for which a significant density is to the left of the best solutions found by the respective algorithms. Also, of note, is that neither AQDF appears very much like a Normal distribution when viewing the histogram estimates. For the kernel density estimates, we see that the right-hand tail appears to capture the behavior of the right-hand-side of the AQDF better than did the Normal estimates. We, however, are more interested in the behavior of the left-hand-side. This too appears to be a better estimation using the KDE given that for both algorithms, the left-hand tail trails off to 0 not far to the left of the histogram estimate. Essentially, the KDE does not risk putting too much density in regions for which it has no data points.

Finally, for the GEV distribution estimates, we find that for both algorithms, the AQDF drops completely off to zero at some bound. For an algorithm that produces solutions very close to the optimal solution for the given problem instance, this is how we should intuitively expect the AQDF to behave. After all, the AQDF must be bounded below by whatever is the optimal solution. And if the algorithm under analysis produces a high density of solutions near this optimal solution, then there must be a sharp drop-off in the AQDF near this bound. It is not known if the best known solution of 425875 is optimal for this problem instance, but it is the best solution found by a large number of search algorithms available in the literature. The best solution found by 1000 iterations of Algorithm 1 is 425903 and by Algorithm 2 is 425950. And overall, the AQDFs of these algorithms appear to behave as if they are producing solutions near some theoretical bound in solution quality – thus making the resulting GEV estimate seem reasonable.

In Figure 7.3 we see, for each of the two algorithms, a graph showing a comparison of all three estimation methods. For each algorithm and for the GEV distribution estimates, we see a sharp peak near the bound of each GEV. For the KDE, we see significant density to the left of the mean of the AQDF (see Normal for position of mean), but most of this density in the KDE is to the right of the bound seen in the GEV with a relatively small amount of probability density to the left of this bound. The KDE appears to behave much like the GEV, but at the same time tails off to the left at a slower rate. For the Normal estimates, we see significant probability density in the left-hand tail. In fact, there is a

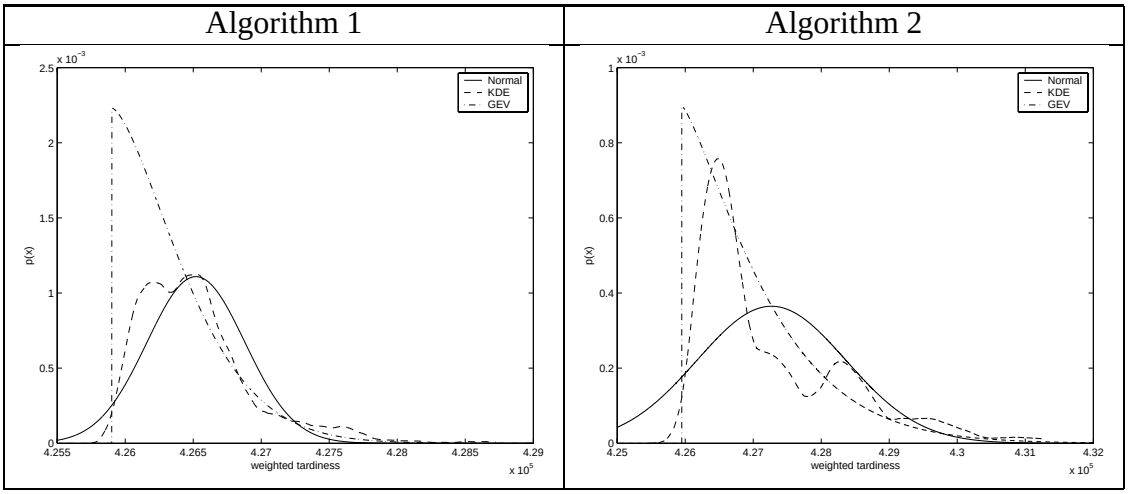


Figure 7.3: Comparison of AQDF estimation methods for an instance of a weighted tardiness scheduling problem and for two iterative stochastic search algorithms referred to simply as "Algorithm 1" and "Algorithm 2". For each algorithm, all three estimation methods are compared on a single graph.

Table 7.1: Comparison of AQDF estimation methods for an instance of a weighted tardiness scheduling problem and for two iterative stochastic search algorithms referred to simply as "Algorithm 1" and "Algorithm 2".

Algorithm	Estimator	best found	$P(x < 425903)$	$P(x < 425875)$	$P(425875)$
Algorithm 1	Normal	425903	0.0431	0.0365	0.00022
	KDE	425903	0.0100	0.0053	0.00013
	GEV	425903	0.0045	0.0000	0.00000
Algorithm 2	Normal	425950	0.1060	0.1014	0.00016
	KDE	425950	0.0076	0.0055	0.00006
	GEV	425950	0.0000	0.0000	0.00000

significant amount of probability density to the left of the best known solution. This seems unlikely to be the case in reality for these AQDFs given the large number of state-of-the-art algorithms that went into the production of that best known solution.

In Figure 7.4 we see, for each of the three estimation methods, a graph showing a comparison of the AQDFs of the two algorithms for this problem instance. For the Normal distribution estimates, we see that although Algorithm 1 appears to find a high density of solutions near the current best known solution, Algorithm 2 has an AQDF with a much higher density given to the region to the left of this current best known solution due to the high standard deviation in the solution qualities produced by Algorithm 2. As just discussed, this is unlikely to be the case in reality for this problem instance. Therefore,

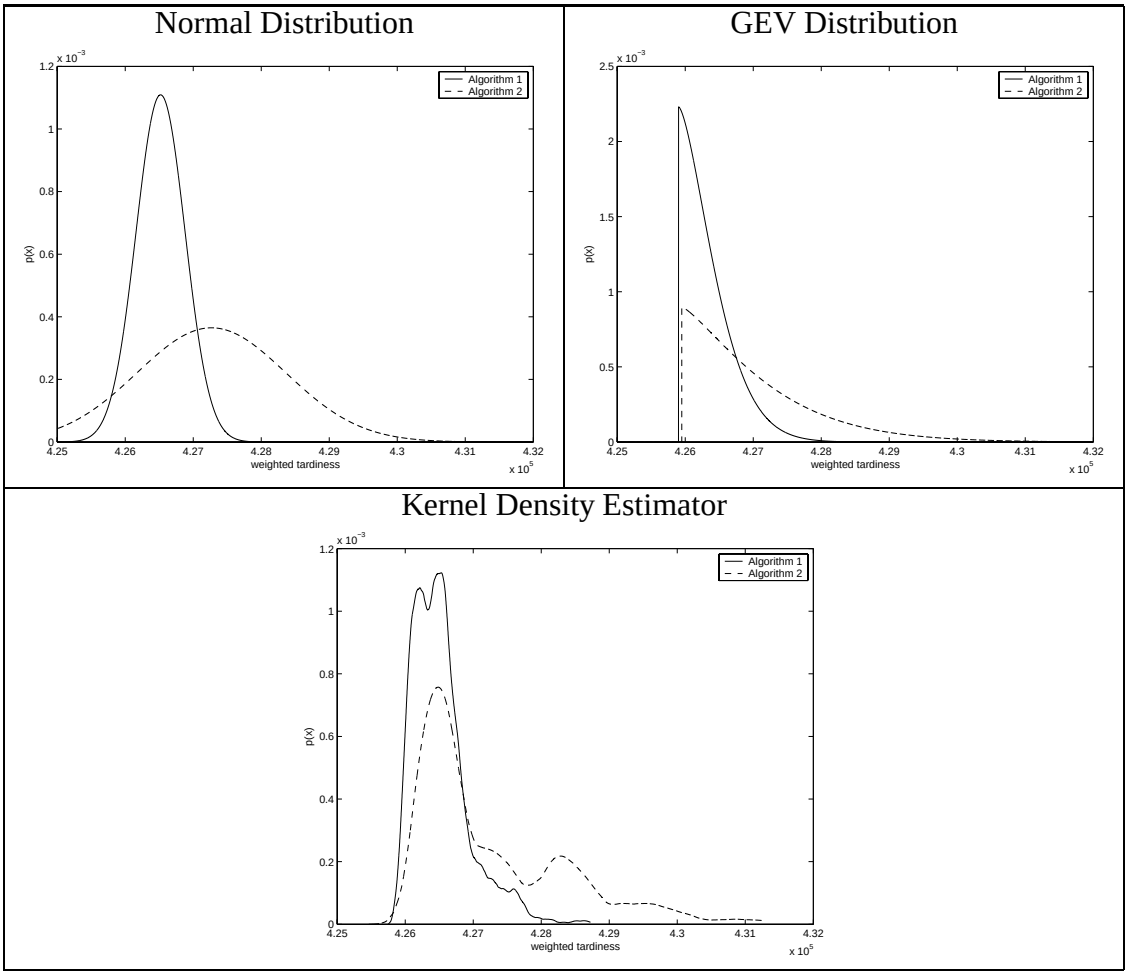


Figure 7.4: Comparison of AQDF estimation methods for an instance of a weighted tardiness scheduling problem and for two iterative stochastic search algorithms referred to simply as "Algorithm 1" and "Algorithm 2". For each estimation method, the AQDF of both algorithms are shown on the same graph.

for this problem instance, there appears to be a good chance that using Normal distribution estimates will likely lead the search astray by recommending heavier use of Algorithm 2 as compared to Algorithm 1. Both the GEV and KDE estimates have a sharp peak in the density near some bound for Algorithm 1. Likewise, they have sharp (but not as sharp) peaks for Algorithm 2 slightly to the right of the Algorithm 1 peaks. This may indicate that both KDE and GEV would lead to a recommendation of following Algorithm 1 more frequently.

We will explore this further by looking closer at what exactly the AQDF estimates tell us numerically. Table 7.1 shows a number of things for each of the two algorithms and each

of the three estimators: 1) best found solution by the algorithm; 2) probability of improving upon the best solution found by either of the algorithms (425903 found by Algorithm 1); 3) probability of finding a solution better than the current best known solution (425875); and 4) probability of finding a solution with quality equal to the current best known. First, note that the Normal estimates seem to be far too overly optimistic, especially for Algorithm 2, giving a probability greater than 0.1 of finding a solution better than the current best known solution on any single iteration of the algorithm ($P(x < 425875)$). Next, note that the GEV distribution estimates might possibly be too pessimistic. Specifically, observe that the GEV estimate gives a probability of 0 ($P(x < 425903)$) to the event of Algorithm 2 improving upon the best solution found by either of the two algorithms. The kernel density estimator perhaps offers a nice mix of the behaviors of these two other methods. It neither appears overly optimistic nor overly pessimistic. It will be interesting to compare the performance of these methods in greater detail in specific problem domains later in this thesis.

7.4 Exploration versus Exploitation

Within the QD-BEACON framework, a method is needed for balancing the tradeoff between exploiting the current estimates of the solution qualities given by the algorithm choices and the need for exploration to improve these estimates. To develop this strategy, we first turn in Section 7.4.1 to Holland's analysis of the k-Armed Bandit Problem and his use of it as demonstration of the near-optimal tradeoff of exploration/exploitation within the genetic algorithm. Then, in Section 7.4.2, we pose a new variation of the k-armed bandit problem that we call the "Max k-Armed Bandit Problem" and we detail a solution to it. Section 7.4.3 discusses the relevance of this solution to the QD-BEACON framework and presents the exploration strategy which it motivates.

7.4.1 The Two-Armed Bandit and K-Armed Bandit Problems

The *k-armed bandit problem* has been used by many as a theoretical analogy for the problem of balancing the tradeoff of exploration and exploitation in search problems [11, 101, 102, 114, 190]. It is the problem of allocating trials to the arms of a k-armed bandit (i.e., a slot machine with k arms, each with different and unknown pay-out distributions) with the goal of maximizing the expected reward over time. Many have presented analyses of various bandit problems (e.g., [11, 101, 102, 3, 4]) and others have used bandit problems

as inspiration for, or justification of, exploration strategies in application domains such as reinforcement learning (see [114, 190]) and genetic algorithms (see [101, 102]).

Our analysis in a later section of a newly posed variation of the k-armed bandit problem is inspired by Holland's analysis of the k-armed bandit and its connection with the genetic algorithm, so we begin by giving an overview of Holland's findings. The *two-armed bandit* and *k-armed bandit* problems are a major part of the theoretical underpinning of the genetic algorithm. Under certain assumptions, Holland demonstrates, using these bandit problems, that the GA achieves a near-optimal tradeoff of exploration and exploitation [101, 102]. Holland's analysis of the bandit problems will serve here as a basis for the design of an exploration policy within the QD-BEACON framework.

The two-armed bandit problem can be stated simply. Consider a slot machine with two arms. The reward for playing one of the arms is μ_1 with variance σ_1^2 and the reward for playing the other arm is μ_2 with variance σ_2^2. Furthermore, $\mu_1 \geq \mu_2$, but we do not know which arm is which. The problem is to maximize expected reward for a series of trials from this two-armed bandit. To solve the problem, it is necessary to determine the optimal tradeoff of exploratory actions (i.e., trying to discover the payoffs of the arms) versus exploitation actions (i.e., playing the arm that appears best). The k-armed bandit problem is the obvious generalization.

For the two-armed bandit problem, Holland showed that the optimal allocation of trials (in terms of minimizing expected loss from trials of the worse of the two arms) allocates n^* trials to the worse of the two arms, where n^* is:[5]

$$n^* \sim b^2 \ln(\frac{N^2}{8\pi b^4 \ln N^2}), \tag{7.15}$$

where N is the total number of trials to both arms and $b = \frac{\sigma_2}{\mu_1 - \mu_2}$. Therefore, the number of trials of the better of the two arms in the optimal allocation is:

$$N - n^* \sim \sqrt{8\pi b^4 \ln N^2} \cdot \exp(\frac{n^*}{2b^2}) - n^*. \tag{7.16}$$

As n^* increases, $\exp(\frac{n^*}{2b^2})$ dominates this equation so we can simplify this expression to:

$$N - n^* \sim \Theta(\exp(cn^*)), \tag{7.17}$$

where c is a constant. The important point is that the number of trials allocated to the

[5] See Holland for the complete derivation [101, 102].

observed better arm should increase exponentially with the number of trials allocated to the observed worse arm.

Holland's analysis of the k-armed bandit problem is significantly more complex. We now have k arms with expected payoffs, $\mu_1 \geq \mu_2 \geq \ldots \geq \mu_k$, and variances, σ_i for $i = 1, \ldots, k$. Holland's analysis showed that the worst-case expected loss for the problem occurs when $\mu_2 = \mu_3 = \ldots = \mu_k$ and $\sigma_2 = \sigma_3 = \ldots = \sigma_k$. Therefore, the best arm should be allocated $N - (k-1)m^*$ trials where N is the total number of trials and where each of the other $k-1$ arms are allocated m^* trials. The optimal number of trials m^* allocated to these worse arms is shown by Holland to be bound by:

$$m_l^* \sim b^2 \ln(\frac{N^2}{8\pi(k-1)^2 b^4 \ln N^2}) \tag{7.18}$$

and

$$m_u^* \sim m_l^* + 2b^2 \ln(k-1). \tag{7.19}$$

Therefore, the number of trials, $N-(k-1)m^*$, allocated to the observed best arm is bound by:

$$N - (k-1)m_u^* \sim \sqrt{8\pi(k-1)^2 b^4 \ln N^2} \cdot \exp(\frac{m_l^*}{2b^2} - \ln(k-1)) - (k-1)m_u^* \tag{7.20}$$

and

$$N - (k-1)m_l^* \sim \sqrt{8\pi(k-1)^2 b^4 \ln N^2} \cdot \exp(\frac{m_l^*}{2b^2}) - (k-1)m_l^*. \tag{7.21}$$

Again, as in the two-armed bandit case, the exponential in each of these bounds dominates and this can all be simplified to:

$$N - (k-1)m^* \sim \Theta(\exp(cm^*)). \tag{7.22}$$

The main point again is that the number of trials allocated to the observed best arm in the optimal allocation should increase exponentially with the number of trials allocated to each of the other $k-1$ arms.

7.4.2 The Max K-Armed Bandit Problem

The problem of choosing among competing heuristics for a stochastic sampling algorithm within the QD-BEACON framework is closely related to the k-Armed Bandit Problem.

On each iteration, QD-BEACON must choose from among a set of heuristics, each with different and unknown expected payoffs. The goal, however, is not quite the same as in the k-Armed Bandit Problem. In the k-Armed Bandit Problem, one wishes to maximize the sum of expected rewards over a series of trials. QD-BEACON's goal is a bit different. QD-BEACON concerns itself with the best single reward it receives over the series of trials. Assuming higher rewards are better, this leads us to pose the Max K-Armed Bandit Problem.

In the Max K-Armed Bandit Problem, we are faced with a series of N trials. In each trial we can choose any of k arms. For each of these arms there is an expected payoff according to some probability distribution. Our goal is to maximize the value of the *best* single reward received over the N trials.

We first detail a solution to the special case of 2-arms and then generalize to the k-armed case. Specifically, we show that to maximize the expected max single sample reward over N trials, the number of samples taken from the observed best arm should grow double exponentially in the number of samples taken from the observed second best. To make this derivation most relevant to the QD-BEACON framework, we assume that each of the arms is sampling at the extreme of some distribution. We specifically assume that each of the two arms is sampling from a Gumbel distribution – the type I extreme value distribution. Using the GEV, though it would be more general, would also unnecessarily complicate the analysis. The GEV special case of the Gumbel allows for an easily determined mean and standard deviation of the distribution in terms of the parameters of the distribution. There also exist straightforward estimators of these parameters from the sample mean and sample standard deviation which prove useful in the derivation.

Let us begin by stating that there are two arms, M_1 and M_2. The rewards of arm M_i are drawn from a Gumbel distribution $G_i(x)$ with location parameter b_i and scale parameter a_i. The mean reward of a single sample of arm M_i is:

$$\mu_i = b_i + 0.5772 a_i, \tag{7.23}$$

where 0.5772 is Euler's number, and the standard deviation is:

$$\sigma_i = \frac{a_i \pi}{\sqrt{6}}. \tag{7.24}$$

In order to fully state the problem, given that we want to maximize the expected largest single sample of a series of trials, we need an expression for the expected value of such a

max single sample of a series of trials. Given N samples $\{X_1, \ldots, X_N\}$ from a distribution, the probability that the maximum of these samples equals x is:

$$P(\max(X_i) = x) = N\, P(X = x)\, P(X \leq x)^{N-1}. \tag{7.25}$$

With our assumption of samples drawn from a Gumbel distribution, we have:

$$\begin{aligned} P(\max(X_i) = x) = \; & \tfrac{N}{a} \exp(-\tfrac{x-b}{a}) \exp(-\exp(-\tfrac{x-b}{a})) \\ & \exp(-(N-1)\exp(-\tfrac{x-b}{a})). \end{aligned} \tag{7.26}$$

This simplifies to:

$$P(\max(X_i) = x) = \frac{1}{a} \exp(-\frac{x - b - a\ln N}{a})\ \exp(-\exp(-\frac{x - b - a\ln N}{a})). \tag{7.27}$$

From this we see that the distribution of the max of N samples drawn from a Gumbel distribution with location parameter b and scale parameter a is also a Gumbel distribution with location parameter, $b_{\max} = b + a\ln N$ and scale parameter $a_{\max} = a$. Thus the expected max reward of N samples from each of the two arms in the problem is:

$$b_i + 0.5772a_i + a_i \ln N. \tag{7.28}$$

Consider that arm M_1 is the better of the two arms in the problem. This necessitates defining what we mean here by better. Specifically, let:

$$b_1 + 0.5772a_1 + a_1 \ln N > b_2 + 0.5772a_2 + a_2 \ln N \tag{7.29}$$

which implies that a_1 must be greater than or equal to a_2 or else for great enough N this inequality would fail to hold.

In the two-armed problem, where we do not know with certainty which arm is M_1 and which is M_2, the expected max reward if we had access to some omniscient observer that could tell us which was which, is clearly $b_1 + 0.5772a_1 + a_1 \ln N$ – the expected max reward of giving all N trials to the better arm. However, given that we cannot know with certainty which arm is which, some exploration is necessary. Consider that we draw n samples from the observed second best arm, and $N - n$ samples from the observed best arm. Now consider the loss of reward associated with sampling from the second best arm. There are two cases to consider:

1. The observed best arm is really the best arm. In this case the loss comes from giving n less samples to the best arm and we have an expected loss equal to: $a_1(\ln N - \ln(N-n))$.

2. The observed best arm is really second best. The loss in this case is a bit more complicated and depends on whether the expected value of giving $N-n$ samples to the second best arm is greater than giving n samples to the best arm. The mathematics is simplified if we assume that there is a higher expected value associated with taking n samples from the best arm as compared to $N-n$ samples from the second best. Also, as we will see later in this analysis, the probability of this second source of loss decreases exponentially with n. Furthermore, this form of loss is at a maximum when the expected value of the max of n samples of the best arm equals that of $N-n$ samples of the second best. With all of this said, we can now consider that the expected loss in this case is: $a_1(\ln N - \ln n)$.

