1 Testing, Evaluation and Performance of Optimization and Learning Systems

D. Whitley, J.P. Watson, A. Howe, and L. Barbulescu

Colorado State University, Fort Collins CO 80523, USA

Abstract. Benchmarks and test suites are widely used to evaluate optimization and learning systems. The advantage is that these test problems provide an objective means of comparing systems. The potential disadvantage is that systems can become overfitted to work well on benchmarks and therefore that good performance on benchmarks does not generalize to real world problems. The meaning and significance of benchmarks is examined in light of theoretical results such as "No Free Lunch." The "structure" of common benchmarks is also explored.

1.1 Introduction and Motivation

Benchmarks are commonly used for testing both optimization and learning algorithms. Often, the legitimacy of a new algorithm is "established" by demonstrating that it finds better solutions than existing algorithms when evaluated on a particular benchmark or collection of benchmarks. Alternatively, the new algorithm may find high-quality solutions faster than existing algorithms for one or more benchmarks.

There are also dangers associated with the use of benchmarks. Algorithms can be overfitted such that they perform well on specific benchmarks, but fail to exhibit good performance on benchmarks with different characteristics. More importantly, there is no guarantee that algorithms developed and evaluated using synthetic benchmarks will perform well on more realistic problem instances [16]. Furthermore, simple algorithms can often provide excellent performance on realistic benchmarks [17].

While the dangers associated with benchmarks are well-known, most researchers continue to use benchmarks to evaluate their algorithms. This is because researchers have few alternatives. How can one algorithm be compared to another without some form of evaluation? Evaluation requires the use of either synthetic or real-world benchmarks, or at least the use of test problems drawn from problem generators so that algorithms can be compared on sets of problem instances that have similar characteristics. Researchers who develop new algorithms and do not demonstrate their merit through some form of comparative testing can expect their work to be ignored. The compulsion to develop "a new method" has resulted in the literature being full of new algorithms, most of which are never used or analyzed by anyone other than the researchers who created them.

Hooker [6] discusses the "evils of competitive testing", and points out the difficulty of making fair comparisons of algorithm performance. Implementation details can significantly impact algorithm performance, as can the values selected for various tuning parameters. Some algorithms have been refined for years. Other algorithms have become so specialized that they only work well on specific benchmarks. Hooker argues that the evaluation of algorithms should be performed in a more scientific, hypothesis-driven manner. Barr et al. [1] suggest guidelines for the experimental evaluation of heuristic methods. Such guidelines are for the most part useful, although rarely followed.

While evaluation is difficult, it is also important. Too many experimental papers (especially conference papers) include no comparative evaluation; researchers may present a hard problem (perhaps newly minted) and then present an algorithm to solve the problem. The question as to whether some other algorithm could have done just as well (or better!) is ignored.

At the same time, recent theoretical results make it difficult to know exactly what it means when we compare one algorithm against another. "No Free Lunch" (NFL) results for search indicate that no algorithm is better than another when compared over all possible problems–or even sets of problems that form a "permutation closure." Similar theoretical results exist for machine learning problems. This paper doesn't offer a solution to the problem of evaluation and testing. It does attempt to put the problem in sharper focus. We first review No Free Lunch, and discuss when it does and does not hold. We then discuss the difficulty of using random test problems–even for problems where No Free Lunch does not hold.

1.2 No Free Lunch for Optimization and Learning

The "No Free Lunch" (NFL) theorem for search [20] [21] proves that no algorithm is better than another over all possible instances of optimization problems; an analogous result holds for all instances of learning problems.

Each instance of an optimization problem has an associated objective function. If we let Λ denote the set of all functions and β a particular set of benchmark functions, we can then define $\Lambda - \beta$ as all functions not in the set of benchmarks. NFL implies that if algorithm **K** is better (on average) than algorithm **Z** on the benchmark set β, then algorithm **Z** must be better (on average) than **K** on the instances in $\Lambda - \beta$. NFL theorems make it clear that comparative evaluation is really a zero-sum game; any winnings on β are balanced by losses on $\Lambda - \beta$.

So what does it mean to evaluate an algorithm on a set of benchmarks and compare it to another algorithm? Given the NFL theorems, comparison is meaningless unless we prove (which virtually never happens) or assume (an assumption which is rarely made explicit) that the benchmarks used in a comparison are somehow representative of a particular sub-class of problems.

Schumacher et al. [13] and Whitley [19] present work relating the NFL theorem to the **permutation closure** of a set of functions. Let $\mathcal{X}$ and $\mathcal{Y}$ denote finite sets and let f: $\mathcal{X} \longrightarrow \mathcal{Y}$ be a function where $f(x_i) = y_i$. Let σ be a permutation such that $\sigma : \mathcal{X} \longrightarrow \mathcal{X}$. We can permute functions as follows:

$$\sigma f(x) = f(\sigma^{-1}(x))$$

Since $f(x_i) = y_i$, the permutation $\sigma f(x)$ can also be viewed as a permutation over the values that make up the codomain (the output values) of the objective function.

We can now define the permutation closure $P(F)$ of a set of functions F.

$$P(F) = \{\sigma f : f \in F \text{ and } \sigma \text{ is a permutation}\}$$

Note that the permutation closure has the following property.

$$P(F \cup F') = P(F) \cup P(F')$$

Proofs are given by Schumacher et al. [13]. This