Now let q be the probability that the observed best arm is really the second best arm. Therefore, $(1-q)$ is the probability that the observed best arm really is the best arm. The expected loss of sampling n times from the observed second best and $N-n$ times from the observed best arm, as a function of n is therefore:

$$l(N) = q(a_1(\ln N - \ln n)) + (1-q)(a_1(\ln N - \ln(N-n))). \tag{7.30}$$

This can be simplified to:

$$l(N) = q(a_1(\ln(N-n) - \ln n)) + a_1(\ln N - \ln(N-n)). \tag{7.31}$$

In order to select a value for n which minimizes the expected loss, we need to define q as a function of n. Let M_b be the arm that is perceived as best (i.e., the arm perceived to have the highest expected max single sample reward over a series of N trials) and M_w be the arm that is perceived as second best. The probability q can be stated as the probability that the expected max value of N samples of M_w is greater than the expected max value of N samples of M_b. If we note that the parameters of a Gumbel distribution can be estimated from the data by $\tilde{a} = \frac{s\sqrt{6}}{\pi}$ and $\tilde{b} = \bar{X} - 0.5772\tilde{a}$, where $\bar{X}$ and s are the sample mean and sample standard deviation, then we can define:

$$q(n) \;=\; P((\tilde{b}_b + 0.5772\tilde{a}_b + \tilde{a}_b \ln N) - (\tilde{b}_w + 0.5772\tilde{a}_w + \tilde{a}_w \ln N) < 0) \tag{7.32}$$

$$= P((\bar{X}_b + \frac{s_b\sqrt{6}}{\pi}\ln N) - (\bar{X}_w + \frac{s_w\sqrt{6}}{\pi}\ln N) < 0) \tag{7.33}$$

$$= P(\bar{X}_b - \bar{X}_w < (s_w - s_b)\frac{\sqrt{6}}{\pi}\ln N). \tag{7.34}$$

The central limit theorem says that $\bar{X}_b$ approaches a normal distribution with mean μ_b and variance $\frac{\sigma_b^2}{N-n}$. Similarly, $\bar{X}_w$ approaches a normal distribution with mean μ_w and variance $\frac{\sigma_w^2}{n}$. The distribution of $\bar{X}_b - \bar{X}_w$ is the convolution of the distributions $\bar{X}_b$ and $-\bar{X}_w$. The convolution of these distributions is by definition a normal distribution with mean $\mu_b - \mu_w$ and variance $\frac{\sigma_b^2}{N-n} + \frac{\sigma_w^2}{n}$. Using an approximation for the tail of a normal distribution, we can define $q(n)$ as:

$$q(n) \lesssim \frac{1}{\sqrt{2\pi}}\frac{\exp(-x^2/2)}{x} \tag{7.35}$$

where

$$x = \frac{(\mu_b - \mu_w) + \frac{\sqrt{6}}{\pi}\ln(N)(\frac{\sigma_b}{\sqrt{N-n}} - \frac{\sigma_w}{\sqrt{n}})}{\sqrt{\frac{\sigma_b^2}{N-n} + \frac{\sigma_w^2}{n}}}. \tag{7.36}$$

Given the above definitions for $q(n)$ and x, we can see that $q(n)$ decreases exponentially in n. This allows us to simplify x. Using the same simplification made by Holland, we can note that no matter the value for σ_b, there is a large enough N such that for n close to its optimal value, $\frac{\sigma_b^2}{N-n} \ll \frac{\sigma_w^2}{n}$ which leads to:

$$x \lesssim \frac{(\mu_b - \mu_w)\sqrt{n} - \frac{\sigma_w\sqrt{6}}{\pi}\ln N}{\sigma_w} \tag{7.37}$$

To select the value of n that will minimize the loss $l(n)$ we begin by taking the derivative of $l(n)$ with respect to n:

$$\frac{dl}{dn} = \frac{dq}{dn}(a_1(\ln(N-n) - \ln n)) - q(n)(\frac{a_1}{N-n} + \frac{a_1}{n}) + \frac{a_1}{N-n}, \tag{7.38}$$

where

$$\frac{dq}{dn} \lesssim -q(n)\frac{x^2+1}{x}\frac{dx}{dn}, \tag{7.39}$$

and

$$\frac{dx}{dn} \lesssim \frac{\mu_b - \mu_w}{2\sigma_w\sqrt{n}}. \tag{7.40}$$

The optimal value of n occurs when $\frac{dl}{dn} = 0$ so we can get a bound on the optimal n by

solving the following inequality:

$$0 \lesssim \frac{a_1}{N-n} - q(n)\frac{a_1 N}{N-n} - q(n)\frac{x^2+1}{x}\frac{dx}{dn}(a_1(\ln(N-n) - \ln n)). \tag{7.41}$$

We can collect the logarithmic terms on the left to obtain:

$$\ln(N-n) - \ln n \lesssim \frac{(1-q(n)N)x}{(N-n)q(n)\frac{dx}{dn}(x^2+1)} \tag{7.42}$$

Recalling that $q(n)$ decreases exponentially with n, $(1-q(n)N)$ rapidly approaches 1. Also noting that $\frac{x}{x^2+1} \lesssim \frac{1}{x}$, we arrive at:

$$\ln(N-n) - \ln n \lesssim \frac{1}{(N-n)q(n)\frac{dx}{dn}x} \tag{7.43}$$

We can substitute in the expressions for $q(n)$ and $\frac{dx}{dn}$.

$$\ln(N-n) - \ln n \lesssim \frac{\sigma_w\sqrt{8\pi}\sqrt{n}}{(N-n)(\mu_b - \mu_w)}\exp(\frac{(\mu_b - \mu_w - \frac{\sigma_w\sqrt{6}}{\pi\sqrt{n}}\ln(N))^2 n}{2\sigma_w^2}) \tag{7.44}$$

Exponentiating both sides of the inequality, we get:

$$N-n \lesssim \exp(\ln(n) + \frac{\sigma_w\sqrt{8\pi}\sqrt{n}}{(N-n)(\mu_b - \mu_w)}\exp(\frac{(\mu_b - \mu_w - \frac{\sigma_w\sqrt{6}}{\pi\sqrt{n}}\ln(N))^2 n}{2\sigma_w^2})) \tag{7.45}$$

You might now be troubled by the recursive expression for $N-n$. This might be especially the case if you note that though $N-n$ grows double exponentially in n, it also appears that $N-n$ decreases exponentially in itself which seemingly implies that $N-n$ decreases triple exponentially in n while increasing double exponentially in n. But, again, the triple exponential decrease is defined recursively. Thus, the decrease itself is decreased double exponentially and we no longer have a triple exponential decrease in n. The question that remains is which term dominates the expression. We can take the fraction involving $N-n$ up into the exponential and gain insight into the answer to this question:

$$N-n \lesssim \exp(\ln(n) + \exp(\frac{(\mu_b - \mu_w - \frac{\sigma_w\sqrt{6}}{\pi\sqrt{n}}\ln(N))^2 n}{2\sigma_w^2} + \ln(\frac{\sigma_w\sqrt{8\pi}\sqrt{n}}{(N-n)(\mu_b - \mu_w)}))). \tag{7.46}$$

Looking at this, we must now determine which part of the double exponential dominates

as the total number of samples N gets large. Consider the following limits:

$$\lim_{N\to\infty} \frac{(\mu_b - \mu_w - \frac{\sigma_w\sqrt{6}}{\pi\sqrt{n}} \ln(N))^2\, n}{2\sigma_w^2} = \infty \tag{7.47}$$

$$\lim_{N\to\infty} \ln\left(\frac{\sigma_w\sqrt{8\pi}\sqrt{n}}{(N-n)(\mu_b - \mu_w)}\right)) = -\infty \tag{7.48}$$

Note that the first expression is dominated by the $(\ln N)^2$ and that within the logarithm of the second expression, the N in the denominator dominates. For large enough N, it is sufficient enough to consider which of $(\ln N)^2$ and $\ln(1/N)$ dominates. Consider the following:

$$\begin{aligned}
\lim_{N\to\infty} \{(\ln N)^2 + \ln(1/N)\} &= \lim_{N\to\infty} \{(\ln N)^2 + \ln 1 - \ln N\} \\
&= \lim_{N\to\infty} \{\ln N\ (\ln N - 1)\} \\
&= \infty
\end{aligned} \tag{7.49}$$

Taking this into account and making a few other obvious simplifications, we can get:

$$N - n \sim \Theta(\exp(\exp(cn))) \tag{7.50}$$

This shows that the number of trials $N - n$ given to the observed best arm should grow double exponentially in n to maximize the expected max single sample reward.

We conjecture that in the Max K-Armed Bandit case, the result is asymptotically identical to the two-armed case. That is, the number of samples given the observed best arm should grow double exponentially in the number of samples given the other $k - 1$ arms. A weak argument for this leap from the result of the two-arm case to the k-arm case is as follows. The worst case loss in the k-armed case will occur when the $k - 1$ worst arms are identical (as was the case in Holland's analysis of the original k-armed bandit problem). With m^* trials given to each of these $k-1$ worse arms, the analysis of the k-armed case can be loosely thought of in terms of a special case of the analysis of the two-armed problem. Specifically, you have the observed best arm and a meta-arm comprised of the aggregation of the other $k-1$ arms. The meta-arm is given $n^* = m^*(k-1)$ trials uniformly distributed across the $k - 1$ arms. Since the $k - 1$ arms are identical in the worst case, the meta-arm should behave in the same way as the second best arm in the two-arm case. And thus, the number of samples $N - m^*(k-1)$ given the observed best arm should grow double

exponentially in $n^* = m^*(k-1)$.

Our analysis of the Max K-Armed Bandit problem (just like Holland's analysis of the original K-armed bandit) makes use of the central limit theorem and other similar assumptions. As such, the end result only directly applies in the limit and its relevance to finite models is tenuous. A finite-time analysis might be more appropriate, but would only directly apply if you knew beforehand the number of trials. In what follows, we use this analysis to provide motivation for exploration within QD-BEACON.

7.4.3 The QD-BEACON Exploration Strategy

In defining an exploration strategy for the QD-BEACON framework, we consider the analysis of the Max K-Armed Bandit Problem. To maximize the expected max single sample reward of a series of N trials, we should give a double exponentially increasing number of samples to the observed best arm relative to the number of samples given the other $k-1$ arms. Specific to our problem within QD-BEACON, this means sampling with the observed best heuristic with frequency increasing double exponentially relative to the number of samples given the other heuristics.

It should first be noted that in our analysis of the Max K-Armed Bandit, "observed best" meant the arm perceived to have the highest expected max of a series of N trials. To compute such expectations, the analysis had assumed that the output of an arm followed a type I extreme value distribution. As was seen, the distribution of the max of N samples from such an arm also follows a type I extreme value distribution. In fact, as we also earlier saw in the extremal types theorem, independent of the distribution of samples drawn from an arm, the distribution of the max of N samples follows a GEV distribution (i.e., one of the three types of extreme value distribution). However, only the type I (Gumbel) distribution leads to a closed form expression for the expected max of N samples, which is one of the reasons it was assumed in the analysis of the Max K-Armed Bandit.

Given this, we can now note a couple of things regarding the QD-BEACON framework. First, the distribution of the best of N samples of a stochastic sampling algorithm does not necessarily follow a type I extreme value distribution and may instead follow either a type II or type III extreme value distribution. Second, if it does follow a type II or type III extreme value distribution, then we cannot easily compute an expected value for the best of N samples. Third, we cannot easily (and certainly not efficiently) compute which of the extreme value distribution types the distribution of the best of N samples follows only given our models of the underlying distribution of samples. Fourth, the analysis of the Max

K-Armed Bandit concerns itself with what happens in the limit, while QD-BEACON is faced with a finite (and potentially unknown) number of trials.

These points, particularly the last, leads QD-BEACON to concern itself primarily with trying to improve upon the current best found solution by its choice of the next heuristic or stochastic search algorithm to apply to the problem. This choice considers the past trials and the next trial and does not attempt to use a projection out over the result of the future N trials. Its definition of the "observed best" arm is the heuristic (or algorithm) that has the highest estimated probability of improving upon the current best found solution. Incorporating this myopic definition of "observed best" into the analysis of the Max K-Armed Bandit would not allow for such a clean analysis (at best) as it would require a closed form expression for the convolution of two GEV distributions.

This discrepancy with the definition of "observed best" may be troublesome to some readers given that the exploration strategy of QD-BEACON is motivated by the analysis of the Max K-Armed Bandit. But keep in mind that the analysis is of the limit as the number of samples N goes to infinity; while the QD-BEACON exploration strategy assumes that the next sample might be the last sample. With this, it therefore makes more sense to consider the observed best to be the heuristic with the higher probability of improving upon the best solution found so far with a single sample, rather than attempting to project out the expected best of an ever-growing series of samples. This is what QD-BEACON does.

Consider, as the exploration strategy for the QD-BEACON framework, Boltzmann exploration as commonly used in reinforcement learning [114, 190] and within simulated annealing [118, 198]. With a Boltzmann exploration strategy, we would choose to use heuristic h_i with probability $P(h_i)$:

$$P(h_i) = \frac{\exp((P_i F_i)/T)}{\sum_{j=1}^{H} \exp((P_j F_j)/T)}, \tag{7.51}$$

where P_i is the probability of finding a solution better than the best found so far as defined previously for any of the methods of estimating AQDF i, where F_i is the ratio of the number of feasible solutions used in estimating P_i to the total number of samples with i, where there are H heuristics to choose from, and where T is a temperature parameter.

Now consider the number of samples expected to be given to the observed best heuristic

Algorithm 7.6: The QD-BEACON Exploration Strategy
Input:
Output: A heuristic / bias function pair $\{\text{Heuristic}, \text{BiasFunction}\}$.
GETHEURISTICBIASFUNCTIONPAIR()
(1) **if** MaxExplore $=$ true
(2) $i \leftarrow \arg\max_j P_j F_j$
(3) **return** $\{\text{Heuristic}_i, \text{BiasFunction}_i\}$
(4) **else**
(5) **foreach** $j =$ heuristic/bias function pair
(6) $W_j \leftarrow \exp((P_j F_j)/T)$
(7) **if** Any overflow conditions
(8) MaxExplore $\leftarrow$ true
(9) **if** MaxExplore $=$ true
(10) $i \leftarrow \arg\max_j P_j F_j$
(11) **return** $\{\text{Heuristic}_i, \text{BiasFunction}_i\}$
(12) **else**
(13) $T \leftarrow T * 0.5$
(14) **if** Underflow of T
(15) MaxExplore $\leftarrow$ true
(16) select i with probability weighted by W_i
(17) **return** $\{\text{Heuristic}_i, \text{BiasFunction}_i\}$

h_b relative to some other heuristic h_o:

$$\frac{N * P(h_b)}{N * P(h_o)} = \exp(\frac{P_b F_b - P_o F_o}{T}), \tag{7.52}$$

where N is the total number of samples. Since h_b is observed best, that means the probability of finding a better solution than the best found so far is greater for h_b than for h_o (i.e., $P_b F_b > P_o F_o$). This implies that we give an exponentially increasing in decreasing T number of samples relative to h_o. To get the double exponential sampling increase, we need to decrease T exponentially. For example, let $T = \exp(-N')$ where N' is the number of samples already taken and sample h_i with probability:

$$P(h_i) = \frac{\exp((P_i F_i)/\exp(-N'))}{\sum_{j=1}^{H} \exp((P_j F_j)/\exp(-N'))}. \tag{7.53}$$

Within QD-BEACON, we have actually chosen to increase the frequency of samples given the observed best heuristic slightly less than double exponentially. We use the fol-

lowing cooling schedule for T as an alternative:

$$T_0 = 1 \tag{7.54}$$

$$T_{N'} = T_{N'-1} * 0.5, \tag{7.55}$$

which is equivalent to $T = 0.5^{N'}$. The reasons for this slower than double exponential increase in the sampling of the observed best include:

1. An acknowledgment that in the analysis of the Max k-Armed Bandit we made a number of simplifying assumptions, including regarding the underlying distributions of the samples of the arms.

2. Cooling via this schedule can also potentially lead to a trivial improvement in implementation efficiency by allowing T to be adjusted via bit shifts if you instead maintain $T' = 1/T$ as an integer.

3. It allows for a longer search before we begin underflowing the denominator in the calculations of $\exp((P_i F_i)/T)$.

In the QD-BEACON framework, once T does reach the point of underflow (or the value of some $(P_i F_i)/T$ overflows, or the value of some $\exp((P_i F_i)/T)$ overflows), the framework begins to chose the heuristic for the next sample deterministically according to:

$$\arg\max_{h_i} P_i F_i, \tag{7.56}$$

where again P_i is the probability of heuristic h_i finding a solution better than the best found so far and F_i is the fraction of feasible solutions found. In other words, if the search lasts sufficiently long, QD-BEACON switches to a complete exploitation policy. This exploration policy for choosing which heuristic / bias function pair to use for the next iteration of the search is described in Algorithm 7.6.

7.5 Summary

In this Chapter we defined the functionality of the QD-BEACON framework. This includes functions that allow stochastic sampling algorithms to report the quality of solutions obtained on each iteration for a specific heuristic / bias function pair and which also allow for the reporting of infeasible solution nodes that are found. The latter is used in domains of

constrained optimization where not every iteration of a stochastic search algorithm necessarily leads to a feasible solution. It also includes functions that allow a stochastic sampling algorithm to choose which heuristic / bias function pair to use on future iterations based on the probability of finding a better solution than the best found so far. A modified version of WHISTLING that integrates the QD-BEACON functionality is presented as an example use of the framework.

Upon the reporting of solution quality, QD-BEACON uses one of several methods to update the appropriate AQDF estimate. The simplest model of an AQDF is that of a normal distribution. The next model discussed used kernel density estimation. Specifically, the KDE of an AQDF as presented here selects the value of the bandwidth h under the assumption that we are sampling from a type I extreme value distribution (i.e., a Gumbel distribution). The third model discussed, argued that stochastic sampling algorithms using strong heuristics, are likely to be sampling at the extreme of the problem instance's underlying solution space. This argument lead to motivation for the use of the generalized extreme value distribution as the model for the AQDF data. A comparison of these three models on a single instance of a scheduling problem using two stochastic search algorithms was then presented. Overall, this analysis showed that the Normal distribution seems to be an overly optimistic model for the AQDF, that the GEV might be overly pessimistic, and that the behavior of the KDE might offer a reasonable modeling methodology for the AQDF that is neither overly optimistic nor pessimistic.

Whichever model of the normal, KDE, or GEV, is used for the AQDF, some method of exploration / exploitation is needed if we desire to generate these models online while searching for the best solution we can find. Taking inspiration from Holland's analysis of the k-Armed Bandit Problem, we defined a new problem that we call the Max k-Armed Bandit Problem. Using an analysis of this Max k-Armed Bandit Problem, we showed that the optimal number of trials given the observed best arm should increase double exponentially in the number of trials given to the observed next best arm. This derivation became the motivation for the choice of Boltzmann exploration with an exponentially decaying temperature parameter – though mostly for implementation efficiency reasons, we ultimately decided upon a slightly less than exponentially decaying temperature parameter.

Chapter 8

Application: Sequencing to Minimize Weighted Tardiness (Revisited)

8.1 Overview

In this Chapter, we revisit the problem of sequencing a set of jobs on a single machine to minimize the weighted tardiness scheduling objective. The goal of this Chapter is to explore the benefit of combining multiple search heuristics within an iterative stochastic search algorithm using the QD-BEACON framework. Given a set of heuristics for the problem, each with its good and bad points, can QD-BEACON effectively combine them to enhance search performance as compared to using a single heuristic, as compared to using the naive strategy of sampling a uniform number of times with each heuristic, and as compared to the state-of-the-art for the problem? These are a few of the questions we explore in this Chapter.

The remainder of this Chapter is organized as follows. In Section 8.2 we present the problem's formalization. For the reader with Chapter 5 fresh in their mind, you may skip Section 8.2 provided that you note that in this Chapter we are considering the problem without sequence-dependent setups (i.e., $s_{i,j} = 0$ for all i, j). Section 8.3 overviews the state-of-the-art in problem solving for the weighted tardiness sequencing problem. In Section 8.4 we describe the benchmark problem sets used in this experimental comparison. Section 8.5 defines our performance criteria. In Section 8.6 we present a comparison of a number of local search algorithms with a variation of multistart dynasearch that has been enhanced using VBSS and the QD-BEACON framework. Then in Section 8.7 we further enhance the current best algorithm for the problem – iterated dynasearch – using the QD-

BEACON framework. Section 8.8 shows how the resulting algorithm compares to other local search algorithms available for comparison. We conclude the Chapter with a summary in Section 8.9.

8.2 Problem Formalization

The Weighted Tardiness Scheduling Problem is a sequencing problem. Specifically, we are given a set of jobs $J = \{j_1, \ldots, j_N\}$. Each of the N jobs j has a weight w_j, duedate d_j, and process time p_j. The particular instance of the problem that we concern ourselves with here is the case where the N jobs must be sequenced on a single machine and where preemption of a job during processing is not permitted. Furthermore, if a machine is processing a job, then it cannot do anything else until that operation is completed.

The objective of this problem is to sequence the set of jobs J on a machine to minimize the total weighted tardiness:

$$T = \sum_{j \in J} w_j T_j = \sum_{j \in J} w_j \max(c_j - d_j, 0), \tag{8.1}$$

where T_j is the tardiness of job j; and c_j, d_j is the completion time and duedate of job j. The completion time of job j is equal to the sum over the process times of all jobs that come before it in the sequence plus the processing time of job j itself. Specifically, let $\pi(j)$ be the position in the sequence of job j. We can now define c_j as:

$$c_j = \sum_{i \in J, \pi(i) <= \pi(j)} p_i \tag{8.2}$$

8.3 State-of-the-Art Solution Methods

In this Section, we overview a number of solution methods available in the literature for the weighted tardiness scheduling problem. These include the branch-and-bound procedure that was originally used to find optimal solutions to problem instances in the benchmark problem set, myopic dispatch policies for the problem, and a number of local search approaches. The final algorithm described in this Section is that of Iterated Dynasearch, which is currently the best known algorithm for the problem.

8.3.1 Branch-and-Bound

The optimal solutions that are available for most of the 40 and 50 job problem instances in the OR-Library were originally found by the branch-and-bound algorithm of Potts and van Wassenhove [156]. They were unable to tackle the 100 job problem instances due to the computational complexity of the problem. Prior to this branch-and-bound algorithm, no previous approach had been applied to problems with greater than 20 jobs. Since this branch-and-bound algorithm (despite its age of 18 years), no new approach (other than heuristic, local search, and other non-systematic approaches) has been able to optimally solve (with guarantees) problems larger than 50 jobs. Narayan, Morton, and Ramnath claim that exact methods are impractical for weighted tardiness problems any larger and advocate the use of dispatch policies [147]. The lower bounding scheme of Akturk and Yildirim gives tighter lower bounds [2], but has not been used within a branch-and-bound algorithm since its appearance in the literature.

The branch-and-bound algorithm of Potts and van Wassenhove use the well-known dominance rules of Emmons to generate precedence relations to prune branches in the search tree (see [73]). The search strategy explores in a newest active node first order. Backward sequencing is assumed in the branching. That is, a search node at level l of the search tree consists of jobs sequenced in the last l positions. The sub-problems at level l are of the same form as the original problem but with l fewer jobs. In addition to the precedence relations, the algorithm further prunes search nodes by noting that, if in a sub-problem it is possible to sequence a job last so it has zero tardiness, then it should be sequenced last in the sub-problem. Further search-space pruning occurs by considering the final job sequences of pairs of search nodes. If two final sequences contain the same subset of jobs, then the one with the lower weighted tardiness of the subset of jobs is kept while the other is pruned. If the weighted tardiness of the subset of jobs is the same in these final sequences, then one of them is discarded. The lower bound for the first $N - l$ nodes of the sequence of a search node at level l is computed by solving a Lagrangian relaxation of the problem. See the original publication of this algorithm for the details of the relaxation and its solution [156].

8.3.2 Myopic Dispatch Policies

Recall from our earlier discussion of dispatch policies in Chapter 5 that dispatch policies are heuristic rules for choosing the next job to sequence based on the current local char-

acteristics of the problem. There are a few dispatch heuristics in the OR literature for the problem with a well-grounded basis. These include:

- WSPT: The weighted shortest process time rule optimizes weighted tardiness when no job can possibly be scheduled earlier than its duedate [188, 145]. It is considered a good heuristic in highly constrained problems (i.e., most jobs cannot be scheduled earlier than their duedate). It is defined as:

$$\text{WSPT}_i = \frac{w_i}{p_i}. \tag{8.3}$$

- EDD: The earliest duedate rule is optimal if it is possible to sequence the jobs such that no job is tardy [145]. In the unweighted version of the problem, EDD optimizes total tardiness if at most one job is tardy [157]. EDD is considered a good rule for loosely constrained problems (i.e., problems where most jobs can be scheduled before their duedate). We define EDD here as:

$$\text{EDD}_i = \frac{1}{d_i} \tag{8.4}$$

- COVERT: The cost over time rule is a more complex dispatch policy that attempts to combine aspects of both EDD and WSPT [29, 145]. It is defined as:

$$\text{COVERT}_i(t) = \frac{w_i}{p_i}(1 - \frac{\max(0, d_i - p_i - t)}{kp_i}), \tag{8.5}$$

  where t is the current time, and k is a parameter that requires tuning. Usually k is set in some ad hoc way. Typical values of k are between 1 and 4 [145].

- R&M: The policy of Rachamadugu and Morton (sometimes referred to as "apparent urgency" or "apparent tardiness cost") is another attempt to combine aspects of EDD and WSPT [160, 146, 145]. It is defined as:

$$\text{R\&M}_i(t) = \frac{w_i}{p_i}\exp(-1 * \frac{\max(0, d_i - p_i - t)}{k\bar{p}}), \tag{8.6}$$

  where $\bar{p}$ is the average processing time, t is the current time, and k is again a parameter that requires tuning. Typical ad hoc values for k are again between 1 and 4.

Dispatch policy methods, due to their robustness, are particularly used in the dynamic variation of the problem in which jobs arrive dynamically and the problem is constantly changing. In the X-Dispatch methods of Narayan, Morton, and Ramnath, the R&M heuristic is extended to allow for the insertion of idle time between the processing of jobs in the dynamic case to handle the case where a particularly important job might arrive unexpectedly [147]. We do not consider the dynamic problem here.

When using one of these heuristics, the next job sequenced is the job with the highest value of the heuristic. Although the name EDD may imply smallest value of the heuristic (i.e., "earliest"), we have defined it as $1/d_i$ to allow for choosing the job with the max of the heuristic values independent of the heuristic.

Potts and van Wassenhove suggest using a strategy that sequences the jobs four times, once with each of the four above heuristics (EDD, WSPT, COVERT, R&M), and takes the best solution of the four [157]. Later, we will take this idea further and use these four heuristics in conjunction with the QD-BEACON framework and stochastic sampling.

8.3.3 Local Search

Though there currently only exist a very few complete optimal algorithms for the weighted tardiness sequencing problem, in recent years there have been a number of successful applications of local search algorithms to the problem. Crauwels, Potts, and Van Wassenhove compare several such local search methods [59]. The local search approaches considered in their study include:

- Strict Descent (D): Allows only operations that improve the objective value. Uses a permutation representation in which pairwise interchanges of jobs in the sequence are the allowed moves. As an alternative, also considered is a binary encoding of the problem with single bit flips as the local operator.

- Descent, neutral moves allowed (DN): Same as D, but which allows local moves to solutions with the same objective value.

- Simulated Annealing (SA): Same representation alternatives and operator sets as above.

- Threshold Accepting (TA): Same representation alternatives and operator sets as above.

Algorithm 8.1: Multistart Dynasearch for Weighted Tardiness Sequencing
Input: Number of restarts I; An instance of the weighted tardiness sequencing problem W.
Output: A solution S.
MULTISTART-DYNASEARCH(I, W)
(1) bestsofar $\leftarrow$ empty solution
(2) **for** 1 **to** I
(3) $\sigma \leftarrow$ random sequence of the jobs j_i in W
(4) stop $\leftarrow$ false
(5) **while** not stop
(6) compute the recursion $F(\sigma_k)$ for $k = 0, \ldots, N$ (use Equation 8.9)
(7) use $F(\sigma_k)$ for $k = 0, \ldots, N$ to compute next sequence σ
(8) **if** σ did not change (is a local optima)
(9) stop $\leftarrow$ true
(10) **if** σ is better than bestsofar
(11) bestsofar $\leftarrow \sigma$
(12) **return** bestsofar

- Tabu Search (TS): Same representation alternatives and operator sets as above.
- Genetic Algorithm (GA): Uses the binary encoding only. They claim they also considered a GA with a permutation encoding and appropriate operators, but do not report on those results.

In the results later in this Chapter, these algorithms will be referred to by the abbreviations given above. In square brackets after the abbreviation will be either a "P" or a "B" indicating whether the results shown used the permutation representation or a binary representation (e.g., D[P] will refer to strict descent using the permutation representation).

8.3.4 Iterated Dynasearch

The current best performing algorithm for the weighted tardiness scheduling problem is arguably the *Iterated Dynasearch* of Congram, Potts, and van de Velde [51]. It too falls into the category of local search, but has been singled out here due to its particular success. Also, later in this Chapter, we will consider the use of VBSS as a method of generating initial solution states for further improvement by Dynasearch.

To describe the Iterated Dynasearch algorithm, we begin by describing *Dynasearch*. Dynasearch is essentially a local hill-climber. Its name comes from its local operator set – or more accurately, its name comes from the method used to compute that operator set. A

Algorithm 8.2: Iterated Dynasearch for Weighted Tardiness Sequencing
Input: Number of kicks I; Kick length α; Start from best every β kicks; An instance of the weighted tardiness sequencing problem W; a "heuristic" to generate initial solution.
Output: A solution S.

ITERATED-DYNASEARCH(I, α, β, W, "heuristic")
(1) bestsofar $\leftarrow$ jobs j_i in W sequenced according to heuristic
(2) $\sigma \leftarrow$ bestsofar
(3) **for** $k = 1$ **to** I
(4) stop $\leftarrow$ false
(5) **while** not stop
(6) compute the recursion $F(\sigma_k)$ for $k = 0, \ldots, N$ (use Equation 8.9)
(7) use $F(\sigma_k)$ for $k = 0, \ldots, N$ to compute next sequence σ
(8) **if** σ did not change (is a local optima)
(9) stop $\leftarrow$ true
(10) **if** σ is better than bestsofar
(11) bestsofar $\leftarrow \sigma$
(12) **if** $k \bmod \beta = 0$
(13) $\sigma \leftarrow$ bestsofar
(14) **for** 1 **to** α
(15) compute random set of independent swaps and move to neighboring σ
(16) **return** bestsofar

common operator set used by local search algorithms for problems represented as permutations or sequences is pairwise swapping or pairwise interchanging. That is, commonly the changes in objective value that results from swapping each of the $O(n^2)$ pairs of jobs i, j is considered and the best such swap is taken. In Dynasearch, instead, the best set of independent swaps is made. In other words, a move in the search space of Dynasearch can make several swaps all at once. This set of independent swaps is found using dynamic programming. Let σ_i be the i-th job in the current sequence. Now initialize:

$$F(\sigma_0) = 0, \tag{8.7}$$

$$F(\sigma_1) = w_{\sigma_1} \max(0, p_{\sigma_1} - d_{\sigma_1}), \tag{8.8}$$

and use the recursion:

$$F(\sigma_k) = \min \left\{ \begin{array}{l} F(\sigma_{k-1}) + w_{\sigma_k} \max(0, P_{\sigma_k} - d_{\sigma_k}), \\ \min_{0 \le i \le k-2} \left\{ \begin{array}{l} F(\sigma_i) + w_{\sigma_k} \max(0, P_{\sigma_i} + p_{\sigma_k} - d_{\sigma_k}) \\ + \sum_{j=i+2}^{k-1} w_{\sigma_j} \max(0, P_{\sigma_j} + p_{\sigma_k} - p_{\sigma_{i+1}} - d_{\sigma_j}) \\ + w_{\sigma_{i+1}} \max(0, P_{\sigma_k} - d_{\sigma_{i+1}}) \end{array} \right\} \end{array} \right. \tag{8.9}$$

where $P_{\sigma_k} = \sum_{i=1}^{k} p_{\sigma_i}$. Once this recursion is computed, the next search state (or best sequence reachable by a set of independent swaps) is determined by backtracking. If $F(\sigma_N)$ was determined by the first term in the minimization, then job σ_N is not swapped and we proceed to determine how $F(\sigma_{N-1})$ was computed. Otherwise, $F(\sigma_N)$ was computed by the second term in the minimization for some index i and we swap jobs σ_N and σ_{i+1}, then proceed to determine how $F(\sigma_i)$ was computed and so forth. If no jobs are swapped by this procedure then we are at a local optima. Otherwise, we repeat the dynamic programming step to compute the next set of independent swaps.

Congram, Potts, and van de Velde then describe Iterated Dynasearch [51]. They do not use "iterated" to mean repeating the above described process from some other randomly chosen starting configuration. Instead, iterated dynasearch makes a series of $\alpha = 6$ randomly chosen sets of independent swaps beginning from the current local optima. They call this a "kick". Every $\beta = 6$ iterations, iterated dynasearch performs this kick on the best found so far local optima instead of on the current local optima. It then begins another iteration of dynasearch from the resulting sequence. The initial solution configuration of the very first iteration of iterated dynasearch is generated via some dispatch policy. Iterated Dynasearch can be seen in Algorithm 8.2.

Additionally, they describe an algorithm they call Multistart Dynasearch, which restarts Dynasearch at a randomly generated initial starting configuration whenever a local optima is reached as shown in Algorithm 8.1. In the multistart version of the algorithm, they do not use a heuristic to seed the starting configurations. They state that unbiased random starts are more effective, without presenting any evidence to this. Later in the results of this Chapter, we show their assumption to be false by using VBSS and the QD-BEACON framework to seed the starting solution configurations of a multistart version of dynasearch. This results in a significant improvement over the use of unbiased starts. Furthermore, Congram *et al.* had shown that Iterated Dynasearch performed better than the multistart version. However, using QD-BEACON, we are able to further improve upon the Iterated Dynasearch performance.

8.4 The Benchmark Problem Set

The problem sets that we use in this Chapter are from the OR-Library [7, 6]. There are three problem sets available in the OR-Library: 1) $N = 40$ jobs; 2) $N = 50$ jobs; and 3) $N = 100$ jobs. The integer process time p_j of a job is uniformly distributed in the interval $[1, 100]$. The integer weight w_j of a job is uniformly distributed in the interval $[1, 10]$. The integer duedate d_j is uniformly distributed in the interval $[N\bar{p}\frac{1-\text{TF}-\text{RDD}}{2}, N\bar{p}\frac{1-\text{TF}+\text{RDD}}{2}]$, where $\bar{p}$ is the average processing time, TF is the average tardiness factor, and RDD is the relative range of duedates. The problem instances generated consider RDD $= \{0.2, 0.4, 0.6, 0.8, 1.0\}$ and TF $= \{0.2, 0.4, 0.6, 0.8, 1.0\}$. For each of the 25 combinations of these two parameters, 5 problem instances are generated. This is done for all three numbers of jobs resulting in 125 instances with 40 jobs, 125 instances with 50 jobs, and 125 instances with 100 jobs. Optimal solutions are known for 124 of the 40 job problems and for 103 of the 50 job problems. For the 100 job problems and for the unsolved 40 and 50 job problems, current best known solutions are available in the OR-Library.

8.5 Performance Criteria

In the experimental results that follow, we use the following performance criteria:

- NO: The number of optimal (or best known) solutions out of 125 problem instances. For the 100 job problem set, the NO is the best known solutions prior to Congram *et al.*'s Iterated Dynasearch to allow for a consistent comparison with all of the solution methods currently available. So for the 100 job problem set, NO is the number of at least as good as previous best known solutions out of 125. There are 8 problem instances with 100 jobs out of 125 for which Congram *et al.* reports new best known solutions. In these 8 cases, our approach also finds these new best known, but does not further improve upon them.

- ARPD: The average relative percentage deviation of the solution value found by the algorithm from the optimal solution value (or best known). "na" is listed if divide by zero would occur.

- MRPD: The maximum relative percentage deviation of the solution value found by the algorithm from the optimal solution value (or best known). "na" is listed if divide by zero would occur.

Algorithm 8.3: QD-BEACON/VBSS Enhanced Multistart Dynasearch for Weighted Tardiness Sequencing
Input: Number of restarts I; An instance of the weighted tardiness sequencing problem W.
Output: A solution S.
VBSS-QD-BEACON-MULTISTART-DYNASEARCH(I, W)
(1) QD-BEACONINIT({EDD,WSPT,R&M,COVERT}, {Corresponding bias functions})
(2) bestsofar $\leftarrow$ best of the four solutions given by direct use of heuristics
(3) **for** 1 **to** I
(4) {heuristic, bias} $\leftarrow$ GETHEURISTICBIASFUNCTIONPAIR()
(5) $\sigma \leftarrow$ sequence of jobs in W obtained by 1 iteration of VBSS
(6) stop $\leftarrow$ false
(7) **while** not stop
(8) compute the recursion $F(\sigma_k)$ for $k = 0, \ldots, N$ (use Equation 8.9)
(9) use $F(\sigma_k)$ for $k = 0, \ldots, N$ to compute next sequence σ
(10) **if** σ did not change (is a local optima)
(11) stop $\leftarrow$ true
(12) UPDATEAQDF(heuristic, bias, objective(σ))
(13) **if** σ is better than bestsofar
(14) bestsofar $\leftarrow \sigma$
(15) **return** bestsofar

For each of these criteria, we present the average of 10 runs of the algorithm across all 125 problem instances of each given size. In the Tables of the following Sections, values in parentheses after the average indicate the best of the 10 runs.

8.6 Using VBSS and QD-BEACON to Enhance Multistart Dynasearch

The multistart version of dynasearch as described by Congram *et al.* begins each restart of the algorithm at an unbiased random starting configuration as discussed above. Congram *et al.* stated that there is no advantage to biasing the starting configurations but presented no evidence. In this Section, we consider using: 1) VBSS and one of the existing dispatch policies for the problem (EDD, WSPT, R&M, and COVERT) to bias the starting solutions of each restart; and 2) VBSS and QD-BEACON and all 4 of the heuristics to bias the starting solutions of each restart. The results contradict Congram *et al.*'s untested hypothesis.

Significant improvement over the results of Multistart Dynasearch can be made by using VBSS and QD-BEACON to bias the starting solution configurations. Algorithm 8.3 shows the enhanced multistart algorithm.

The bias function was tuned for each of these heuristics for a relatively small number of iterations for a set of bias functions chosen via the rationale presented in Chapter 4. The bias functions used by VBSS for each heuristic are: 1) EDD, v^4; 2) WSPT, v^1; 3) R&M, v^2; and COVERT, v^3. R&M and COVERT are both stronger heuristics as compared to WSPT, thus the stronger bias functions. EDD, as we have defined it, gives very small real values less than one in a relatively small range. This requires the stronger bias function to spread out the values used by VBSS for EDD in its roulette wheel decisions. The scale parameter k of the COVERT and R&M heuristics has been set to 3.0. QD-BEACON uses these same 4 heuristics and bias functions. We will refer to the various algorithms considered using the following abbreviations:

- EDD[N]: Using VBSS and the EDD heuristic to seed N restarts of multistart dynasearch.
- WSPT[N]: Using VBSS and the WSPT heuristic to seed N restarts of multistart dynasearch.
- COVERT[N]: Using VBSS and the COVERT heuristic to seed N restarts of multistart dynasearch.
- RM[N]: Using VBSS and the R&M heuristic to seed N restarts of multistart dynasearch.
- NAIVE[N]: Using VBSS to seed N restarts of multistart dynasearch ($N/4$ restarts with each of the 4 heuristics).
- NORM[N]: Using QD-BEACON/VBSS and all 4 heuristics with Normal distribution estimates of the AQDFs to seed N restarts of multistart dynasearch.
- KDE[N]: Using QD-BEACON/VBSS and all 4 heuristics with Kernel Density Estimates of the AQDFs to seed N restarts of multistart dynasearch.
- GEV[N]: Using QD-BEACON/VBSS and all 4 heuristics with GEV distribution estimates of the AQDFs to seed N restarts of multistart dynasearch.

- M-DYNA[N]: N restarts of multistart dynasearch using unbiased starting configurations as specified by Congram *et al.* originally.

The results of a comparison of these algorithms for the 40 job problems, 50 job problems, and 100 job problems can be found in Table 8.1, Table 8.2, and Table 8.3, respectively. We can make the following general observations about these results:

- M-DYNA[N], the original Multistart Dynasearch algorithm using unbiased random starting solution configurations, is the second worst algorithm considered in this comparison for any number of restarts N. It is only able to defeat WSPT[N] (Multistart Dynasearch with starting configurations seeded by VBSS and the WSPT heuristic) which does very poorly on average. Thus, Congram *et al.*'s previously untested hypothesis is falsified (i.e., there is a benefit to using heuristic knowledge to seed the starts of the multistart dynasearch algorithm).

- For the 40 job problem set (Table 8.1), combining multiple heuristics in the search is not at all necessary. Using VBSS with only the COVERT heuristic to seed the starting solution configurations performs best for each number of restarts considered; and for 400 or more restarts is able to consistently solve to optimality all 125 instances from the 40 job problem set.

- Turning to the 50 job problem set (Table 8.2), we see that for up to 200 restarts seeding the starting configurations with VBSS and COVERT again does best in terms of number of instances solved to optimality; while we now see that seeding the restarts with VBSS and EDD does best in terms of relative percentage deviation. There is one problem instance for which seeding the restarts with VBSS and COVERT does very poorly (see the high MRPD for COVERT[N] for any value of N). By combining the heuristics, we can also combine their problem-solving strengths. As you can see in the results for 400 or more restarts, the best strategy for combining heuristics is KDE[N]. It is the only strategy that is able to consistently solve to optimality all 125 problem instances with 50 jobs.

- Combining heuristics is most effective in solving the instances from the 100 job problem set (Table 8.3). In particular, using QD-BEACON with Kernel Density Estimates performs best in terms of the number of best known solutions found for all numbers of restarts considered and is only marginally second best in terms of relative percentage deviation. For 800 restarts or less, QD-BEACON using the GEV

Table 8.1: 40 Job Set: QD-BEACON/VBSS Enhanced Multistart Dynasearch vs the original Multistart Dynasearch. For each number of restarts, bold indicates the best in terms of number of optimal solutions; italics indicates the best in terms of percentage deviation.

Algorithm	NO	ARPD	MRPD
COVERT[100]	124 (125)	0.03 (0.00)	3.70 (0.00)
KDE[100]	123.3 (124)	0.011 (0.010	1.29 (1.29)
NORM[100]	122.7 (125)	0.007 (0.000)	0.88 (0.00)
GEV[100]	122.7 (123)	0.011 (0.011)	1.29 (1.29)
EDD[100]	121 (124)	0.034 (0.002)	3.21 (0.26)
RM[100]	116.3 (119)	0.13 (0.09)	9.77 (9.77)
M-DYNA[100]	112.7 (118)	0.38 (0.21)	10.58 (8.33)
WSPT[100]	89.3 (98)	1.38 (0.77)	51.36 (21.51)
COVERT[200]	124.7 (125)	0.008 (0.000)	0.96 (0.00)
KDE[200]	124 (125)	0.007 (0.000)	0.86 (0.00)
GEV[200]	123.3 (124)	0.007 (0.0005)	0.88 (0.06)
NORM[200]	123 (125)	0.007 (0.000)	0.88 (0.00)
EDD[200]	122.3 (124)	0.027 (0.002)	3.18 (0.26)
RM[200]	119 (121)	0.053 (0.016)	4.87 (1.29)
M-DYNA[200]	115.3 (121)	0.25 (0.11)	7.59 (6.70)
WSPT[200]	97 (105)	1.21 (0.71)	48.78 (16.86)
COVERT[400]	125 (125)	0.00 (0.00)	0.00 (0.00)
NAIVE[400]	125 (125)	0.00 (0.00)	0.00 (0.00)
KDE[400]	124.3 (125)	0.007 (0.000)	0.86 (0.00)
GEV[400]	124 (125)	0.007 (0.000)	0.86 (0.00)
NORM[400]	123.3 (125)	0.007 (0.000)	0.86 (0.00)
EDD[400]	123.3 (124)	0.008 (0.002)	0.95 (0.26)
RM[400]	120.3 (121)	0.023 (0.016)	1.86 (1.29)
M-DYNA[400]	119.3 (122)	0.13 (0.05)	7.25 (5.41)
WSPT[400]	101.3 (109)	1.07 (0.60)	46.49 (16.86)
NAIVE[800]	125 (125)	0.00 (0.00)	0.00 (0.00)
GEV[800]	124.7 (125)	0.0007 (0.0000)	0.086 (0.000)
KDE[800]	124.3 (125)	0.007 (0.000)	0.86 (0.00)
NORM[800]	123.7 (125)	0.007 (0.000)	0.86 (0.00)
M-DYNA[800]	121.3 (123)	0.07 (0.05)	5.93 (5.41)
NAIVE[1600]	125 (125)	0.00 (0.00)	0.00 (0.00)
GEV[1600]	125 (125)	0.00 (0.00)	0.00 (0.00)
KDE[1600]	124.3 (125)	0.007 (0.000)	0.86 (0.00)
NORM[1600]	123.7 (125)	0.007 (0.000)	0.86 (0.00)
M-DYNA[1600]	122.7 (124)	0.04 (0.02)	4.55 (2.84)

Table 8.2: 50 Job Set: QD-BEACON/VBSS Enhanced Multistart Dynasearch vs the original Multistart Dynasearch. For each number of restarts, bold indicates the best in terms of number of optimal solutions; italics indicates the best in terms of percentage deviation.

Algorithm	NO	ARPD	MRPD
COVERT[100]	119.7 (122)	1.08 (0.19)	125.89 (22.92)
NORM[100]	113.7 (122)	0.03 (0.001)	1.62 (0.10)
KDE[100]	111.7 (123)	0.059 (0.002)	2.74 (0.13)
GEV[100]	111 (122)	0.059 (0.002)	2.79 (0.10)
EDD[100]	109.7 (115)	0.022 (0.009)	1.19 (0.42)
RM[100]	105.3 (114)	0.20 (0.19)	22.92 (22.92)
M-DYNA[100]	89.3 (96)	0.49 (0.32)	12.10 (11.20)
WSPT[100]	73 (82)	na (1.24)	na (31.55)
COVERT[200]	121.3 (122)	1.08 (0.19)	125.89 (22.92)
KDE[200]	116.7 (125)	0.053 (0.00)	2.70 (0.00)
NORM[200]	116.7 (124)	0.03 (0.0008)	1.59 (0.10)
GEV[200]	114 (124)	0.059 (0.0008)	2.79 (0.10)
EDD[200]	112.3 (115)	0.013 (0.008)	0.54 (0.42)
RM[200]	110.3 (116)	0.20 (0.19)	22.92 (22.92)
M-DYNA[200]	95.7 (104)	0.33 (0.21)	12.05 (11.20)
WSPT[200]	78.7 (87)	na (1.00)	na (31.55)
KDE[400]	124.0 (125)	0.022 (0.00)	2.70 (0.00)
NAIVE[400]	123 (124)	0.0014 (0.0008)	0.11 (0.10)
COVERT[400]	121.7 (122)	1.07 (0.19)	125.89 (22.92)
NORM[400]	121.3 (124)	0.029 (0.0008)	1.59 (0.10)
GEV[400]	121.0 (124)	0.054 (0.0008)	2.72 (0.10)
EDD[400]	116.7 (120)	0.008 (0.006)	0.42 (0.42)
RM[400]	114.3 (118)	0.19 (0.19)	22.92 (22.92)
M-DYNA[400]	101.3 (108)	0.24 (0.16)	8.57 (7.26)
WSPT[400]	82 (92)	1.14 (0.92)	34.43 (31.55)
KDE[800]	124.7 (125)	0.007 (0.00)	2.67 (0.00)
NAIVE[800]	124 (124)	0.0008 (0.0008)	0.10 (0.10)
NORM[800]	123 (124)	0.029 (0.0008)	1.59 (0.10)
GEV[800]	122.7 (124)	0.054 (0.0008)	2.72 (0.10)
M-DYNA[800]	107.3 (114)	0.17 (0.08)	7.98 (4.89)
KDE[1600]	125 (125)	0.000 (0.000)	0.00 (0.00)
GEV[1600]	124 (125)	0.021 (0.000)	2.67 (0.00)
NAIVE[1600]	124 (124)	0.0008 (0.0008)	0.10 (0.10)
NORM[1600]	123.3 (124)	0.029 (0.0008)	1.59 (0.10)
M-DYNA[1600]	113.3 (118)	0.12 (0.06)	5.93 (4.89)

Table 8.3: 100 Job Set: QD-BEACON/VBSS Enhanced Multistart Dynasearch vs the original Multistart Dynasearch. For each number of restarts, bold indicates the best in terms of number of best known solutions; italics indicates the best in terms of percentage deviation.

Algorithm	NO	ARPD	MRPD
KDE[100]	79.7 (95)	0.18 (0.15)	11.28 (11.28)
GEV[100]	76.3 (93)	0.16 (0.10)	11.28 (11.28)
NORM[100]	76 (94)	0.19 (0.15)	11.28 (11.28)
COVERT[100]	70 (81)	na (na)	na (na)
RM[100]	69.3 (77)	0.22 (0.15)	13.51 (11.28)
EDD[100]	60.3 (70)	0.39 (0.29)	17.09 (11.28)
M-DYNA[100]	48.7 (60)	2.48 (1.61)	92.59 (61.20)
WSPT[100]	39 (45)	na (na)	na (na)
KDE[200]	88.7 (102)	0.16 (0.14)	11.28 (11.28)
NORM[200]	85 (102)	0.18 (0.14)	11.28 (11.28)
GEV[200]	82.7 (97)	0.13 (0.02)	10.07 (1.67)
COVERT[200]	76.7 (87)	na (na)	na (na)
RM[200]	75.3 (82)	0.20 (0.15)	13.51 (11.28)
EDD[200]	66 (78)	0.38 (0.28)	16.92 (11.28)
M-DYNA[200]	56.3 (68)	2.06 (1.34)	92.59 (61.20)
WSPT[200]	43.3 (47)	na (na)	na (na)
KDE[400]	94.3 (108)	0.12 (0.04)	10.07 (5.14)
NORM[400]	90.7 (106)	0.13 (0.02)	10.07 (1.79)
GEV[400]	89 (104)	0.12 (0.005)	10.07 (0.31)
NAIVE[400]	85 (94)	0.14 (0.14)	11.28 (11.28)
COVERT[400]	84 (93)	na (na)	na (na)
RM[400]	78.7 (84)	0.19 (0.15)	13.51 (11.28)
EDD[400]	73.7 (83)	0.26 (0.16)	9.75 (6.69)
M-DYNA[400]	62 (70)	1.66 (1.06)	76.18 (53.31)
WSPT[400]	46.7 (52)	na (na)	na (na)
KDE[800]	100.7 (111)	0.12 (0.04)	10.07 (5.14)
NORM[800]	97 (108)	0.123 (0.016)	10.07 (1.79)
GEV[800]	95.3 (110)	0.11 (0.001)	10.07 (0.05)
NAIVE[800]	89.3 (98)	0.14 (0.14)	11.28 (11.28)
M-DYNA[800]	68.3 (76)	1.29 (0.47)	74.28 (18.06)
KDE[1600]	107.3 (117)	0.11 (0.04)	8.47 (5.14)
NORM[1600]	104.7 (117)	0.07 (0.015)	5.69 (1.79)
GEV[1600]	100.3 (113)	0.10 (0.001)	10.07 (0.05)
NAIVE[1600]	95 (101)	0.12 (0.098)	9.75 (6.69)
M-DYNA[1600]	73.3 (84)	1.16 (0.46)	71.06 (18.06)

Table 8.4: Tracking of the number of samples allocated to each of the four heuristics by QD-BEACON using KDE on a single (100 job) problem instance.

COVERT	RM	EDD	WSPT
61	17	12	10
156	22	12	10
253	25	12	10
352	26	12	10
451	27	12	10
551	27	12	10
650	28	12	10
749	28	13	10
846	30	14	10
944	31	14	11

distribution does best in terms of relative percentage deviation; while QD-BEACON using Normal distributions does best in terms of relative percentage deviation for 1600 restarts.

In Table 8.4, we show a tracking of the number of samples allocated to each of the four heuristics by QD-BEACON using KDE on a single (100 job) problem instance. For this particular problem instance, QD-BEACON strongly favors using the COVERT heuristic as can be seen in the number of samples allocated to COVERT. However, the exploration strategy does not entirely abandon the other heuristics. In particular, the R&M heuristic is allocated a few samples throughout the run. EDD and WSPT, though they are given far fewer samples, are not totally abandoned (e.g., WSPT is given an additional sample on iteration 976 despite not being sampled since iteration 40).

A lesson that can be taken home from this study is that for the easier problems (e.g., 40 job instances), the multiple heuristic approach may not be necessary. Searching the neighborhood around a single heuristic may be sufficient. In this case, it was. As you increase problem difficulty (the 50 job problem set and 100 job problem set), there is an advantage to incorporating multiple heuristics within the search routine. It is for the most difficult problem instances (the 100 job problem set) that we gain the most benefit of using the QD-BEACON framework.

Algorithm 8.4: QD-BEACON Enhanced Iterated Dynasearch for Weighted Tardiness Sequencing
Input: Number of kicks I; Kick length α; Start from best every β kicks; An instance of the weighted tardiness sequencing problem W.
Output: A solution S.
QD-BEACON-ITERATED-DYNASEARCH(I, α, β, W)
(1) best[EDD] $\leftarrow$ jobs j_i in W sequenced according to EDD
(2) best[WSPT] $\leftarrow$ jobs j_i in W sequenced according to WSPT
(3) best[RM] $\leftarrow$ jobs j_i in W sequenced according to R&M
(4) best[COVERT] $\leftarrow$ jobs j_i in W sequenced according to COVERT
(5) bestsofar $\leftarrow$ best of best[EDD], best[WSPT], best[RM], best[COVERT]
(6) $\sigma^{EDD} \leftarrow$ best[EDD]
(7) $\sigma^{WSPT} \leftarrow$ best[WSPT]
(8) $\sigma^{RM} \leftarrow$ best[RM]
(9) $\sigma^{COVERT} \leftarrow$ best[COVERT]
(10) N[EDD] $\leftarrow 0$, N[WSPT] $\leftarrow 0$, N[RM] $\leftarrow 0$, N[COVERT] $\leftarrow 0$
(11) QD-BEACONINIT({EDD,WSPT,R&M,COVERT}, {bias functions not used})
(12) **for** $k = 1$ **to** I
(13) {heuristic, bias} $\leftarrow$ GETHEURISTICBIASFUNCTIONPAIR()
(14) N[heuristic] $\leftarrow 1+$ N[heuristic]
(15) stop $\leftarrow$ false
(16) **while** not stop
(17) compute the recursion $F(\sigma_k^{heuristic})$ for $k = 0, \ldots, N$ (use Equation 8.9)
(18) use $F(\sigma_k^{heuristic})$ for $k = 0, \ldots, N$ to compute next sequence $\sigma^{heuristic}$
(19) **if** $\sigma^{heuristic}$ did not change (is a local optima)
(20) stop $\leftarrow$ true
(21) UPDATEAQDF(heuristic, bias, objective($\sigma^{heuristic}$))
(22) **if** $\sigma^{heuristic}$ is better than bestsofar
(23) bestsofar $\leftarrow \sigma^{heuristic}$
(24) **if** $\sigma^{heuristic}$ is better than best[heuristic]
(25) best[heuristic] $\leftarrow \sigma^{heuristic}$
(26) **if** N[heuristic] $\bmod \beta = 0$
(27) $\sigma^{heuristic} \leftarrow$ best[heuristic]
(28) **for** 1 **to** α
(29) compute random set of independent swaps and move to neighboring $\sigma^{heuristic}$
(30) **return** bestsofar

8.7 Using QD-BEACON to Enhance Iterated Dynasearch

The Iterated Dynasearch algorithm is the current best performing algorithm for weighted tardiness sequencing problems. Even the VBSS/QD-BEACON Enhanced Multistart Dynasearch, though better than plain Multistart Dynasearch, does not compare to Iterated Dynasearch. The question we consider here is whether Iterated Dynasearch can be improved further by using QD-BEACON in some way.

Recall that the Iterated Dynasearch algorithm seeds its initial solution by the direct deterministic application of a dispatch policy. The dispatch policy that we have chosen to use is a best of four heuristics where the four heuristics are EDD, WSPT, R&M, and COVERT.[1] Now recall that given the initial start, iterated dynasearch makes local improvements via sets of independent swaps until a local optima is reached. When a local optima is reached, it "kicks" the current search state with a series of random sets of independent swaps. Every so many iterations, it "kicks" the current best found solution instead of the current local optima.

We modify the Iterated Dynasearch algorithm to interleave the execution of 4 simultaneous iterated dynasearches. Each of these 4 searches begins at an initial starting configuration given by one of the 4 dispatch policies. The QD-BEACON framework is used to model the distributions of local optima encountered by each of these 4 independent searches. QD-BEACON is further used to control the number of iterations given to each of the independent searches based on the probability of finding a better solution than the best found so far implied by the AQDFs of each independent search. The hypothesis is that if the 4 different starting configurations are spread far enough apart in the objective landscape then we may be able to conduct a local search of as many as four different regions of the search space while expending more effort on the region that looks the most promising. The QD-BEACON enhanced version of iterated dynasearch is shown in Algorithm 8.4.

The results can be found in Table 8.5 (40 job problems), Table 8.6 (50 job problems), and Table 8.7 (100 job problems). We compare the following:

- I-DYNA[N]: The original Iterated Dynasearch, where the starting configuration of the search is the best solution given by the deterministic application of each of the four dispatch policies. N is the number of kicks.

- NORM-I-DYNA[N]: Iterated Dynasearch enhanced using QD-BEACON. Four independent iterated dynasearches are interleaved, each beginning from a dispatch policy

[1] Congram *et al.* originally used R&M.

Table 8.5: 40 Job Set: QD-BEACON Enhanced Iterated Dynasearch vs the original Iterated Dynasearch. For each number of iterations (kicks), bold indicates the best in terms of number of optimal solutions; italics indicates the best in terms of percentage deviation.

Algorithm	NO	ARPD	MRPD
I-DYNA[100]	122.7 (125)	0.05 (0.00)	5.01 (0.00)
KDE-I-DYNA[100]	122.3 (124)	0.03 (0.01)	3.09 (1.29)
GEV-I-DYNA[100]	121.7 (125)	0.045 (0.000)	4.53 (0.00)
NORM-I-DYNA[100]	121.3 (123)	0.03 (0.01)	3.09 (1.29)
KDE-I-DYNA[200]	125 (125)	0.0 (0.0)	0.0 (0.0)
I-DYNA[200]	124 (125)	0.026 (0.000)	3.17 (0.00)
NORM-I-DYNA[200]	123.7 (125)	0.025 (0.000)	2.66 (0.00)
GEV-I-DYNA[200]	123.7 (125)	0.025 (0.000)	2.72 (0.00)
KDE-I-DYNA[400]	125 (125)	0.0 (0.0)	0.0 (0.0)
I-DYNA[400]	124.7 (125)	0.0008 (0.0000)	0.11 (0.00)
GEV-I-DYNA[400]	124.3 (125)	0.007 (0.000)	0.86 (0.00)
NORM-I-DYNA[400]	124 (125)	0.025 (0.000)	2.66 (0.00)
KDE-I-DYNA[800]	125 (125)	0.0 (0.0)	0.0 (0.0)
I-DYNA[800]	125 (125)	0.0 (0.0)	0.0 (0.0)
GEV-I-DYNA[800]	124.7 (125)	0.003 (0.000)	0.43 (0.00)
NORM-I-DYNA[800]	124.7 (125)	0.003 (0.000)	0.43 (0.00)
NORM-I-DYNA[1600]	125 (125)	0.0 (0.0)	0.0 (0.0)
KDE-I-DYNA[1600]	125 (125)	0.0 (0.0)	0.0 (0.0)
GEV-I-DYNA[1600]	125 (125)	0.0 (0.0)	0.0 (0.0)
I-DYNA[1600]	125 (125)	0.0 (0.0)	0.0 (0.0)

solution given by one of the four heuristics. QD-BEACON using Normal distribution estimates is used to control the proportion of computation time allocated each of the searches as in Algorithm 8.4. N is the total number of kicks that must be allocated to the four searches.

- KDE-I-DYNA[N]: Iterated Dynasearch enhanced using QD-BEACON. Four independent iterated dynasearches are interleaved, each beginning from a dispatch policy solution given by one of the four heuristics. QD-BEACON using kernel density estimates is used to control the proportion of computation time allocated each of the searches as in Algorithm 8.4. N is the total number of kicks that must be allocated to the four searches.

- GEV-I-DYNA[N]: Iterated Dynasearch enhanced using QD-BEACON. Four independent iterated dynasearches are interleaved, each beginning from a dispatch policy

Table 8.6: 50 Job Set: QD-BEACON Enhanced Iterated Dynasearch vs the original Iterated Dynasearch. For each number of iterations (kicks), bold indicates the best in terms of number of optimal solutions; italics indicates the best in terms of percentage deviation.

Algorithm	NO	ARPD	MRPD
KDE-I-DYNA[100]	111.7 (120)	0.038 (0.007)	2.05 (0.29)
I-DYNA[100]	111 (116)	0.06 (0.03)	4.06 (1.63)
NORM-I-DYNA[100]	110.7 (120)	0.04 (0.02)	2.80 (2.27)
GEV-I-DYNA[100]	109.7 (120)	0.034 (0.007)	1.47 (0.49)
NORM-I-DYNA[200]	119 (121)	0.019 (0.006)	1.51 (0.41)
KDE-I-DYNA[200]	118.7 (121)	0.019 (0.005)	1.43 (0.29)
GEV-I-DYNA[200]	116.7 (122)	0.017 (0.004)	1.04 (0.36)
I-DYNA[200]	115.3 (119)	0.03 (0.01)	2.32 (0.62)
NORM-I-DYNA[400]	121 (123)	0.009 (0.003)	0.67 (0.17)
GEV-I-DYNA[400]	121 (123)	0.012 (0.002)	1.00 (0.23)
KDE-I-DYNA[400]	120.7 (122)	0.014 (0.005)	1.16 (0.29)
I-DYNA[400]	119.3 (121)	0.010 (0.004)	0.74 (0.23)
GEV-I-DYNA[800]	122.7 (124)	0.004 (0.002)	0.31 (0.23)
NORM-I-DYNA[800]	122 (124)	0.004 (0.001)	0.25 (0.16)
KDE-I-DYNA[800]	121.7 (124)	0.009 (0.002)	0.78 (0.29)
I-DYNA[800]	121 (123)	0.005 (0.003)	0.27 (0.23)
NORM-I-DYNA[1600]	123 (124)	0.003 (0.001)	0.23 (0.16)
GEV-I-DYNA[1600]	123 (124)	0.003 (0.002)	0.29 (0.23)
KDE-I-DYNA[1600]	122.7 (124)	0.007 (0.002)	0.70 (0.23)
I-DYNA[1600]	122 (123)	0.004 (0.003)	0.21 (0.17)

solution given by one of the four heuristics. QD-BEACON using GEV distribution estimates is used to control the proportion of computation time allocated each of the searches as in Algorithm 8.4. N is the total number of kicks that must be allocated to the four searches.

The results of a comparison of these algorithms for the 40 job problems, 50 job problems, and 100 job problems can be found in Table 8.1, Table 8.2, and Table 8.3, respectively. We can make the following general observations about these results:

- For the 40 job problem set (Table 8.1), with a small number of total kicks, $N = 100$, the original iterated dynasearch, I-DYNA[N] is able to find more of the optimal solutions on average as compared to the other algorithms; while using QD-BEACON to enhance the search does result in improved relative percentage deviation from the optimal solution values. However, more importantly, with as few as 200 total kicks, using QD-BEACON with Kernel Density Estimation to enhance the search (KDE-

Table 8.7: 100 Job Set: QD-BEACON Enhanced Iterated Dynasearch vs the original Iterated Dynasearch. For each number of iterations (kicks), bold indicates the best in terms of number of optimal solutions; italics indicates the best in terms of percentage deviation.

Algorithm	NO	ARPD	MRPD
I-DYNA[100]	93.3 (108)	0.069 (0.004)	5.09 (0.16)
KDE-I-DYNA[100]	85 (100)	0.06 (0.01)	3.54 (0.64)
NORM-I-DYNA[100]	84.7 (101)	0.09 (0.02)	4.83 (1.79)
GEV-I-DYNA[100]	81.3 (95)	0.12 (0.03)	6.72 (1.89)
I-DYNA[200]	102.3 (114)	0.026 (0.002)	2.03 (0.16)
KDE-I-DYNA[200]	101.7 (116)	0.025 (0.007)	1.44 (0.64)
NORM-I-DYNA[200]	99 (111)	0.048 (0.017)	3.29 (1.79)
GEV-I-DYNA[200]	95 (106)	0.077 (0.005)	6.62 (0.22)
KDE-I-DYNA[400]	112 (119)	0.020 (0.006)	1.44 (0.64)
NORM-I-DYNA[400]	111 (120)	0.03 (0.01)	3.15 (1.79)
I-DYNA[400]	110.7 (120)	0.0064 (0.0009)	0.46 (0.08)
GEV-I-DYNA[400]	108.7 (118)	0.014 (0.001)	0.71 (0.09)
NORM-I-DYNA[800]	118.7 (124)	0.013 (0.005)	1.40 (0.64)
KDE-I-DYNA[800]	118.3 (122)	0.013 (0.005)	1.40 (0.64)
GEV-I-DYNA[800]	117.7 (124)	0.0058 (0.0003)	0.61 (0.04)
I-DYNA[800]	117.3 (123)	0.0051 (0.0004)	0.45 (0.04)
KDE-I-DYNA[1600]	122.7 (125)	0.005 (0.000)	0.61 (0.00)
I-DYNA[1600]	121.7 (125)	0.0022 (0.0000)	0.24 (0.00)
GEV-I-DYNA[1600]	121.7 (124)	0.0022 (0.0001)	0.25 (0.01)
NORM-I-DYNA[1600]	121.3 (124)	0.012 (0.005)	1.40 (0.64)

I-DYNA[N]) can consistently solve to optimality all 125 problem instances. The original iterated dynasearch requires 800 kicks (I-DYNA[800]) before it begins to consistently solve all 125 problem instances to optimality.

- For the 50 job problem set (Table 8.2), some version of the QD-BEACON enhanced algorithm is the best performer for each number of kicks considered and for both the number of optimal solutions found and for relative percentage deviation. An even stronger observation is that for any number of kicks greater than or equal to 200, the original iterated dynasearch is the worst performer of those algorithms considered in terms of number of solutions solved to optimality (i.e., the QD-BEACON enhanced algorithm is better than I-DYNA independent of which of the three estimation methods is used for the AQDFs).

- For the 100 job problem set (Table 8.3), I-DYNA does best for 200 or less kicks in

Table 8.8: 40 Job Set: Comparison of QD-BEACON Enhanced Iterated Dynasearch with various Local Search Algorithms.

Algorithm	NO	ARPD	MRPD	TIME
KDE[200]	124	0.01	0.86	0.70
KDE-I-DYNA[200]	125	0.00	0.00	0.20
D[P]	120	0.04	4.21	0.55
D[B]	121	0.00	0.05	0.76
DN[P]	115	0.01	0.41	0.77
DN[B]	123	0.00	0.05	0.81
SA[P]	121	0.01	0.81	0.84
SA[B]	116	0.00	0.07	0.85
TA[P]	121	0.06	6.70	0.86
TA[B]	119	0.00	0.05	0.83
TS[P]	123	0.00	0.17	0.84
TS[B]	122	0.00	0.21	0.82
GA[B]	119	0.00	0.08	0.45

terms of number of best known solutions found; while the QD-BEACON enhanced algorithm using kernel density estimation is best in terms of relative percentage deviation. However, as more kicks are given the algorithms, the QD-BEACON enhanced approach does best in number of best known solutions found (for 400 kicks or more).

Overall, the QD-BEACON enhanced approach has some benefit. For the 40 job problems, it allows us to consistently solve all problem instances to optimality with fewer iterations (kicks) and thus less CPU time. For the 50 job problems, the QD-BEACON enhanced approach is always best, independent of the number of iterations given the algorithms. Finally, for the most difficult problem set (the 100 job problems), the QD-BEACON enhanced approach performs best for longer searches.

8.8 Comparison with Other Local Search Algorithms

In this Section, we compare the QD-BEACON enhanced iterated dynasearch to the local search algorithms listed earlier in Section 8.3.3. These local search algorithms were originally tested by Crauwels *et al.* on an HP 9000/G50 which ran at 96MHz [59]. We have used a very crude conversion of the CPU times reported by Crauwels *et al.* to the equivalent CPU times on our Sun Ultra 10/300MHz. That is, we have multiplied the times reported by Crauwels *et al.* by $\frac{96}{300}$.

Table 8.9: 50 Job Set: Comparison of QD-BEACON Enhanced Iterated Dynasearch with various Local Search Algorithms.

Algorithm	NO	ARPD	MRPD	TIME
KDE[200]	117	0.05	2.70	1.27
GEV-I-DYNA[800]	123	0.00	0.31	1.26
D[P]	119	0.00	0.26	1.41
D[B]	112	0.01	0.18	1.76
DN[P]	102	0.02	0.84	1.79
DN[B]	114	0.01	0.19	1.76
SA[P]	115	0.09	11.20	1.96
SA[B]	101	0.01	0.45	1.92
TA[P]	111	0.01	0.28	2.05
TA[B]	108	0.01	0.18	1.82
TS[P]	118	0.00	0.16	1.89
TS[B]	115	0.01	0.36	1.72
GA[B]	113	0.01	0.19	1.00

Table 8.10: 100 Job Set: Comparison of QD-BEACON Enhanced Iterated Dynasearch with various Local Search Algorithms.

Algorithm	NO	ARPD	MRPD	TIME
KDE[400]	94	0.12	10.07	8.70
KDE-I-DYNA[1600]	123	0.01	0.61	5.61
D[P]	86	0.06	4.78	12.54
D[B]	57	0.06	1.85	12.83
DN[P]	72	0.07	4.78	12.83
DN[B]	69	0.04	1.85	13.02
SA[P]	59	0.12	4.78	12.96
SA[B]	59	0.16	1.05	13.06
TA[P]	70	0.10	4.78	12.83
TA[B]	64	0.06	1.86	12.67
TS[P]	103	0.04	4.39	12.03
TS[B]	66	0.04	0.34	14.43
GA[B]	77	0.03	0.76	11.94

The results of this comparison with various local search algorithms is seen in Table 8.8, Table 8.9, and Table 8.10 for the 40 job, 50 job, and 100 job problem sets, respectively. In each table, we list the local search results as reported by Crauwels *et al.* with the CPU times adjusted as we have stated above. Also listed in each table, is the best variation of the multistart dynasearch algorithm for a comparable CPU time as well as the best variation of the iterated dynasearch algorithm for a comparable CPU time. We can make the following

observations:

- For the 40 job problems (Table 8.8), KDE-I-DYNA[200] is the clear winner in the comparison. As mentioned earlier, for as few as 200 kicks (iterations) of the QD-BEACON with KDE Enhanced Iterated Dynasearch algorithm, we can optimally solve all 125 of the 40 job instances. This can be done in a fraction of the time the other algorithms require. The next best algorithm is the QD-BEACON/VBSS Enhanced Multistart Dynasearch with 200 restarts (KDE[200]), but this algorithm requires significantly more time than KDE-I-DYNA[200].

- For the 50 job problems (Table 8.9), GEV-I-DYNA[800] is the clear winner. The QD-BEACON (with GEV) Enhanced Iterated Dynasearch algorithm with 800 iterations (kicks) optimally solves more of the 50 job problems with less relative percentage deviation from the optimal and in less CPU time than any of the other local search algorithms. In the same amount of CPU time, there is only enough time for 200 restarts of the multistart algorithm, while the 800 kicks of the iterated algorithm lead to better results.

- For the 100 job problems (Table 8.10), in less than half the amount of time, KDE-I-DYNA[1600] is able to find significantly more of the best known solutions than any of the other local search algorithms. In fact, the QD-BEACON with KDE Enhanced Iterated Dynasearch with 1600 iterations (kicks) requires less time than 400 restarts of the QD-BEACON with KDE Enhanced Multistart Dynasearch; and in that less time is able to find approximately 30% more of the best known solutions.

The overall result is that the QD-BEACON Enhanced Iterated Dynasearch Algorithm is clearly the dominant algorithm for this problem as compared to the other local search algorithms available as benchmarks.

8.9 Summary

In this Chapter, we considered the problem of weighted tardiness sequencing. This is an NP-Hard optimization problem for which complete search algorithms are currently unable to guarantee finding optimal solutions to problems larger than 40-50 jobs.

We presented enhanced versions of the two current best local search algorithms for the problem:

- First, we used the QD-BEACON framework to combine the use of four different dispatch policies within the VBSS framework to seed the starting solution configurations of a multistart version of the dynasearch algorithm. The result was that, contrary to Congram *et al.*'s untested hypothesis, we were able to obtain better results by using heuristic guidance in seeding the restarts. We were able to then further improve upon the performance by combining multiple heuristics within the QD-BEACON framework.

- Second, we used the QD-BEACON framework to enhance the iterated dynasearch algorithm. Until now, Iterated Dynasearch was the best performing algorithm for this problem. Our enhanced version, uses QD-BEACON to control the amount of search allocated to each of four independent iterated dynasearches whose execution is interleaved. Each of these four iterated dynasearches begins at a different initial starting solution provided by one of four dispatch policies. QD-BEACON is used to model the distribution of local optima encountered by each and to control the number of iterations (kicks) given to each of the four. This allows the search to explore as many as four different regions of the search space, concentrating more effort on the more promising regions as the search continues. The original Iterated Dynasearch algorithm of Congram *et al.* began its search at a single starting solution state. The QD-BEACON Enhanced version exhibits improved performance and thus shows us that there is a benefit to the approach. Additionally, this QD-BEACON Enhanced Iterated Dynasearch is the new best algorithm for the problem.

Chapter 9

Application: Resource Constrained Project Scheduling with Time Windows

9.1 Overview

In this Chapter, we consider the resource constrained project scheduling problem with time windows (RCPSP/max). RCPSP/max is the RCPSP with generalized precedence relations between start times of activities. It is a difficult makespan minimization problem well studied by the Operations Research community. Finding feasible solutions to instances of the RCPSP/max is NP-Hard, making the optimization problem very difficult. The RCPSP/max problem adds an interesting dimension to the analysis of the QD-BEACON/VBSS framework in that solving this problem requires handling a difficult CSP problem within the context of the optimization process.

The RCPSP/max problem is applicable within the COMIREM project of the CMU Robotics Institute [186, 187]. COMIREM (Continuous, Mixed-Initiative Resource Management) is a system for collaborative, incremental development of plans and schedules in dynamic, resource-constrained domains. Through lightweight plug-in applications, the objective of COMIREM is to deliver a wide-range of web-based planning and scheduling services. The VBSS/QD-BEACON framework, with its simple, fast, effective stochastic search abilities, will become a valuable addition to the COMIREM system provided it is capable of efficiently solving the RCPSP/max problem.

The RCPSP/max problem is also applicable within the CMU Robotics Institute project FIRE. The FIRE (Federation of Intelligent Robotic Explorers) project team is developing a market-based distributed multi-robot coordination architecture [89, 90]. The architecture is

comprised of three layers: the planning, executive, and behavior layers. The planning layer is further subdivided into various components. Two primary components of this planning layer include: 1) the RoboTrader which is responsible for activities such as bidding on tasks and conducting auctions; and 2) the RoboScheduler which is responsible for activities such as evaluating the cost associated with scheduling a potential new task and the savings associated with removing some task from its current schedule. It is within the RoboScheduler that the RCPSP/max problem is relevant. In the current instantiation of the FIRE system, the Scheduler is faced with the need to frequently solve numerous scheduling problems that are essentially instances of the traveling salesperson problem (TSP). One of the future goals of the project is to incorporate more complex scheduling constraints such as temporal constraints between the start times of pairs of tasks, coordinated execution of tasks involving multiple robots, and the ability of a robot to execute multiple tasks in parallel provided it has the resources to do so. All of these constraints can be specified within the formalization of the RCPSP/max problem. And thus, the algorithms that are now developed in this Chapter can be plugged into the Scheduling component of the FIRE system when the need for them arises.

The remainder of this Chapter is organized as follows. In Section 9.2 we formalize the RCPSP/max problem. Section 9.3 overviews the state-of-the-art in solution methods available for the RCPSP/max problem. Section 9.4 describes the set of benchmark instances used in the experiments studied later in this Chapter. Section 9.5 defines our performance criteria. Section 9.6 details the QD-BEACON/VBSS Iterative Priority-Rule Method. Section 9.7 presents a comparison of a number of different algorithms with our approach – combining multiple search heuristics using QD-BEACON. We conclude the Chapter with a summary in Section 9.8.

9.2 Problem Formalization

The RCPSP/max problem can be defined formally as follows. Define $P =< A, \Delta, R >$ as an instance of RCPSP/max. Let A be the set of activities $A = \{a_0, a_1, a_2, \ldots, a_n, a_{n+1}\}$. Activity a_0 is a dummy activity representing the start of the project and a_{n+1} is similarly the project end. Each activity a_j has a fixed duration p_j, a start-time S_j, and a completion-time C_j which satisfy the constraint $S_j + p_j = C_j$. Let Δ be a set of temporal constraints between activity pairs $< a_i, a_j >$ of the form $S_j - S_i \in [T_{i,j}^{\min}, T_{i,j}^{\max}]$. The Δ are generalized precedence relations between activities. The $T_{i,j}^{\min}$ and $T_{i,j}^{\max}$ are minimum and maximum

time-lags between the start times of pairs of activities. Let R be the set of renewable resources $R = \{r_1, r_2, \ldots r_m\}$. Each resource r_k has an integer capacity $c_k \geq 1$. Execution of an activity a_j requires one or more resources. For each resource r_k, the activity a_j requires an integer capacity $rc_{j,k}$ for the duration of its execution. An assignment of start-times to the activities in A is time-feasible if all temporal constraints are satisfied and is resource-feasible if all resource constraints are satisfied. A schedule is feasible if both sets of constraints are satisfied. The problem is then to find a feasible schedule with minimum makespan M where $M(S) = \max\{C_i\}$. That is we wish to find a set of assignments to S such that $S_{\text{sol}} = \arg\min_S M(S)$. The maximum time-lag constraints are what makes this problem especially difficult. Particularly, due to the maximum time-lag constraints, finding feasible solutions alone to this problem is NP-Hard.

9.3 State-of-the-Art Solution Methods

Neumann, Schwindt, and Zimmermann give a very thorough and exhaustive overview of the RCPSP/max problem and of the solution techniques that have been used to solve it [150]. For more detail on any of the algorithms mentioned in this Section see either the references to the individual papers describing them or see Neumann *et al.*'s survey. The algorithms presented in this Section are organized according to branch-and-bound approaches (Section 9.3.1), priority-rule methods (Section 9.3.2), local search (Section 9.3.3), and iterative sampling (Section 9.3.4).

9.3.1 Branch-and-Bound

There are a large number of branch-and-bound approaches for the RCPSP/max problem. We do not give the details of any of these here since this would be an unnecessary side-track. We simply point out that there has been much success in applying branch-and-bound algorithms to the problem. Though for many problem instances it is too costly to execute a branch-and-bound long enough to prove optimality, good solutions are often obtained in a reasonable amount of computation time through truncation (i.e., not allowing the search to run to completion). Following is a list of the branch-and-bound approaches (abbreviations given will later be used in the results section) for which results are available for the benchmark set used in this Chapter:

- B&B$_{DRH98}$: De Reyck and Herroelen's branch-and-bound algorithm [63].

- B&B$_{F98}$: Fest *et al.*'s branch-and-bound algorithm [77].
- B&B$_{S98}$: Schwindt's branch-and-bound algorithm [177, 81].
- B&B$_{DPP98}$: Dorndorf *et al.*'s branch-and-bound algorithm [70].

9.3.2 Priority-Rule Methods

In this Section, I overview priority-rule methods for the problem. It should be noted that a priority rule method, as used in this Chapter, is not the same thing as a dispatch policy. It actually refers to a backtracking CSP search that uses one or more priority-rules (heuristics) to choose an activity to schedule next. By "schedule next", I do not mean in a sequencing sort of way, but rather I mean fixing its start time variable in place. Recall, that the RCPSP/max is more than just an optimization problem. It is also a constraint satisfaction problem. And thus, when a start time becomes fixed, there is also some amount of constraint propagation that takes place, further constraining the domains of the start time variables. The priority-rule method described in this Section is important because it will be the method which is later randomized using VBSS and QD-BEACON in Section 9.6.

The specific priority-rule method that I will now describe is referred to as the "direct method" with a "serial schedule generation scheme" [81, 150]. There also exist a couple decomposition methods as well as a parallel schedule generation scheme, but Franck *et al.* found the direct method with serial generation scheme to perform better in general.

The priority-rule method makes use of a graph representation of the temporal constraints. This graph representation is called a project network. Nodes represent activities. Directed edges with positive weights represent minimal time-lag constraints and directed edges with negative weights represent maximal time-lags. The priority-rule method begins with a preprocessing phase which consists in propagating constraints and a few other tricks. First, all strong components in the project network are found; and then later in the schedule generation whenever an activity is scheduled, all other activities from the same strong component (also referred to as a cycle structure) are then scheduled next one at a time. Next, for each pair of activities with a combined resource requirement greater than the resource capacity, a temporal constraint is added to the problem to break up the pair (two-element forbidden set). Finally, constraint propagation is used to propagate temporal constraints to adjust the domains on the start times of the activities.

After the preprocessing phase is complete, the priority-rule method begins the serial generation scheme. The serial generation scheme is a backtracking CSP search proce-

Algorithm 9.1: Priority-Rule Method for RCPSP/max: The Serial Schedule Generation Scheme
Input: The initial earliest and latest start times of the activities, ES_i, LS_i, the weights of the project network $\delta_{i,j}$, the immediate predecessors of the activities Pred_i, a maximum number of unscheduling steps (backtracks) $u_{\max}$, a heuristic h.
Output: A set of assignments to the start times S_i.
SERIAL-SCHEDULE-GENERATION(ES_i, LS_i, $\delta_{i,j}$, Pred_i, $u_{\max}$, h)

(1) $S_0 \leftarrow 0$
(2) Add a_0 to scheduled set
(3) $u \leftarrow 0$
(4) initialize all resource profiles
(5) **while** not all activities scheduled
(6) **if** some but not all activities from some cycle structure CS scheduled
(7) let the eligible set be all unscheduled activities in CS such that all of its predecessors have been scheduled
(8) **else**
(9) let the eligible set be all activities such that all predecessors have been scheduled and such that all of the predecessors of the activities in its cycle structure have either been scheduled already or are in that same cycle structure
(10) $j^* \leftarrow$ the eligible activity preferred by heuristic h
(11) $t^* \leftarrow$ the earliest resource feasible start time $\geq ES_{j^*}$
(12) **if** $t^* > LS_{j^*}$
(13) $u \leftarrow u + 1$
(14) **if** $u > u_{\max}$
(15) maximum number of unscheduling steps reached
(16) **return** the empty solution
(17) UNSCHEDULE(j^*, $t^* - LS_{j^*}$)
(18) **else**
(19) Schedule j^* at time t^*
(20) $S_{j^*} = t^*$
(21) Add j^* to scheduled set
(22) update all resource profiles
(23) **foreach** not yet scheduled activity i
(24) $ES_i = \max(ES_i, S_{j^*} + \delta_{j^*,i})$
(25) $LS_i = \min(LS_i, S_{j^*} - \delta_{i,j^*})$

Algorithm 9.2: Priority-Rule Method for RCPSP/max: The Unscheduling Step
Input: An activity j^*, a value Δ.
Output:
UNSCHEDULE(j^*, Δ)
(1) $U \leftarrow$ the set of all scheduled activities i such that $S_i - \delta_{j^*,i} = LS_{j^*}$
(2) **if** a_0 in U
(3) no feasible schedule can be found
(4) **return** terminate
(5) **foreach** i in U
(6) $ES_i = S_i + \Delta$
(7) remove i from the scheduled set
(8) **foreach** scheduled activity i such that $S_i > \min_{h \in U} S_h$
(9) remove i from the scheduled set
(10) update resource profiles
(11) **foreach** not yet scheduled activity i
(12) $ES_i = \max(\delta_{0,i}, \max_{h \in U}(ES_h + \delta_{h,i}))$
(13) $LS_i = -\delta_{i,0}$
(14) **foreach** scheduled activity k
(15) $ES_i = \max(ES_i, S_k + \delta_{k,i})$
(16) $LS_i = \min(LS_i, S_k - \delta_{i,k})$

dure. Each decision point in the search chooses an activity to schedule next from a set of "eligible" activities. An activity is eligible if all of its temporal predecessors have been scheduled. This decision is made using a priority-rule (possible heuristics for this decision are discussed below). Once selected, the start time of an activity is scheduled at its earliest time and resource feasible time. At this point, constraint propagation takes place to adjust the domains of the start times of the not yet scheduled activities. If there is no time and resource feasible time to start the selected activity, then the serial generation scheme performs what it calls an unscheduling step. The unscheduling step essentially is a backtrack. It finds the scheduled activity or activities which begin one or more maximal time lag constraints involving the activity which we were unable to schedule. It unschedules these and constrains the start times to occur late enough to allow for the scheduling of this unschedulable activity. It then also unschedules all activities that had been scheduled later than any of these other unscheduled activities. Appropriate constraint propagation also takes place at this point. The serial generation scheme is shown in Algorithm 9.1 with the unscheduling step (backtracking step) shown in Algorithm 9.2. Neumann *et al.* recommend setting the maximum number of unscheduling steps equal to $10\sqrt{n}$.

The serial schedule generation scheme requires a priority-rule or activity selection

heuristic. There are a wide variety of such heuristics available in the literature. Neumann *et al.* recommend five in particular. These five heuristics are those that we later use along with VBSS and QD-BEACON:

- LST: smallest "latest start time" first:

$$\text{LST}_i = \frac{1}{1 + LS_i}. \tag{9.1}$$

- MST: "minimum slack time" first:

$$\text{MST}_i = \frac{1}{1 + LS_i - ES_i}. \tag{9.2}$$

- MTS: "most total successors" first:

$$\text{MTS}_i = |\text{Successors}_i| \tag{9.3}$$

  where Successors_i is the set of not necessarily immediate successors of a_i in the project network.

- LPF: "longest path following" first

$$\text{LPF}_i = \text{lpath}(i, n+1) \tag{9.4}$$

  where $\text{lpath}(i, n+1)$ is the length of the longest path from a_i to a_{n+1}.

- RSM: "resource scheduling method"

$$\text{RSM}_i = \frac{1}{1 + \max\left(0, \max_{g \in \text{eligible set}, g \neq i}\left(ES_i + p_i - LS_g\right)\right)}. \tag{9.5}$$

Note that I have rephrased a few of these heuristics from Neumann *et al.*'s definitions so that for each, the eligible activity with the highest heuristic value is chosen. Other heuristics are available in the literature (see for example, [151, 79]). There are others available for the RCPSP without generalized precedence constraints [122, 121] that do not obviously apply to the RCPSP/max problem.

Later in the discussion of results, we will refer to the following multiple run priority methods:

- PR_{FNS5}: Executing the direct method with serial generation scheme 5 times, once with each of the heuristics described above, and taking the best solution of the 5. Frank *et al.* originally suggested using the best of these 5 heuristics solutions [81, 150]. Results shown later are of our implementation.

- PR_{FN10}: Similarly, this is a best of 10 heuristics. The results shown later are as reported by Dorndorf *et al.* [70] and Cesta *et al.* [32] of Franck and Neumann's best of 10 heuristics method [80].[1]

9.3.3 Local Search

Franck *et al.* describe a number of local search algorithms for the problem [82, 81, 150]. We leave the reader to the references for the details of the algorithms. Here we simply state that they begin by constructing one or more schedules using the direct priority-rules method and improve those schedules via local search. The algorithms later used in the comparison include:

- GA: a genetic algorithm.

- TS: tabu search.

- SA: simulated annealing

9.3.4 Iterative Sampling Earliest Solutions

Cesta *et al.* present an algorithm for the RCPSP/max problem that they call *Iterative Sampling Earliest Solutions (ISES)* [30, 31, 32]. ISES begins by finding a time feasible solution with a maximum horizon (initially very large) on the project's makespan, assuming one exists. The resulting time-feasible solution, for any interesting problem instance, is generally not resource-feasible. ISES proceeds by iteratively "leveling" resource-constraint conflicts. That is, it first detects sets of activities that temporally overlap and whose total resource requirement exceeds the resource capacity. Given the set of resource-constraint conflicts, it chooses one of the conflicts using heuristic-equivalency (i.e., chooses randomly from among all those resource-conflicts within an "acceptance band" in heuristic value from the

[1] Franck and Neumann's technical report describing this best of 10 strategy is no longer available according to both the library at their institution as well as the secretary of their lab. We have been unable to find out what the 10 heuristics are that produce these results.

heuristic preferred choice). It then levels the chosen conflict by posting a precedence constraint between two of the activities in the conflicted set. It continues until a time-feasible and resource-feasible solution is found or until some resource-conflict cannot be leveled. This is then iterated some fixed number of times within a stochastic sampling framework. Then, given the best found solution of the stochastic sampling process, the entire algorithm is repeated iteratively for smaller and smaller horizons, specifically for a horizon on the makespan of a time-feasible solution equal to the makespan of the best feasible solution found so far. The rest of the low-level details of the ISES algorithm can be found in the references. Cesta *et al.* show ISES to be better than the previous best heuristic algorithm for the RCPSP/max problem (namely PR_{FN10}).

9.4 The Benchmark Problem Set

The set of benchmark problem instances that we use in the experimental study in this Chapter is that of Schwindt available at http://www.wior.uni-karlsruhe.de/LS_Neumann/Forschung/ProGenMax/rcpspmax.html. It is a problem set that has been generated by the problem generator known as ProGen/max [178, 179, 176, 175].[2] There are many approaches to the RCPSP/max problem that have used this problem set, and thus a large number of approaches to which to compare our QD-BEACON/VBSS approach. Furthermore, lower bounds, current best known solutions, and in many cases optimal solutions are available for the problem instances of this benchmark set.

There are 1080 problem instances in this problem set. Of the 1080 problem instances, 1059 have feasible solutions and the other 21 are provably infeasible. Each instance has 100 activities and 5 renewable resources. The difficulty of the problem instances is controlled by a number of parameters of the problem generator. The two primary parameters that govern problem difficulty are the order strength of the project network and the resource strength of the network. The order strength is a measure of how much of the project network is strictly ordered. An order strength of 0 means that the temporal constraints allow all activities to be executed simultaneously (though the resource constraints do not necessarily allow this). An order strength equal to 1 means that for any pair of activities there is a minimal time lag from the start of one of the activities to the start of the other, resulting in a strict order of the activities in the project. Projects with lower order strengths are

[2]The ProGen/max problem generator is largely based on a problem generator for the version of the problem without generalized precedence constraints – the ProGen [124, 123] problem generator.

generally more difficult. The resource strength is a measure of the scarcity of resources. A resource strength of 0 means that the resources only have the minimum capacity necessary to process a single activity at a time. A resource strength of 1 means that the schedule that results from starting each activity at its earliest temporally feasible start time is also resource feasible and thus optimal. Projects with smaller values of the resource strength parameter are generally more difficult, though in the case of resource strength equal to (or close to) 0, many resource conflicts can be dealt with fairly easily. See the complete detailed description of the problem generator for more detail of these and other generator parameters [178].

9.5 Performance Criteria

In the experiments that follow, we use the following performance criteria which have been used by several others to compare the performance of algorithms for the RCPSP/max problem:

- Δ_{LB}: the average relative deviation from the known lower bound, averaged across all problem instances for which a feasible solution was found. Note that this is based on the number of problem instances for which the given algorithm was able to find a feasible solution and thus might be based on a different number of problem instances for each algorithm compared. This criteria, as defined, is exactly as used by all of the other approaches to the problem available in the literature.
- NO: the number of optimal solutions found. Currently, there are known optimal solutions for 789 of the 1080 problem instances.
- NF: the number of feasible solutions found. Of the 1080 problem instances, 1059 possess at least one feasible solution. The other 21 can be proven infeasible (e.g., by the preprocessing step of the priority-rule method).
- TIME: CPU time in seconds.

For all stochastic algorithms, values shown are averages across 10 runs. Values in parentheses are best of the 10 runs. In the results, as an added comparison point, we list the above criteria for the current best known solutions as BEST. Note that BEST is the best known prior to the algorithms of this thesis. We further improve upon the best known solutions to some of the problem instances, but this is not considered in BEST.

9.6 QD-BEACON/VBSS Iterative Priority-Rule Method

In this Section we detail the QD-BEACON/VBSS Iterative Priority-Rule Method. We begin by modifying the serial generation scheme to use VBSS to randomize the heuristic used to choose which activity to schedule from among the eligible activities. This modified serial generation scheme can be found in Algorithm 9.3. The only change from Algorithm 9.1 is the use of VBSS to randomize the choice from among the eligible activities. The unscheduling step is as described earlier.

We detail the QD-BEACON/VBSS Iterative Priority-Rule Method in Algorithm 9.4. This algorithm iteratively calls the VBSS randomized priority-rule serial generation scheme algorithm with heuristic / bias function pairs chosen using the QD-BEACON framework and updates the AQDF of the appropriate heuristic / bias function pair after each iteration.

In the results that follow, we will refer to using the stochastic sampling framework VBSS within the priority-rule method for N iterations by:

- LST[N]: sampling using the LST heuristic (polynomial bias function degree 10).
- MST[N]: sampling using the MST heuristic (polynomial bias function degree 10).
- MTS[N]: sampling using the MTS heuristic (polynomial bias function degree 2).
- LPF[N]: sampling using the LPF heuristic (polynomial bias function degree 3).
- RSM[N]: sampling using the RSM heuristic (polynomial bias function degree 4).
- NAIVE[N]: sampling an equal number of times with each of the five heuristics (N iterations total).

No in depth tuning process was performed to determine the bias functions. A few were tried for a small set of problem instances. The range of bias functions used in this simple tuning were determined by the general guidelines outlined in Chapter 4. That is, for the LST and MST heuristic, we considered stronger bias functions (such as the polynomial of degree 10) because of the way we have these heuristics defined. That is, they are defined as a fraction of 1 over a potentially large number – thus they tend to be small real values less than one in a small range and require a stronger bias to spread out the values. The MTS and LPF values are already fairly well spread so somewhat weaker bias functions were considered. Our initial intuition regarding the RSM heuristic was that it would require a

Algorithm 9.3: VBSS Enhanced Priority-Rule Method for RCPSP/max: The Serial Schedule Generation Scheme
Input: The initial earliest and latest start times of the activities, ES_i, LS_i, the weights of the project network $\delta_{i,j}$, the immediate predecessors of the activities Pred_i, a maximum number of unscheduling steps (backtracks) $u_{\max}$, a heuristic h, and a bias function b.
Output: A set of assignments to the start times S_i.
VBSS-SERIAL-SCHEDULE-GENERATION(ES_i, LS_i, $\delta_{i,j}$, Pred_i, $u_{\max}$, h, b)
(1) $S_0 \leftarrow 0$
(2) Add a_0 to scheduled set
(3) $u \leftarrow 0$
(4) initialize all resource profiles
(5) **while** not all activities scheduled
(6) **if** some but not all activities from some cycle structure CS scheduled
(7) let the eligible set be all unscheduled activities in CS such that all of its predecessors have been scheduled
(8) **else**
(9) let the eligible set be all activities such that all predecessors have been scheduled and such that all of the predecessors of the activities in its cycle structure have either been scheduled already or are in that same cycle structure
(10) $j^* \leftarrow$ the eligible activity chosen randomly by VBSS using heuristic h and bias function b
(11) $t^* \leftarrow$ the earliest resource feasible start time $\geq ES_{j^*}$
(12) **if** $t^* > LS_{j^*}$
(13) $u \leftarrow u + 1$
(14) **if** $u > u_{\max}$
(15) maximum number of unscheduling steps reached
(16) **return** the empty solution
(17) UNSCHEDULE(j^*, $t^* - LS_{j^*}$)
(18) **else**
(19) Schedule j^* at time t^*
(20) $S_{j^*} = t^*$
(21) Add j^* to scheduled set
(22) update all resource profiles
(23) **foreach** not yet scheduled activity i
(24) $ES_i = \max(ES_i, S_{j^*} + \delta_{j^*,i})$
(25) $LS_i = \min(LS_i, S_{j^*} - \delta_{i,j^*})$

Algorithm 9.4: QD-BEACON/VBSS Iterative Priority-Rule Method
Input: The initial earliest and latest start times of the activities, ES_i, LS_i, the weights of the project network $\delta_{i,j}$, the immediate predecessors of the activities Pred$_i$, a maximum number of unscheduling steps (backtracks) $u_{\max}$, heuristic/bias function pairs $\{h_i, b_i\}$, and a maximum number of iterations I.
Output: A set of assignments to the start times S_i.
QD-BEACON-ITERATIVE-PRIORITY-RULE-METHOD(ES_i, LS_i, $\delta_{i,j}$, Pred$_i$, $u_{\max}$, $\{h_i, b_i\}$, I)
(1) QD-BEACONINIT($\{h_i, b_i\}$)
(2) bestsofar $\leftarrow$ best of the calls to SERIAL-SCHEDULE-GENERATION(ES_i, LS_i, $\delta_{i,j}$, Pred$_i$, $u_{\max}$, h_i)
(3) **for** 1 **to** I
(4) {heuristic, bias} $\leftarrow$ GETHEURISTICBIASFUNCTIONPAIR()
(5) Schedule $\leftarrow$ VBSS-SERIAL-SCHEDULE-GENERATION(ES_i, LS_i, $\delta_{i,j}$, Pred$_i$, $u_{\max}$, heuristic, bias)
(6) UPDATEAQDF(heuristic, bias, makespan(Schedule))
(7) **if** makespan(Schedule) $<$ makespan(bestsofar)
(8) bestsofar $\leftarrow$ Schedule

stronger bias function – in the order of that required by the LST and MST heuristics – but as it turns out, the RSM heuristic by itself does not generally perform well when randomized.

We will refer to the QD-BEACON/VBSS Iterative Priority-Rule Method using the above five heuristic / bias function pairs for N iterations according to the AQDF estimation method as follows:

- NORM[N]: N iterations using Normal distribution estimates.
- KDE[N]: N iterations using kernel density estimates.
- GEV[N]: N iterations using GEV distribution estimates.

9.7 Results

Table 9.2 shows a summary of the results of using VBSS with the priority-rule method and of the QD-BEACON/VBSS Iterative Priority-Rule Method. Table 9.1 lists the problem instances for which we were able to improve upon the current best known solutions. We can make a number of observations:

- Looking at the VBSS enhanced priority-rule method results, we can see that for any number of iterations, the best single heuristic to use in terms of the number of optimal

Table 9.1: New best known solutions found by using QD-BEACON/VBSS within a randomized iterative priority-rule method. LB is the lower bound for the makespan.

Instance	LB	Previous Best	New Best	Algorithm(s)
C364	341	372	**365**	MTS[100], NORM[100], KDE[100]
D65	440	539	**521**	NORM[2000], KDE[2000], GEV[2000]
D96	434	450	**445**	LPF[20], KDE[100], GEV[100]
D127	428	445	**434**	LPF[200]
D277	558	575	**569**	NORM[2000], KDE[2000], GEV[2000]

solutions found by the method is the "longest-path following first" (LPF) heuristic. Furthermore, VBSS and LPF is able to improve upon the best known solutions to a couple of problem instances. However, we can also observe that the VBSS method using LPF is worst in terms of the number of feasible solutions found. Using LPF and VBSS appears to perform very well on the problem instances for which it can find feasible solutions, while at the same time having difficulties finding any feasible solution for a large number of other problem instances.

- We can observe similar behavior when the second best heuristic in terms of number of optimal solutions found (VBSS using the "most total successors" (MTS)) is considered. VBSS using MTS is also able to improve upon the best known solutions to a couple of problem instances. However, like VBSS using LPF, VBSS using MTS performs poorly in terms of finding feasible solutions to the problems of the benchmark set.

- Although VBSS using any of the other three heuristics does not perform as well in terms of finding optimal solutions as compared to using LPF or MTS, using these other heuristics allows the search to find feasible solutions for many more of the problem instances. Thus, we can see that by combining the five heuristics either by the naive strategy or by using QD-BEACON, that we can find feasible solutions to nearly all of the 1059 problem instances on average; while at the same time combining the strengths of the individual heuristics in terms of finding optimal, or near-optimal, solutions.

- Comparing the use of QD-BEACON to guide the choice of search heuristic to the naive strategy of giving an equal number of iterations to each of the five heuristics, we see that the QD-BEACON method using any of the three estimation models always performs better than the naive strategy. For any number of iterations considered,

the naive strategy is always the worst in terms of the number of optimal solutions found. Furthermore, somewhat more interestingly, it is also always worst in terms of CPU time. Despite the overhead required by QD-BEACON for the estimation of the AQDFs, the naive strategy appears to be generally slower – as much as 2.5 seconds slower in the 2000 iteration case. The reason for this is that although QD-BEACON has extra computational overhead in modeling the AQDFs it is also able to take advantage of these models, giving less iterations to heuristics that appear less likely to even find a feasible solution. The naive strategy results in more iterations that do not even find a feasible solution, thus performing the maximum number of unscheduling steps allowed by the serial generation scheme for such infeasible iterations.

- Of the three methods for estimating AQDFs considered, kernel density estimation performs best for the RCPSP/max problem. Except for $N = 100$, KDE[N] finds more optimal solutions than the other considered methods. Furthermore, KDE[N] requires significantly less CPU time than does GEV[N] (at least it does for the particular estimation procedure of the GEV distribution employed here). Also, the additional overhead of KDE[N] compared to NORM[N] appears to be negligible given the CPU timing results.

Table 9.3 lists the results of a comparison of QD-BEACON/VBSS Iterative Priority-Rule Method and various other algorithms, including branch-and-bound approaches and stochastic search algorithms. We can make the following observations:

- The best performing heuristic method is clearly KDE[N]. In approximately 1/6 of the CPU time used by the previous best performing heuristic method – ISES – KDE[1000] finds as many optimal solutions with a significantly lower average deviation from the known lower bounds. In less than 1/3 of the CPU time required by ISES, KDE[2000] consistently finds as many feasible solutions as ISES; KDE[2000] consistently finds more optimal solutions than ISES; and KDE[2000] on average finds solutions that deviate significantly less from the known lower bounds as compared to ISES.

- All of the branch-and-bound algorithms considered (except for B&B$_{DRH98}$) are able to find more optimal solutions than KDE[2000]. However, in approximately 1/6 the amount of CPU time, KDE[2000] on average performs as well as the best branch-and-bound algorithm – B&B$_{DPP98}$ – in terms of deviation from the known lower bounds (and better than B&B$_{DPP98}$ for the best run of KDE[2000]). KDE[2000] is a

Table 9.2: Summary of the results of using VBSS with the priority-rule method and of the QD-BEACON/VBSS Iterative Priority-Rule Method.

Algorithm	Δ_{LB}	NO	NF	TIME
LPF[20]	4.4 (4.2)	616 (628)	942 (956)	0.4
LST[20]	6.1 (5.7)	600.7 (612)	1041 (1043)	0.2
MTS[20]	4.6 (4.4)	600 (617)	953.3 (965)	0.4
MST[20]	6.6 (6.1)	598.3 (606)	1038.7 (1041)	0.3
RSM[20]	8.5 (7.7)	447.7 (494)	1027.3 (1031)	0.2
LPF[100]	4.2 (4.0)	632.3 (642)	959.7 (969)	1.1
MTS[100]	4.4 (4.3)	626 (638)	970 (981)	1.1
LST[100]	5.5 (5.2)	617.3 (625)	1044 (1044)	0.6
MST[100]	5.9 (5.6)	609 (614)	1042 (1043)	0.8
RSM[100]	7.4 (6.9)	510.3 (536)	1033.3 (1035)	0.6
LPF[200]	4.1 (4.0)	638.7 (647)	965 (974)	2.1
MTS[200]	4.3 (4.2)	634(648)	979 (986)	2.0
LST[200]	5.3 (5.1)	625.3 (633)	1044.3 (1045)	1.1
MST[200]	5.7 (5.5)	614.3 (623)	1043.3 (1044)	1.5
RSM[200]	7.1 (6.5)	529.7 (555)	1034.7 (1036)	1.0
LPF[400]	4.1 (4.0)	643.3 (650)	972.3 (980)	4.0
MTS[400]	4.3 (4.2)	641.7 (654)	983.7 (989)	3.7
LST[400]	5.2 (4.9)	631 (638)	1044.7 (1045)	1.9
MST[400]	5.5 (5.3)	619 (629)	1043.7 (1045)	2.8
RSM[400]	6.7 (6.3)	544 (564)	1035.7 (1037)	1.7
GEV[100]	5.3 (4.9)	650.7 (667)	1050.7 (1053)	0.8
KDE[100]	5.3 (4.9)	649.7 (662)	1050.7 (1053)	0.8
NORM[100]	5.3 (4.9)	648.7 (661)	1050.7 (1053)	0.8
NAIVE[100]	5.3 (5.0)	646.3 (650)	1050 (1052)	0.9
KDE[500]	4.8 (4.6)	665.7 (680)	1053 (1055)	3.1
NORM[500]	4.9 (4.6)	662.3 (673)	1053 (1055)	3.0
GEV[500]	4.9 (4.6)	660 (677)	1053 (1055)	3.2
NAIVE[500]	4.8 (4.6)	658.3 (666)	1052.7 (1055)	3.7
KDE[1000]	4.7 (4.5)	670.3 (683)	1054.7 (1057)	5.8
GEV[1000]	4.8 (4.5)	667(682)	1054.7 (1057)	6.5
NORM[1000]	4.8 (4.5)	666.7 (678)	1054.7 (1057)	5.8
NAIVE[1000]	4.7 (4.5)	664.7 (673)	1054.7 (1057)	7.0
KDE[2000]	4.6 (4.4)	675.7 (689)	1057 (1059)	11.2
NORM[2000]	4.7 (4.4)	672.3 (685)	1057 (1059)	11.0
GEV[2000]	4.7 (4.4)	672.3 (685)	1057 (1059)	13.0
NAIVE[2000]	4.6 (4.4)	669.7 (678)	1057 (1059)	13.5

Table 9.3: Comparison of the QD-BEACON/VBSS Iterative Priority-Rule Method with various other algorithms for the RCPSP/max problem.

Algorithm	Δ_{LB}	NO	NF	TIME
BEST	3.3	789	1059	–
B&B$_{DRH98}$	n/a	606	1009	n/a[c]
B&B$_{S98}$	6.9	684	1059	66.7[a]
B&B$_{F98}$	7.0	768	1059	66.7[a]
B&B$_{DPP98}$	4.6	774	1059	66.7[a]
PR$_{FNS5}$	6.5	603	991	0.2
PR$_{FN10}$	7.7	601	1053	n/a[c]
TS	5.8	593	1059	n/a[c]
SA	5.7	630	1059	n/a[c]
GA	5.3	634	1059	n/a[c]
ISES	8.0 (7.3)	669.8 (683)	1057 (1059)	35.7[b]
KDE[1000]	4.7 (4.5)	670.3 (683)	1054.7 (1057)	5.8
KDE[2000]	4.6 (4.4)	675.7 (689)	1057 (1059)	11.2

[a] Adjusted from original publication by a factor of $\frac{200}{300}$. These branch-and-bound algorithms were implemented on a 200 Mhz Pentium, while we used for our algorithms a Sun Ultra 10 / 300MHz.

[b] Adjusted from original publication by a factor of $\frac{266}{300}$. ISES was originally implemented on a Sun UltraSparc 30 / 266 MHz, while we used for our algorithms a Sun Ultra 10 / 300MHz.

[c] Timing results were not available in some cases. This is indicated by "n/a".

competitive alternative to truncated branch and bound if one requires good solutions but not necessarily optimal solutions in a highly limited amount of time.

9.8 Summary

In this Chapter we considered the RCPSP/max problem. RCPSP/max is a very difficult makespan minimization scheduling problem for which finding feasible solutions alone is NP-hard. We used VBSS and QD-BEACON to enhance a priority-rules method for the problem. The resulting algorithm that we call the QD-BEACON/VBSS Iterative Priority-Rule Method is the new best performing heuristic method for the RCPSP/max problem. It outperforms the previous best heuristic algorithm (ISES), finding optimal solutions to more

problem instances in less CPU time than ISES. QD-BEACON/VBSS Iterative Priority-Rule Method is also competitive to some of the best branch-and-bound approaches to the problem, finding (on average) solutions that deviate less from known lower bounds as compared to branch-and-bound algorithms in a fraction of the time, despite the branch-and-bound algorithms' ability to find significantly more optimal solutions than the QD-BEACON approach. If one requires good solutions but not necessarily optimal solutions in a highly limited amount of time, then the QD-BEACON/VBSS Iterative Priority-Rule Method is a viable and effective alternative to truncated branch-and-bound. Finally, using the QD-BEACON/VBSS Iterative Priority-Rule Method we were able to improve upon the current best known solutions to 5 of the 1080 problem instances (or 5 of the 270 problem instances for which the optimal solution is not currently known).

Chapter 10

Conclusion

10.1 Summary

In this thesis, we considered stochastic search algorithms and their strength as robust, scalable problem solvers. Tools and algorithms were designed and analyzed that allow for effective performance enhancement of existing stochastic search algorithms as well as for the design of new, hybrid search algorithms that combine the problem solving strengths of multiple heuristics. The ideas encapsulated in this thesis are closely related to the concept of an algorithm portfolio. In fact, the QD-BEACON framework of this thesis can be viewed as an extension of the algorithm portfolio concept. As you may recall, an algorithm portfolio executes multiple search algorithms in parallel either on multiple processors or interleaved on a single processor. The QD-BEACON framework of this thesis can be used for an effective search control mechanism for such an algorithm portfolio. Much of the prior work with algorithm portfolios gives an equal amount of computation time to each of the algorithms in the portfolio. By using the QD-BEACON framework, statistical models of the quality of solutions generated by each of the algorithms can be computed online and used as a control strategy for the algorithm portfolio – determining how many cycles to allocate to each of the interleaved search strategies.

In the algorithmic exploration of this thesis, we began by presenting a new stochastic sampling algorithm called VBSS. VBSS is a value-biased alternative to the rank-biased stochastic sampling algorithm known as HBSS. We also considered a novel mode of computation for the stochastic decisions within VBSS based on a computational model of wasp social hierarchy formation. This lead to a variation of VBSS that we call WHISTLING. The power of the VBSS (or WHISTLING) algorithm, as compared to HBSS, is its abil-

ity to make better use of the discriminatory power inherent in search heuristics within the stochastic sampling framework. This superior search performance was demonstrated when we showed that the VBSS framework is able to find significantly better solutions to weighted tardiness sequencing problems in significantly less computation time as compared to HBSS.

We also showed that VBSS can be used to seed the starting solution configurations of a multistart hill-climber to enhance the search performance of that local search algorithm. This is somewhat contrary to popular belief that multistart hill-climbers perform better when the starting configurations are random and unbiased. However, for the problems considered in this thesis, we have strong heuristics at our disposal. These heuristics appear to be very well-informed in a large number of cases. Therefore, when using a randomization of these strong heuristics within a multistart hill-climber, we do see a benefit over unbiased random starts since each of these stochastic samples is likely to start the search at or near an already good solution. For weighted tardiness sequencing with sequence-dependent setups, we saw that such an approach has the ability to find better solutions as compared to truncated systematic heuristic search algorithms (such as LDS or DDS) in significantly less CPU time. Part of the reason for this result is the size of the problem space in this domain. LDS and DDS are forced to explore many heuristically unattractive solution trajectories (i.e., solution trajectories with only 1 or 2 discrepancies, but where the 1 or 2 discrepancies are large in heuristic value) due to their systematic nature, while VBSS can avoid such solution trajectories. In the variation of the problem without setups, VBSS was seen to be able to enhance an algorithm called multistart dynasearch, contrary to the untested claim of the authors of the multistart dynasearch algorithm.

Next we defined an algorithmic tool that we call an AQDF as the distribution of solution qualities produced by an iterative stochastic search algorithm. It is an algorithm dependent model – modeling the performance of any single algorithm. It is a problem instance dependent model – modeling the performance of the given algorithm on a single problem instance. The AQDF is related to the concept of a performance profile (PP) in that it can be viewed as a detailed model of the time slice of a PP associated with single iteration runs of an iterative stochastic search algorithm (especially stochastic sampling algorithms). The AQDF can be used to model the quality of solutions produced across the iterations of a stochastic sampling algorithm or it can be used to model the quality of the local optima discovered by a hill-climbing algorithm.

The final, but central, algorithmic tool developed and presented in this thesis is that of

the QD-BEACON framework. QD-BEACON provides the functionality to estimate the statistical model of the AQDF online during a search. Furthermore, QD-BEACON provides the functionality to use these models to guide stochastic search algorithms for more effective search control. For example, QD-BEACON can be used to choose from among multiple search heuristics on an iteration by iteration basis within a stochastic sampling algorithm. For weighted tardiness sequencing problems, we showed that using QD-BEACON in such a manner can lead to a more effective search algorithm as compared to using any single heuristic as well as compared to using the naive strategy of giving an equal number of iterations to each heuristic. We also showed that QD-BEACON can be used to model the local optima of a single-start local search algorithm with the ability to escape local optima. Given such ability, we are able to interleave the execution of multiple copies of this single-start algorithm where each copy of the algorithm begins in a different region of the search space. QD-BEACON is used to allocate compute time among the multiple interleaved copies according to the distributions of local optima encountered by each. This technique was used to enhance the previous best performing algorithm for the weighted tardiness scheduling problem. Finally, QD-BEACON, in conjunction with the VBSS algorithm, was used to enhance a backtracking, heuristic-guided CSP search algorithm for the RCPSP/max problem. VBSS was used to randomize the heuristic activity-selection heuristics and QD-BEACON was used to select from among a set of state-of-the-art heuristics. The resulting algorithm, that we call the QD-BEACON/VBSS Iterative Priority-Rule Method, is the new best-performing heuristic algorithm for the RCPSP/max problem.

Below, in Section 10.2, we summarize the contributions of this thesis. We conclude this thesis with Section 10.3 which considers possible future refinements and extensions of the research of this thesis.

10.2 Contributions

10.2.1 VBSS and WHISTLING

- **Exploiting the inherent discriminatory power of heuristics**.
  The VBSS framework was developed as a value-biased alternative to the rank-biased HBSS algorithm. In using the actual heuristic values within the stochastic decisions of the sampling algorithm, VBSS is better able to leverage the inherent discriminatory power of a search heuristic. This is both theoretically and experimentally valid.

In theory, HBSS uses only the order of the choices given by the heuristic; while VBSS instead uses the heuristic values directly, incorporating the complete information available within the heuristic. We experimentally demonstrated this improved search performance on a weighted tardiness scheduling problem.

- **Improved decision-making efficiency**.
  Since the need to rank-order the choices at each step is alleviated in the value-biased approach of VBSS, the stochastic decisions can be computed in an efficient $O(n)$ time (small constant multiplier), rather than the $O(n \log n)$ time required by the obvious implementation of the rank-biased approach of HBSS. Though we do not rule out the possibility of developing an $O(n)$ rank-biased decision scheme, such a scheme would necessarily operate with a much larger constant multiplier compared to the value-biased approach.

- **One-pass computation**.
  The dominance tournaments of WHISTLING allow each search decision to be made by a single pass through the choices, rather than two – small, but useful speedup.

10.2.2 The QD-BEACON Framework and the AQDF

- **Descriptive tool for analysis of quality distributions**.
  The AQDF is a tool that can be used to model the distribution of solutions given by single iteration runs of a stochastic search algorithm. It is an algorithm dependent model – modeling the performance of any single algorithm. It is a problem instance dependent model – modeling the performance of the given algorithm on a single problem instance. The AQDF is related to the concept of a performance profile (PP) in that it can be viewed as a detailed model of the time slice of a PP associated with single iteration runs of an iterative stochastic search algorithm (especially stochastic sampling algorithms).

- **The ability to use an AQDF for online prescriptive search guidance**.
  The QD-BEACON framework provides a number of functions that allow iterative stochastic search algorithms to model AQDFs and use them online for effective search guidance.

- **Methods for estimating an AQDF**.
  Three different methodologies were developed for estimating an AQDF. The first

assumed the AQDF was a normal distribution. The second made no distribution assumption and used kernel density estimation (KDE). The KDE approach computes its window width under the assumption of a type I extreme value distribution (Gumbel distribution). The third method was inspired by the body of work on extreme value theory and models the AQDF using a maximum likelihood estimate of the generalized extreme value (GEV) distribution. Of these three method, the KDE approach is best. Using normal estimates leads to an overly optimistic view of heuristic performance for heuristics that have a high variance in the quality of results produced and often provides the search with misleading guidance. The GEV distribution offers a much closer fit to the actual distributions that are modeled by an AQDF. Theoretically this is true since we argued that the AQDF of a stochastic sampler guided by a strong heuristic samples from the extreme of the underlying distribution of the search space (or similarly, that the distribution of local optima encountered by a state-of-the-art local search algorithm is also at the extreme of the search space). However, the maximum likelihood estimated of the GEV for many of the algorithms of this thesis turn out to follow the type of extreme value distribution that is bounded from below (for a minimization problem). The number of samples we are interested in using to model AQDFs is very small relative to that of the theory behind the GEV. The result tends to be a GEV that is bounded from below too tightly to the set of samples, thus offering search guidance that is too pessimistic. The KDE approach appears to be a nice balance of the approaches. Using a scale parameter motivated by the GEV, the KDE approach produces AQDF estimates with characteristics similar to the GEV, while preserving some amount of optimism in the left-hand tail of the AQDF.

- **Theoretically motivated exploration policy**.
  A new multiarmed bandit problem – The Max K-Armed Bandit Problem – was posed and analyzed. The result is that to maximize the expected max single sample reward of a series of N pulls from a multiarmed bandit, the number of samples given the observed best arm should grow double exponentially in the number of samples given each of the other arms. This result motivated the use of Boltzmann exploration with an exponentially decreasing temperature parameter as the exploration policy of choice within the QD-BEACON framework, which empirically worked much better than more conservative cooling schedules.

10.2.3 Applications

- **Weighted Tardiness with Sequence-Dependent Setups**.
  It was demonstrated that VBSS is able to find better solutions for this problem in less CPU time than using the rank-biased approach of HBSS as a proof of the inherent hypothesis of the design of VBSS. We also saw that results could be greatly improved with minimal additional computational overhead by applying a simple local hill-climber to the solutions of each iteration. Another significant result is that this multistart hill-climber is able to find better solutions than truncated systematic search procedures such as LDS and DDS in significantly less time while exploring a small fraction of the factorially-sized search-space. While the systematic procedures get bogged down exploring poor regions of the search-space, stochastic samplers such as VBSS are able to avoid some poor regions of the search-space by considering the discriminatory power of the heuristic values. LDS and DDS systematically consider solutions in order of increasing discrepancy from the heuristic path, but many small discrepancy (one or two discrepancies) solution paths lead to poor solutions. This can be evident in the values given by a strong heuristic (e.g., high value given to one choice and an extremely small value given to the discrepancy).

- **Weighted Tardiness (no setups)**.
  Contrary to the assumptions of the authors of the multistart dynasearch algorithm, we found that using VBSS and QD-BEACON with search heuristics to bias the starting solution configurations results in significant improvement in performance as compared to unbiased starting solution configurations. We also effectively used QD-BEACON to enhance the performance of the previous best known algorithm for the problem – Iterated Dynasearch. Our new algorithm – QD-BEACON Enhanced Iterated Dynasearch – is now the current best performing algorithm on this widely used set of benchmarks for the weighted tardiness sequencing problem.

- **RCPSP/max**.
  Using VBSS and QD-BEACON, we enhanced a state-of-the-art priority-rule based search algorithm for the problem. The resulting algorithm is superior to the previous best performing heuristic search algorithm (ISES) for the problem and competitive with a number of truncated branch-and-bound approaches to the problem. The QD-BEACON/VBSS Iterative Priority-Rule Method is also able to improve upon the best known solutions to a few of the problem instances in the benchmark set.

10.3 Refinements and Extensions

- **Applying framework to additional problem domains**.
  In this thesis, we used the QD-BEACON and VBSS frameworks to design and enhance the problem solving performance of stochastic search algorithms in a number of optimization domains, including: 1) the weighted tardiness sequencing problem with sequence-dependent setups; 2) the weighted tardiness sequencing problem (no setups); and 3) the resource constrained project scheduling with time windows. There are volumes of combinatorial optimization problems which can potentially benefit from solution procedures involving VBSS and/or the QD-BEACON framework.

- **Evolving the Scheduler component of the FIRE system to new problems**.
  One of the components of the planning layer of the rovers in the FIRE architecture [89, 90] is a scheduler. This Scheduler component is in charge of solving any scheduling problem that the rover requires in order to place bids on tasks or in order to evaluate bids from other rovers for its own tasks. Currently, the particular scheduling problems faced by the Scheduler component of the FIRE system are essentially TSP instances. This Scheduler component solves these TSP instances using the VBSS algorithm of Chapter 4 of this thesis. With the volumes of work existing for the TSP problem, the current instantiation of the Scheduler component of the FIRE system is not obviously ground-breaking work. But the goal was not to develop the next best TSP algorithm. Rather, the goal was to develop a scheduling component with the flexibility to adapt to different and/or more complex scheduling constraints and objective functions. As the FIRE Project progresses, the VBSS based Scheduler component can be easily adapted as the need to solve more complex scheduling problems arises. For example, if the optimization objective was changed to minimizing weighted tardiness, then we can simply pull out the TSP search heuristic and plug-in one of the heuristics used in either Chapter 5 or Chapter 8 and potentially combine these using the QD-BEACON framework. Also, as more complex timing constraints are considered such as tightly coupled rover coordination/timing constraints, the new results obtained for RCPSP/max in Chapter 9 can be plugged into the Scheduler component of the rovers.

- **Enhancing other types of search algorithms with QD-BEACON**.
  The QD-BEACON framework is not limited to boosting the performance of stochastic sampling algorithms. In fact, in this thesis, we have seen examples of how QD-

BEACON can be used to enhance the performance of a number of stochastic search algorithms. First, we saw how QD-BEACON could be used to combine multiple search heuristics in a stochastic sampling algorithm. Second, we saw how using QD-BEACON, VBSS, and multiple search heuristics could be used to enhance multistart hill-climbing algorithms – for example, the QD-BEACON/VBSS Enhanced Multistart Dynasearch algorithm. Third, QD-BEACON was used outside the domain of multistart algorithms when it was used to interleave, and control the amount of CPU time given to, multiple simultaneously executing iterated dynasearches – the QD-BEACON Enhanced Iterated Dynasearch algorithm. Finally, QD-BEACON and VBSS were used to create an iterative multi-pass version of a priority-rule method in the constrained optimization domain of RCPSP/max – the QD-BEACON/VBSS Iterative Priority-Rule Method. This algorithm combines stochastic sampling within a backtracking heuristic-guided CSP search algorithm and uses QD-BEACON to select from among competing heuristics. Other stochastic search algorithms could benefit from the application of the QD-BEACON framework. For example, it could perhaps be used within a genetic algorithm to choose from among multiple crossover or mutation operators. Or, it might be used within an algorithm portfolio – as is essentially done in the QD-BEACON Enhanced Iterated Dynasearch algorithm – to control the amount of computation given each algorithm in the portfolio. It may also be useful to revisit some of the related work ideas, using the more detailed models available within the QD-BEACON framework. For example, Sadeh's expected cost improvement distributions used to detect unpromising runs of simulated annealing may benefit from the extreme value theory motivation of the QD-BEACON framework.

- **Learning bias functions**.
  The QD-BEACON framework is already designed to allow one to consider multiple bias functions for each heuristic (i.e., by modeling the AQDF of a heuristic / bias function pair). To some extent this is not always necessary. For example, rationale was given for selecting a bias function for the VBSS algorithm according to the types of values given by your choice of heuristic function. However, it may not always be an easy decision to make a priori. One possible solution is to use the QD-BEACON framework with heuristic / bias function pairs. Another possibility might be to adapt the bias function according to some learned model of the range of heuristic values (e.g., by tracking the heuristic values) or perhaps adapting the bias function according

to the distribution of the quality of solutions (e.g., weakening the bias to expand a search for which the AQDF is hitting a "wall").

- **Asymmetric kernel functions**.
  As has been demonstrated, the best performing estimation method for the AQDF within the QD-BEACON framework is that of kernel density estimation. If you recall, the kernel function chosen for KDE within QD-BEACON was the Epanechnikov kernel. It was chosen primarily for computational efficiency reasons that were outlined – it is bounded and it is easy to compute its integral. It might be desirable, however, to consider an asymmetric kernel density estimator [18]. For the problem domains considered in this thesis, the theoretical lower bound for any problem instance is bounded below by zero. Also, many of the AQDFs modeled by the framework for these problem domains are strongly skewed to the right and appear to be bounded on the left. Asymmetric kernel density estimators are specifically designed for densities defined on $[0, \infty)$. A couple of examples include the Gamma kernel [34], the Inverse Gaussian [171], and the Reciprocal Inverse Gaussian [171]. For our purposes, these may or may not be of use since they would make our required computations more complex. But it might be possible to design such an asymmetric kernel that does suit our computational needs.

- **Going beyond assumption of search time per iteration invariance**.
  The QD-BEACON framework, as defined, assumes that the amount of search time required by each possible choice of search heuristic is approximately the same for all of its choices. In the algorithms presented in this thesis, this holds for the most part. For example, in the sequencing problems, using VBSS to compute a single solution always requires approximately the same amount of CPU time. Adding the hill-climber on top of the VBSS solution adds almost negligible CPU time due to the very short hill-climbs typically required by the algorithms of this thesis to reach local optima, so the invariance assumption still approximately holds in this case. In the QD-BEACON Enhanced Iterated Dynasearch algorithm, each dynasearch to a local optima requires very few steps, so in using QD-BEACON to interleave the simultaneous Iterated Dynasearches, the invariance assumption yet again approximately holds. The one case in this thesis where the search time per iteration invariance assumption does not hold is the QD-BEACON/VBSS Iterative Priority-Rule Method for the RCPSP/max problem. However, in this algorithm, heuristics that are likely

to often lead to infeasible solutions are not sampled as frequently as those that often find feasible solutions. It is these infeasible runs that require the most time. Therefore, although the invariance assumption does not hold, the behavior of the algorithm is favorable just the same. There are other cases, however, for which it could be a detriment to problem solving performance if the invariance assumption does not hold. For example, in a sequencing problem, using the VBSS algorithm, it might be possible to have two heuristics which require drastically different amounts of CPU time for each call to the heuristic function. If this was the case, then it would be important to extend the QD-BEACON framework to consider this, in addition to the model of the AQDF, in choosing which heuristic to use for each iteration of VBSS. For example, if VBSS with heuristic h_1 takes twice as much time per iteration as compared to heuristic h_2, then perhaps you want to instead consider the probability of finding a better solution than the best found so far given 2 iterations with h_2 as compared to the probability given a single iteration with h_1.

- **Using QD-BEACON to distribute search among solving multiple problem instances**.
  Another potentially interesting problem to which to apply the QD-BEACON framework is that of distributing a limited amount of computation time to the solving of multiple problem instances. This is one problem for which the anytime computation community has spent a great deal of effort. Perhaps, the modeling methods and exploration strategy of the QD-BEACON framework can be used in conjunction with some of the anytime computation tools such as performance profiles to further enhance this distribution of compute time among multiple problem instances.

Bibliography

[1] L. Adler, N. M. Fraiman, E. Kobacker, M. Pinedo, J. C. Plotnitcoff, and T. P. Wu. BPSS: a scheduling system for the packaging industry. *Operations Research*, 41:641–648, 1993.

[2] M. S. Akturk and M. B. Yildirim. A new lower bounding scheme for the total weighted tardiness problem. *Computers and Operations Research*, 25(4):265–278, 1998.

[3] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. In *Proceedings of the 15th International Conference on Machine Learning*, pages 100–108. Morgan Kaufmann, 1998.

[4] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 2002.

[5] A. Bauer, B. Bullnheimer, R. F. Hartl, and C. Strauss. An ant colony optimization approach for the single machine total tardiness problem. In *CEC99: Proceedings of the Congress on Evolutionary Computation*, pages 1445–1450, July 1999.

[6] J. E. Beasley. OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990. http://www.ms.ic.ac.uk/info.html.

[7] J. E. Beasley. Weighted tardiness. In *OR-Library*. Imperial College Management School, 1998. Available at, http://mscmga.ms.ic.ac.uk/jeb/orlib/wtinfo.html.

[8] T. Beaumariage and K. Kempf. Attractors in manufacturing systems with chaotic tendencies, 1995. Presentation at INFORMS-95, New Orleans, http://www.informs.org/Conf/NewOrleans95/ TALKS/TB07.3.html.

[9] R. E. Bellmann. *Dynamic Programming*. Princeton University Press, 1957.

[10] N. Benvenuto, M. Marchesi, G. Orlandi, F. Piazza, and A. Uncini. Finite wordlength digital filter design using an annealing algorithm. In *Proceedings of ICASSP-89: IEEE International Conference on Acoustic, Speech and Signal Processing*, pages 861–864, May 1989.

[11] D. A. Berry and B. Fristedt. *Bandit Problems: Sequential Allocation of Experiments*. Chapman and Hall, London, UK, 1985.

[12] M. Boddy and T. Dean. Solving time-dependent planning problems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 979–984. Morgan Kaufmann, August 1989.

[13] M. Boddy and T. Dean. Decision-theoretic deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67(2):245–286, 1994.

[14] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Santa Fe Institute Studies in the Sciences of Complexity. Oxford University Press, 1999.

[15] E. Bonabeau, A. Sobkowski, G. Theraulaz, and J. L. Deneubourg. Adaptive task allocation inspired by a model of division of labor in social insects. In D. Lundh and B. Olsson, editors, *Bio Computation and Emergent Computing*, pages 36–45. World Scientific, 1997.

[16] E. Bonabeau, G. Theraulaz, and J. L. Deneubourg. Fixed response thresholds and the regulation of division of labor in insect societies. *Bulletin of Mathematical Biology*, 60:753–807, 1998.

[17] D. Bosq. *Nonparametric Statistics for Stochastic Processes: Estimation and Prediction*. Lecture Notes in Statistics. Springer, 1998.

[18] T. Bouezmarni and O. Scaillet. Consistency of asymmetric kernel density estimators and smoothed histograms with application to income data. Technical Report STAT DP 0306, Université Catholique De Louvain, 2003.

[19] J. A. Boyan. *Learning Evaluation Functions for Global Optimization*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1998.

[20] J. A. Boyan and A. W. Moore. Using prediction to improve combinatorial optimization search. In *Proceedings of the Sixth International Workshop on Artificial Intelligence and Statistics (AISTATS-6)*, 1997.

[21] J. A. Boyan and A. W. Moore. Learning evaluation functions for global optimization and boolean satisfiability. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 1998.

[22] M. F. Bramlette. Initialization, mutation and selection methods in genetic algorithms for function optimization. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 100–107, San Mateo, CA, 1991. Morgan Kaufmann.

[23] J. Bresina, M. Drummond, and K. Swanson. Search space characterization for a telescope scheduling application. In *Working Notes of the AAAI 1994 Fall Symposium, Planning and Learning: On To Real Applications*. AAAI Press, 1994.

[24] J. Bresina, M. Drummond, and K. Swanson. Expected solution quality. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1583–1590. Morgan Kaufmann, 1995.

[25] J. L. Bresina. Heuristic-biased stochastic sampling. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference, Volume One*, pages 271–278. AAAI Press, 1996.

[26] B. Bullnheimer, R. F. Hartl, and C. Strauss. An improved ant system algorithm for the vehicle routing problem. *Annals of Operations Research*, 89:319–328, 1999.

[27] M. Campos, E. Bonabeau, G. Théraulaz, and J. Deneubourg. Dynamic scheduling and division of labor in social insects. *Adaptive Behavior*, 8(2):83–96, 2000.

[28] Y. J. Cao and Q. H. Wu. Optimization of control parameters in genetic algorithms: A stochastic approach. *International Journal of Systems Science*, 30(5):551–559, May 1999.

[29] D. C. Carroll. *Heuristic Sequencing of Single and Multiple Components*. PhD thesis, M.I.T., Massachusetts, 1965.

[30] A. Cesta, A. Oddi, and S. F. Smith. An iterative sampling procedure for resource constrained project scheduling with time windows. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1022–1029. Morgan Kaufmann, 1999.

[31] A. Cesta, A. Oddi, and S. F. Smith. A constraint-based method for project scheduling with time windows. Technical Report CMU-RI-TR-00-34, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, February 2000.

[32] A. Cesta, A. Oddi, and S. F. Smith. A constraint-based method for project scheduling with time windows. *Journal of Heuristics*, 8:109–136, 2002.

[33] H. Chen, C. Gomes, and B. Selman. Formal models of heavy-tailed behavior in combinatorial search. In *Using Uncertainty Within Computation: Papers from the 2001 AAAI Fall Symposium, Technical Report FS-01-04*. AAAI Press, 2-4 November 2001.

[34] S. X. Chen. Probability density function estimation using gamma kernels. *Annals of the Institute of Statistical Mathematics*, 52(3):471–480, 2000.

[35] S. Y. Chen, S. N. Talukdar, and N. M. Sadeh. Job shop scheduling by an asynchronous team of optimization agents. In *IJCAI-93 Workshop on Knowledge-Based Production Planning, Scheduling, and Control: Workshop Notes*, pages 73–82, August 1993.

[36] W. Y. Chiang, M. S. Fox, and P. S. Ow. Factory model and test data descriptions: OPIS experiments. Technical Report CMU-RI-TR-90-05, The Robotics Institute, Carnegie Mellon University, March 1990.

[37] V. A. Cicirello. Intelligent retrieval of solid models. Master's thesis, Department of Mathematics and Computer Science, Drexel University, Philadelphia, PA, June 1999.

[38] V. A. Cicirello. A game-theoretic analysis of multi-agent systems for shop floor routing. Technical Report CMU-RI-TR-01-28, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, September 2001.

[39] V. A. Cicirello and W. C. Regli. Resolving non-uniqueness in design feature histories. In W. F. Bronsvoort and D. C. Anderson, editors, *Fifth ACM/SIGGRAPH*

Symposium on Solid Modeling and Applications, pages 76–84. ACM Press, 9-11 June 1999. Ann Arbor, MI.

[40] V. A. Cicirello and W. C. Regli. Machining feature-based comparison of mechanical parts. In *International Conference on Shape Modeling and Applications*, pages 176–185. ACM SIGGRAPH, the Computer Graphics Society, and EUROGRAPHICS, IEEE Computer Society Press, May 2001. Genova, Italy.

[41] V. A. Cicirello and W. C. Regli. An approach to feature-based comparison of solid models of machined parts. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 16(5):385–399, November 2002.

[42] V. A. Cicirello and S. F. Smith. Modeling GA performance for control parameter optimization. In D. Whitley, D. Goldberg, E. Cantú-Paz, L. Spector, I. Parmee, and H. Beyer, editors, *GECCO-2000: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 235–242. Morgan Kaufmann Publishers, 8-12 July 2000. Las Vegas, NV.

[43] V. A. Cicirello and S. F. Smith. Ant colony control for autonomous decentralized shop floor routing. In *ISADS-2001: International Symposium on Autonomous Decentralized Systems*, pages 383–390. IEEE Computer Society Press, March 2001. Dallas, TX.

[44] V. A. Cicirello and S. F. Smith. Improved routing wasps for distributed factory control. In *The IJCAI-01 Workshop on Artificial Intelligence and Manufacturing, Working Notes*, pages 26–32. AAAI SIGMAN, 4-9 August 2001. Seattle, WA.

[45] V. A. Cicirello and S. F. Smith. Randomizing dispatch scheduling policies. In *Using Uncertainty Within Computation: Papers from the 2001 AAAI Fall Symposium, Technical Report FS-01-04*, pages 30–37. AAAI Press, 2-4 November 2001. North Falmouth, Massachusetts.

[46] V. A. Cicirello and S. F. Smith. Wasp-like agents for distributed factory coordination. Technical Report CMU-RI-TR-01-39, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 2001.

[47] V. A. Cicirello and S. F. Smith. Wasp nests for self-configurable factories. In J. P. Müller, E. Andre, S. Sen, and C. Frasson, editors, *Proceedings of the Fifth*

International Conference on Autonomous Agents, pages 473–480. ACM SIGART, ACM/SIGGRAPH, ACM/SIGCHI, ACM Press, May-June 2001. Montreal, Quebec, Canada.

[48] V. A. Cicirello and S. F. Smith. Amplification of search performance through randomization of heuristics. In P. Van Hentenryck, editor, *Principles and Practice of Constraint Programming – CP 2002: 8th International Conference, Proceedings*, volume LNCS 2470 of *Lecture Notes in Computer Science*, pages 124–138. Springer-Verlag, 7-13 September 2002. Ithaca, NY.

[49] D. A. Clark, J. Frank, I. P. Gent, E. MacIntyre, N. Tomov, and T. Walsh. Local search and the number of solutions. In *Proceedings of the Second International Conference on Principles and Practices of Constraint Programming (CP-96)*, pages 119–133, 1996.

[50] S. Coles. *An Introduction to Statistical Modeling of Extreme Values*. Springer-Verlag, 2001.

[51] R. K. Congram, C. N. Potts, and S. L. van de Velde. An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 14(1):52–67, Winter 2002.

[52] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. McGraw-Hill, 1990.

[53] P. Cowling, G. Kendall, and L. Han. An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. In *Proceedings of the Congress on Evolutionary Computation (CEC-2002)*, pages 1185–1190, May 2002.

[54] P. Cowling, G. Kendall, and E. Soubeiga. Adaptively parameterised hyperheuristics for sales summit scheduling. In *Selected Papers from the 4th Metaheuristics International Conference*. Kluwer, July 2001.

[55] P. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach to scheduling a sales summit. In *Practice and Theory of Automated Timetabling III: Third International Conference, PATAT 2000, Selected Papers*, number LNCS 2079 in Lecture Notes in Computer Science, pages 176–190. Springer-Verlag, August 2001.

[56] P. Cowling, G. Kendall, and E. Soubeiga. A parameter-free hyperheuristic for scheduling a sales summit. In *Proceedings of the 4th Metaheuristics International Conference*, pages 127–131, July 2001.

[57] P. Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics: A robust optimisation method applied to nurse scheduling. In J. J. Merelo, P. Adamidis, and H. G. Beyer, editors, *Parallel Problem Solving from Nature – PPSN VII: 7th International Conference, Proceedings*, number LNCS 2439 in Lecture Notes in Computer Science, pages 851–860. Springer-Verlag, September 2002.

[58] P. Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation. In S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, and G. R. Raidl, editors, *Applications of Evolutionary Computing: EvoWorkshops 2002 Proceedings*, number LNCS 2279 in Lecture Notes in Computer Science, pages 1–10. Springer-Verlag, April 2002.

[59] H. A. J. Crauwels, C. N. Potts, and L. N. Van Wassenhove. Local search heuristics for the single machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 10(3):341–350, Summer 1998.

[60] R. Davidson and D. Harel. Drawing graphs nicely using simulated annealing. *ACM Transactions on Graphics*, 15(4):301–331, October 1996.

[61] K. De Jong. Adaptive system design: A genetic approach. *IEEE Transactions on Systems, Man, and Cybernetics*, 10(9):566–574, 1980.

[62] K. A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, Ann Arbor, MI, 1975.

[63] B. De Reyck and W. Herroelen. A branch-and-bound procedure for the resource-constrained project scheduling problem with generalized precedence constraints. *European Journal of Operational Research*, 111:152–174, 1998.

[64] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 49–54. AAAI Press, August 1988.

[65] M. den Besten, T. Stützle, and M. Dorigo. Ant colony optimization for the total weighted tardiness problem. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao,

E. Lutton, J. J. Merelo, and H. S. Schwefel, editors, *Proceedings of PPSN-VI, Sixth International Conference on Parallel Problem Solving from Nature*, volume 1917 of *Lecture Notes in Computer Science*, pages 611–620. Springer Verlag, 2000.

[66] G. Di Caro and M. Dorigo. AntNet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998.

[67] M. Dorigo and G. Di Caro. The ant colony optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, 1999.

[68] M. Dorigo and L. M. Gambardella. Ant colonies for the traveling salesman problem. *BioSystems*, 43:73–89, 1997.

[69] M. Dorigo and L. M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[70] U. Dorndorf, E. Pesch, and T. Phan-Huy. A time-oriented branch-and-bound algorithm for resource-constrained project scheduling with generalised precedence constraints. *Management Science*, 46(10):1365–1384, 2000.

[71] G. Dueck and T. Scheuer. Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, 90:161–175, 1990.

[72] A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, July 1999.

[73] H. Emmons. One-machine sequencing to minimize certain functions of job tardiness. *Operations Research*, 17:701–705, 1969.

[74] V. A. Epanechnikov. Non-parametric estimation of a multivariate probability density. *Theory of Probability and Its Applications*, 14(1):153–158, 1969.

[75] S. L. Epstein. For the right reasons: the FORR architecture for learning in a skilled domain. *Cognitive Science*, 18:479–511, 1994.

[76] S. L. Epstein, E. C. Freuder, R. Wallace, A. Morozov, and B. Samuels. The adaptive constraint engine. In P. Van Hentenryck, editor, *Principles and Practice of Constraint Programming – CP 2002: 8th International Conference, Proceedings*, volume LNCS 2470 of *Lecture Notes in Computer Science*, pages 525–540. Springer-Verlag, 2002.

[77] A. Fest, R. H. Möhring, F. Stork, and M. Uetz. Resource-constrained project scheduling with time windows: A branching scheme based on dynamic release dates. Technical Report 596, Technische Universität Berlin, Berlin, Germany, 1999.

[78] P. Forsyth and A. Wren. An ant system for bus driver scheduling. Technical Report 97.25, University of Leeds, School of Computer Studies, July 1997. Presented at the 7th International Workshop on Computer-Aided Scheduling of Public Transport, Boston, July 1997.

[79] B. Franck and K. Neumann. Priority-rule methods for the resource-constrained project scheduling problem with minimal and maximal time lags – an empirical analysis. In *The 5th International Workshop on Project Management and Scheduling*, pages 88–91, 1996.

[80] B. Franck and K. Neumann. Resource-constrained project scheduling with time windows: Structural questions and priority-rule methods. Technical Report WIOR-492, Universität Karlsruhe, Karlsruhe, Germany, 1998.

[81] B. Franck, K. Neumann, and C. Schwindt. Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling. *OR Spektrum*, 23:297–324, 2001.

[82] B. Franck and T. Selle. Metaheuristics for the resource-constrained project scheduling with schedule-dependent time windows. Technical Report WIOR-546, Universität Karlsruhe, Karlsruhe, Germany, 1998.

[83] J. Frank, P. Cheeseman, and J. Stutz. When gravity fails: Local search topology. *Journal of Artificial Intelligence Research*, 7:249–281, 1997.

[84] E. C. Freuder, R. Dechter, M. L. Ginsberg, B. Selman, and E. Tsang. Systematic versus stochastic constraint satisfaction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 2027–2032. Morgan Kaufmann, 1995.

[85] L. M. Gambardella and M. Dorigo. HAS-SOP: Hybrid ant system for the sequential ordering problem. Technical Report IDSIA 97-11, Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA), Lugano, Switzerland, 1997.

[86] L. M. Gambardella, E. Taillard, and G. Agazzi. MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. Technical Report IDSIA-06-99, Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA), Lugano, Switzerland, 1999.

[87] F. Glover. Tabu search – part I. *ORSA Journal on Computing*, 1(3):190–206, Summer 1989.

[88] F. Glover. Tabu search – part II. *ORSA Journal on Computing*, 2(1):4–32, Winter 1990.

[89] D. Goldberg, V. Cicirello, M. B. Dias, R. Simmons, S. Smith, T. Smith, and A. Stentz. A distributed layered architecture for mobile robot coordination: Application to space exploration. In *The 3rd International NASA Workshop on Planning and Scheduling for Space*, 27-29 October 2002. Houston, TX.

[90] D. Goldberg, V. Cicirello, M. B. Dias, R. Simmons, S. Smith, and A. Stentz. Market-based multi-robot planning in a distributed layered architecture. In *Multi-Robot Systems: From Swarms to Intelligent Automata: Proceedings of the 2003 International Workshop on Multi-Robot Systems*, volume 2, pages 27–38. Kluwer Academic Publishers, 17-19 March 2003. Washington, DC.

[91] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, 1989.

[92] C. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference*, pages 431–437. AAAI Press, 1998.

[93] C. P. Gomes and B. Selman. Algorithm portfolio design: Theory vs. practice. In *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence (UAI-97)*. Morgan Kaufmann, 1997.

[94] C. P. Gomes and B. Selman. Algorithm portfolios. *Artificial Intelligence*, 126:43–62, 2001.

[95] C. P. Gomes, B. Selman, and N. Crato. Heavy-tailed distributions in combinatorial search. In *Principles and Practices of Constraint Programming (CP-97)*, Lecture Notes in Computer Science, pages 121–135. Springer-Verlag, 1997.

[96] C. P. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24:67–100, 2000.

[97] A. G. Gray and A. W. Moore. Rapid evaluation of multiple density models. In C. M. Bishop and B. J. Frey, editors, *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, January 2003.

[98] J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1):122–128, Jan-Feb 1986.

[99] L. Han, G. Kendall, and P. Cowling. An adaptive length chromosome hyperheuristic genetic algorithm for a trainer scheduling problem. In *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'02)*, pages 267–271, November 2002.

[100] W. Harvey and M. Ginsberg. Limited discrepency search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 607–613. Morgan Kaufmann, 1995.

[101] J. H. Holland. Genetic algorithms and the optimal allocations of trials. *SIAM Journal on Computing*, 2(2):88–105, 1973.

[102] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press, 1975. Second Edition: MIT Press, 1992.

[103] J. N. Hooker. Testing heuristics: We have it all wrong. *Journal of Heuristics*, 1:33–42, 1995.

[104] H. Hoos and T. Stutzle. Characterizing the run-time behavior of stochastic local search. Technical Report AIDA-98-01, Darmstadt University of Technology, Germany, 1998.

[105] H. H. Hoos. *Stochastic Local Search*. PhD thesis, Darmstadt University of Technology, Germany, November 1998.

[106] E. Horvitz, Y. Ruan, C. Gomes, H. Kautz, B. Selman, and M. Chickering. A bayesian approach to tackling hard computational problems. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence (UAI-2001)*, pages 235–244, 2001.

[107] E. J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of the Third Workshop on Uncertainty in Artificial Intelligence*, pages 429–444. AAAI and Association for Uncertainty in Artificial Intelligence, July 1987.

[108] E. J. Horvitz. Reasoning under varying and uncertain resource constraints. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 111–116. AAAI Press, August 1988.

[109] J. R. M. Hosking. Algorithm AS 215: Maximum-likelihood estimation of the paramaters of the generalized extreme-value distribution. *Applied Statistics*, 34(3):301–310, 1985.

[110] J. R. M. Hosking and A. J. Macleod. Maximum likelihood estimation of the parameters of the generalized extreme-value distribution. In *StatLib: Applied Statistics Algorithms*. CMU, 1994. http://lib.stat.cmu.edu/apstat/.

[111] A. E. Howe, E. Dahlman, C. Hansen, M. Scheetz, and A. von Mayrhauser. Exploiting competitive planner performance. In S. Biundo and M. Fox, editors, *Recent Advances in AI Planning, 5th European Conference on Planning (ECP'99), Proceedings*, volume LNCS 1809 of *Lecture Notes in Computer Science*, pages 62–72. Springer-Verlag, 1999.

[112] B. A. Huberman, R. M. Lukose, and T. Hogg. An economics approach to hard computational problems. *Science*, 275:51–54, 3 January 1997.

[113] L. P. Kaelbling. *Learning in Embedded Systems*. PhD thesis, Department of Computer Science, Stanford University, Stanford, California, 1990.

[114] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, May 1996.

[115] H. Kautz, E. Horvitz, Y. Ruan, C. Gomes, and B. Selman. Dynamic restart policies. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*. AAAI Press, July 2002.

[116] K. Kempf and T. Beaumariage. Chaotic behavior in manufacturing systems. In *AAAI-94 Workshop Program, Reasoning About the Shop Floor, Workshop Notes*, pages 82–96. AAAI Press, 1994.

[117] G. Kendall, E. Soubeiga, and P. Cowling. Choice function and random hyperheuristics. In *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'02)*, pages 667–671, November 2002.

[118] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, May 1983.

[119] T. E. Koch, V. Scheer, J. Wakunda, and A. Zell. A parallel, hybrid meta-optimization for finding better parameters of an evolution strategy in real world optimization problems. In A. S. Wu, editor, *Proceedings of the 2000 Genetic and Evolutionary Computation Conference Workshop Program*, pages 17–19, July 2000. Evolutionary Computation and Parallel Processing Workshop.

[120] I. Kocsis, L. Farkas, and L. Nagy. 3G base station positioning using simulated annealing. In *The 13th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, September 2002.

[121] R. Kolisch and S. Hartmann. Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis. Technical Report 469, Christian-Albrechts-Universität zu Kiel, Kiel, Germany, February 1998.

[122] R. Kolisch and S. Hartmann. Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis. In J. Weglarz, editor, *Project Scheduling: Recent Models, Algorithms and Applications*, pages 147–178. Kluwer, Amsterdam, the Netherlands, 1999.

[123] R. Kolisch and A. Sprecher. PSPLIB: A project scheduling problem library. Technical Report 396, Christian-Albrechts-Universität zu Kiel, Kiel, Germany, March 1996.

[124] R. Kolisch, A. Sprecher, and A. Drexl. Characterization and generation of resource-constrained project scheduling problems. *Management Science*, 41:1693–1703, 1995.

[125] R. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

[126] R. Korf. Improved limited discrepancy search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference, Volume 1*, pages 286–291. AAAI Press, 1996.

[127] R. E. Korf. Linear-space best-first search: Summary of results. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-92)*, July 1992.

[128] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[129] M. G. Lagoudakis, M. L. Littman, and R. E. Parr. Selecting the right algorithm. In *Using Uncertainty Within Computation: Papers from the 2001 AAAI Fall Symposium, Technical Report FS-01-04*, pages 74–75. AAAI Press, November 2001.

[130] P. Langley. Systematic and nonsystematic search strategies. In *Artificial Intelligence Planning Systems: Proceedings of the First International Conference*, pages 145–152, 1992.

[131] K. Larson and T. Sandholm. Deliberation in equilibrium: Bargaining in computationally complex problems. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, pages 48–55, July-August 2000.

[132] K. Larson and T. Sandholm. Bargaining with limited computation: Deliberation equilibrium. *Artificial Intelligence*, 132:183–217, 2001.

[133] K. Larson and T. Sandholm. Costly valuation computation in auctions. In *Proceedings of the Eighth Conference on Theoretical Aspects of Reasoning about Knowledge (TARK)*, pages 169–182, July 2001.

[134] Y. H. Lee, K. Bhaskaran, and M. Pinedo. A heuristic to minimize the total weighted tardiness with sequence-dependent setups. *IIE Transactions*, 29:45–52, 1997.

[135] S. Lin and B. W. Kernighan. An effective heuristic for the traveling-salesman problem. *Operational Research*, 21:498–516, 1973.

[136] A. López, R. Molina, J. Mateos, and A. K. Katsaggelos. Spect image reconstruction using prior models. *International Journal of Pattern Recognition and Artificial Intelligence*, 16(3):317–330, 2002.

[137] M. Luby, A. Sinclair, and D. Zuckerman. Optimal speedup of las vegas algorithms. *Information Processing Letters*, 47(4):173–180, 1993.

[138] A. J. Macleod. Remark AS R76: A remark on algorithm AS 215: Maximum-likelihood estimation of the paramaters of the generalized extreme-value distribution. *Applied Statistics*, 38(1):198–199, 1989.

[139] U. Maulik, S. Bandyopadhyay, and M. K. Pakhira. Clustering using annealing evolution: Application to pixel classification of satellite images. In *The 3rd Indian Conference on Computer Vision, Graphics and Image Processing: Online ICVGIP-2002 Proceedings*, December 2002. http://www.ee.iitb.ac.in/˜icvgip/.

[140] K. N. McKay. The factory from hell – a modelling benchmark. In *Proceedings of the NSF Workshop on Intelligent, Dynamic Scheduling for Manufacturing Systems*, pages 99–114, June 1993.

[141] D. Merkle, M. Middendorf, and H. Schmeck. Ant colony optimization for resource-constrained project scheduling. In *GECCO-2000: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 893–900. Morgan Kaufmann, July 2000.

[142] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1996.

[143] D. Morley. Painting trucks at general motors: The effectiveness of a complexity-based approach. In *Embracing Complexity: Exploring the Application of Complex Adaptive Systems to Business*, pages 53–58. The Ernst and Young Center for Business Innovation, 1996.

[144] D. Morley and C. Schelberg. An analysis of a plant-specific dynamic scheduler. In *Proceedings of the NSF Workshop on Intelligent, Dynamic Scheduling for Manufacturing Systems*, pages 115–122, June 1993.

[145] T. E. Morton and D. W. Pentico. *Heuristic Scheduling Systems: With Applications to Production Systems and Project Management*. John Wiley and Sons, 1993.

[146] T. F. Morton and R. M. V. Rachamadugu. Myopic heuristics for the single machine weighted tardiness problem. Technical Report CMU-RI-TR-83-9, Carnegie Mellon University, Pittsburgh, PA, November 1982.

[147] V. Narayan, T. Morton, and P. Ramnath. X-Dispatch methods for weighted tardiness job shops. GSIA Working Paper 1994-14, Carnegie Mellon University, Pittsburgh, PA, July 1994.

[148] A. Nareyek. Using global constraints for local search. In E. C. Freuder and R. J. Wallace, editors, *Constraint Programming and Large Scale Discrete Optimization*, volume DIMACS 57, pages 9–28. American Mathematical Society Publications, 2001.

[149] A. Nareyek. Choosing search heuristics by non-stationary reinforcement learning. In M. G. C. Resende and J. P. de Sousa, editors, *Metaheuristics: Computer Decision Making*. Kluwer Academic Publishers, 2003.

[150] K. Neumann, C. Schwindt, and J. Zimmermann. *Project Scheduling with Time Windows and Scarce Resources: Temporal and Resource-Constrained Project Scheduling with Regular and Nonregular Objective Functions*. Lecture Notes in Economics and Mathematical Systems. Springer-Verlag, 2002.

[151] K. Neumann and J. Zhan. Heuristics for the minimum project-duration problem with minimal and maximal time lags under fixed resource constraints. *Journal of Intelligent Manufacturing*, 6:145–154, 1995.

[152] NIST/SEMATECH. *e-Handbook of Statistical Methods*. NIST/SEMATECH, 2003. http://www.itl.nist.gov/div898/handbook/.

[153] A. Oddi and S. F. Smith. Stochastic procedures for generating feasible schedules. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference*, pages 308–314. AAAI Press, 1997.

[154] A. J. Parkes. Clustering at the phase transition. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 340–345. AAAI Press, 1997.

[155] J. C. Pemberton and R. E. Korf. An incremental search approach to real-time planning and scheduling: Preliminary results. In *Proceedings of the NSF Workshop on Intelligent, Dynamic Scheduling for Manufacturing Systems*, pages 134–145, June 1993.

[156] C. N. Potts and L. N. van Wassenhove. A branch and bound algorithm for the total weighted tardiness problem. *Operations Research*, 33(2):363–377, 1985.

[157] C. N. Potts and L. N. Van Wassenhove. Single machine tardiness sequencing heuristics. *IIE Transactions*, 23(4):346–354, December 1991.

[158] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1992. Second Edition.

[159] S. Prestwich. Local search and backtracking vs non-systematic backtracking. In *Using Uncertainty Within Computation: Papers from the 2001 AAAI Fall Symposium, Technical Report FS-01-04*, pages 109–115. AAAI Press, 2-4 November 2001.

[160] R. V. Rachamadugu and T. E. Morton. Myopic heuristics for the single machine weighted tardiness problem. Working Paper 30-82-83, GSIA, Carnegie Mellon University, Pittsburgh, PA, 1982.

[161] N. Raman, R. V. Rachamadugu, and F. B. Talbot. Real time scheduling of an automated manufacturing center. *European Journal of Operational Research*, 40:222–242, 1989.

[162] P. Ross, S. Schulenburg, J. G. Marín-Blázquez, and E. Hart. Hyper-heuristics: Learning to combine simple heuristics in bin-packing problem. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 942–948. Morgan Kaufmann, 2002.

[163] O. Rossi-Doria and B. Paechter. An hyperheuristic approach to course timetabling problem using an evolutionary algorithm. In *The 1st Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2003)*, August 2003.

[164] Y. Ruan, E. Horvitz, and H. Kautz. Restart policies with dependence among runs: A dynamic programming approach. In P. Van Hentenryck, editor, *Principles and Practice of Constraint Programming – CP 2002: 8th International Conference, Proceedings*, volume LNCS 2470 of *Lecture Notes in Computer Science*, pages 573–586. Springer-Verlag, 2002.

[165] W. Ruml. Incomplete tree search using adaptive probing. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 235–241, 4-10 August 2001.

[166] W. Ruml. Using prior knowledge with adaptive probing. In *Using Uncertainty Within Computation: Papers from the 2001 AAAI Fall Symposium, Technical Report FS-01-04*, pages 116–120. AAAI Press, 2-4 November 2001.

[167] W. Ruml. *Adaptive Tree Search*. PhD thesis, Harvard University, May 2002.

[168] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, New Jersey 07458, 1995.

[169] S. Sachdev. *Explorations in Asynchronous Teams*. PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, 3 December 1998.

[170] N. M. Sadeh, Y. Nakakuki, and S. R. Thangiah. Learning to recognize (un)promising simulated annealing runs: Efficient search procedures for job shop scheduling and vehicle routing. *Annals of Operations Research*, 75:189–208, 1997.

[171] O. Scaillet. Density estimation using inverse and reciprocal inverse gaussian kernels. Technical Report IRES DP 2001-17, Université Catholique De Louvain, 2001.

[172] J. D. Schaffer, R. A. Caruana, L. J. Eshelman, and R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, CA, 1989. Morgan Kaufmann.

[173] R. Schoonderwoerd, O. Holland, and J. Bruten. Ant-like agents for load balancing in telecommunications networks. In *Agents '97, Proceedings of the First International Conference on Autonomous Agents*, pages 209–216. ACM Press, 1997.

[174] R. Schoonderwoerd, O. Holland, J. Bruten, and L. Rothkrantz. Ant-based load balancing in telecommunications networks. *Adaptive Behavior*, 5(2):169–207, 1997.

[175] C. Schwindt. ProGen/max: A new problem generator for different resource-constrained project scheduling problems with minimal and maximal time lags. Technical Report WIOR-449, Universität Karlsruhe, Karlsruhe, Germany, July 1995.

[176] C. Schwindt. Generation of resource-constrained project scheduling problems with minimal and maximal time lags. Technical Report WIOR-489, Universität Karlsruhe, Karlsruhe, Germany, November 1996.

[177] C. Schwindt. A branch-and-bound algorithm for the resource-constrained project duration problem subject to temporal constraints. Technical Report WIOR-544, Universität Karlsruhe, Karlsruhe, Germany, 1998.

[178] C. Schwindt. Generation of resource-constrained project scheduling problems subject to temporal constraints. Technical Report WIOR-543, Universität Karlsruhe, Karlsruhe, Germany, November 1998.

[179] C. Schwindt. Project generator progen/max and psp/max-library, 2003. http://www.wior.uni-karlsruhe.de/LS_Neumann/Forschung/ProGenMax/.

[180] B. Selman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In D. S. Johnson and M. A. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, October 11-13, 1993*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1996.

[181] A. K. Sen and A. Bagchi. Graph search methods for non-order-preserving evaluation functions: Applications to job sequencing problems. *Artificial Intelligence*, 86(1):43–73, September 1996.

[182] L. Shi and S. Ólafsson. Nested partitions method for global optimization. *Operations Research*, 48(3):390–407, May-June 2000.

[183] B. W. Silverman. Algorithm AS 176: Kernel density estimation using the fast fourier transform. *Applied Statistics*, 31(1):93–99, 1982.

[184] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Monographs on Statistics and Applied Probability. Chapman and Hall, 1986.

[185] J. Singer, I. P. Gent, and A. Smaill. Backbone fragility and the local search cost peak. *Journal of Artificial Intelligence Research*, 12:235–270, 2000.

[186] S. F. Smith, D. W. Hildum, and D. Crimm. Toward the design of web-based planning and scheduling services. In *Proceedings of the ECP-01 / Planet Workshop on Automated Planning and Scheduling Technologies in New Methods for Electronic, Mobile and Collaborative Work*, September 2001.

[187] S. F. Smith, D. W. Hildum, and D. A. Crimm. Interactive resource management in the COMIREM planner. In *Proceedings of the IJCAI-03 Workshop on Mixed-Initiative Intelligent Systems*, August 2003.

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

[189] T. Stützle. An ant approach to the flow shop problem. In *Proceedings of the* $6th$ *European Congress on Intelligent Techniques & Soft Computing (EUFIT'98)*, volume 3, pages 1560–1564. Verlag Mainz, Wissenschaftsverlag, Aachen, 1998.

[190] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[191] S. Talukdar, L. Baerentzen, A. Gove, and P. de Souza. Asynchronous teams: Cooperation schemes for autonomous agents. *Journal of Heuristics*, 4:295–321, 1998.

[192] S. N. Talukdar. Asynchronous teams. Technical Report EDRC 18-42-93, Engineering Design Research Center, Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, 1993.

[193] R. A. Tapia and J. R. Thompson. *Nonparametric Probability Density Estimation*. The Johns Hopkins University Press, Baltimore and London, 1978.

[194] G. Theraulaz, E. Bonabeau, and J. L. Deneubourg. Self-organization of hierarchies in animal societies: The case of the primitively eusocial wasp polistes dominulus christ. *Journal of Theoretical Biology*, 174:313–323, 1995.

[195] G. Theraulaz, E. Bonabeau, and J. L. Deneubourg. Response threshold reinforcement and division of labour in insect societies. *Proceedings of the Royal Society of London B*, 265(1393):327–335, February 1998.

[196] G. Theraulaz, S. Goss, J. Gervet, and J. L. Deneubourg. Task differentiation in polistes wasp colonies: A model for self-organizing groups of robots. In *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 346–355. MIT Press, 1991.

[197] S. van der Zwaan and C. Marques. Ant colony optimisation for job shop scheduling. In *Proceedings of the '99 Workshop on Genetic Algorithms and Artficial Life GAAL'99*, Lisbon, Portugal, March 1999.

[198] P. J. M. van Laarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Applications*. D. Reidel Publishing Company (Kluwer Academic Publishers), 1987.

[199] T. Walsh. Depth-bounded discrepency search. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 1388–1395. Morgan Kaufmann, 1997.

[200] L. Wasserman. Lecture notes: 36-713. nonparametric methods. http://www.stat.cmu.edu/˜larry/=stat713/, 2001.

[201] J. P. Watson, L. Barbulescu, A. E. Howe, and L. D. Whitley. Algorithm performance and problem structure for flow-shop scheduling. In *Proceedings, Sixteenth National Conference on Artificial Intelligence (AAAI-99), Eleventh Innovative Applications of Artificial Intelligence Conference (IAAI-99)*, pages 688–695. AAAI Press, 1999.

[202] J. P. Watson, J. C. Beck, A. E. Howe, and L. D. Whitley. Toward an understanding of local search cost in job-shop scheduling. In *Proceedings of the Sixth European Conference on Planning*, 2001.

[203] J. P. Watson, J. C. Beck, A. E. Howe, and L. D. Whitley. Problem difficulty for tabu search in job-shop scheduling. *Artificial Intelligence*, 143(2), February 2003.

[204] D. F. Wong, H. W. Leong, and C. L. Liu. *Simulated Annealing for VLSI Design*. Kluwer Academic, 1988.

[205] S. J. Wu and P. T. Chow. Genetic algorithms for nonlinear mixed discrete-integer optimization problems via meta-genetic parameter optimization. *Engineering Optimization*, 24(2):137–159, 1995.

[206] X. Yao. Call routing by simulated annealing. *International Journal of Electronics*, 79(4):379–387, 1995.

[207] S. Zilberstein. *Operational Rationality through Compilation of Anytime Algorithms*. PhD thesis, University of California at Berkeley, 1993.

[208] S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, Fall 1996.

[209] S. Zilberstein and S. Russell. Optimal composition of real-time systems. *Artificial Intelligence*, 82:181–213, 1996.

Appendix A

Weighted Tardiness Scheduling with Sequence-Dependent Setups: A Benchmark Set

A.1 Overview

This Appendix details the set of problem instances used in the experiments of Chapter 5. These problem instances are available online at either of:

- http://www.cs.cmu.edu/~vincent/benchmarks.html
- http://www.ozone.ri.cmu.edu/benchmarks.html

The file format for a problem instance file is described in Section A.2. The current best known solutions to these instances which were found by various algorithms discussed in Chapter 5 are listed in Section A.3.

A.2 Instance File Format

Each instance of the benchmark library is stored in a separate file according to the following file format:

```
Problem Instance: <instance number>
Problem Size: <number of jobs in instance>
```

```
Begin Generator Parameters
Tau: <tau>
R: <R>
Eta: <eta>
P_bar: <average process time>
P_MIN: <minimum process time>
P_MAX: <maximum process time>
S_bar: <average setup time>
MAX_WEIGHT: <maximum weight value>
C_max: <target makespan>
D_bar: <average duedate>
End Generator Parameters
Begin Problem Specification
Process Times:
<process time for job 0>
...
<process time for job n>
Weights:
<weight for job 0>
...
<weight for job n>
Duedates:
<duedate for job 0>
...
<duedate for job n>
Setup Times:
<job i> <job j> <setup time for j if it follows i>
// i=-1 indicates the setup time if j is first job
End Problem Specification
```

A.3 Best Known Solutions

In this list of best known solutions, we list the problem instance number, best found objective value for the instance, and the algorithm that found it. Algorithms are referred to as

follows:

- LDS-all-two: This is LDS (Limited Discrepancy Search) allowed to run long enough to consider all two-discrepancy solutions.

- HBSS,<iterations>,<bias>: This is HBSS with the specified number of iterations and a bias function equal to $\text{rank}^{-1*\text{bias}}$.

- VBSS,<iterations>,<bias>: This is VBSS with the specified number of iterations and a bias function equal to $\text{value}^{\text{bias}}$.

- VBSS-HC,<iterations>,<bias>: This is the random-restart hill-climber using VBSS to seed the starting solutions for the specified number of iterations and a bias function equal to $\text{value}^{\text{bias}}$.

- SA,<totalEvals>: This is simulated annealing with a modified Lam schedule for the specified number of total evaluations. Search begins with ATCS solution.

These algorithms are described in detail in Chapter 5 of this thesis. Other algorithms considered (including DDS, other variations of LDS, IS, Lee *et al.*'s single-start hill-climber, the deterministic ATCS policy) also found some of these best known solutions but did not exclusively find any best known solutions. The best known solutions are as follows:

Problem Instance	Objective Value	Algorithm
1	978	VBSS-HC,10000,5
2	6489	VBSS-HC,10000,5
3	2348	VBSS-HC,5000,5
4	8311	SA,20000000
5	5606	VBSS-HC,1000,5
6	8244	VBSS,5000,5
7	4347	VBSS-HC,10000,5
8	327	VBSS-HC,5000,5
9	7598	LDS-all-two
10	2451	VBSS-HC,10000,5
11	5263	VBSS-HC,2500,5
12	0	LDS-all-two
13	6147	VBSS-HC,2500,5
14	3941	VBSS,2500,5
15	2915	VBSS-HC,5000,5
16	6711	VBSS-HC,10000,5
17	462	VBSS-HC,5000,5
18	2514	VBSS,10000,5
19	279	VBSS-HC,2500,5
20	4193	VBSS-HC,10000,5
21	0	LDS-all-two
22	0	LDS-all-two
23	0	LDS-all-two
24	1791	SA,20000000
25	0	SA,20000000
26	0	LDS-all-two
27	229	SA,20000000
28	72	SA,20000000
29	0	LDS-all-two
30	575	SA,20000000

Problem Instance	Objective Value	Algorithm
31	0	LDS-all-two
32	0	LDS-all-two
33	0	LDS-all-two
34	0	LDS-all-two
35	0	LDS-all-two
36	0	LDS-all-two
37	2407	VBSS-HC,10000,5
38	0	LDS-all-two
39	0	LDS-all-two
40	0	LDS-all-two
41	73176	SA,20000000
42	61859	VBSS-HC,2500,5
43	149990	LDS-all-two
44	38726	SA,20000000
45	62760	VBSS,100,9
46	37992	SA,20000000
47	77189	SA,20000000
48	68920	LDS-all-two
49	84143	SA,20000000
50	36235	SA,20000000
51	58574	VBSS-HC,5000,5
52	105367	VBSS-HC,5000,5
53	95452	VBSS-HC,10000,5
54	123558	VBSS,2500,5
55	76368	VBSS,5000,5
56	88420	SA,20000000
57	70414	VBSS,10000,5
58	55522	VBSS-HC,10000,5
59	59060	VBSS-HC,10000,5
60	73328	VBSS-HC,10000,5

Problem Instance	Objective Value	Algorithm
61	79884	SA,20000000
62	47860	SA,20000000
63	78822	VBSS,200,23
64	96378	SA,20000000
65	134881	SA,20000000
66	64054	SA,20000000
67	34899	SA,20000000
68	26404	SA,20000000
69	75414	SA,20000000
70	81200	SA,20000000
71	161233	VBSS-HC,1000,5
72	56934	VBSS-HC,5000,5
73	36465	LDS-all-two
74	38292	VBSS-HC,10000,5
75	30980	LDS-all-two
76	67553	VBSS-HC,5000,5
77	40558	SA,20000000
78	25105	SA,20000000
79	125824	VBSS,5000,5
80	31844	VBSS,10000,5
81	387148	VBSS,200,17
82	413488	VBSS,200,17
83	466070	SA,20000000
84	331659	VBSS,100,20
85	558556	SA,20000000
86	365783	VBSS,200,20
87	403016	SA,20000000
88	436855	VBSS,200,11
89	416916	VBSS-HC,5000,5
90	406939	LDS-all-two

Problem Instance	Objective Value	Algorithm
91	347175	VBSS,100,14
92	365779	VBSS,100,11
93	410462	VBSS,100,19
94	336299	VBSS,100,10
95	527909	VBSS,100,9
96	464403	LDS-all-two
97	420287	VBSS,10000,5
98	532519	VBSS,100,12
99	374781	VBSS,100,12
100	441888	VBSS-HC,500,5
101	355822	LDS-all-two
102	496131	LDS-all-two
103	380170	VBSS,100,18
104	362008	VBSS,200,15
105	456364	VBSS,200,17
106	459925	LDS-all-two
107	356645	VBSS,200,15
108	468111	VBSS,200,21
109	415817	VBSS,10,13
110	421282	LDS-all-two
111	350723	VBSS,5000,5
112	377418	VBSS-HC,10000,5
113	263200	VBSS,100,5
114	473197	VBSS-HC,5000,5
115	460225	VBSS,100,10
116	540231	LDS-all-two
117	518579	VBSS-HC,10000,5
118	357575	LDS-all-two
119	583947	VBSS,200,13
120	399700	VBSS-HC,10000,5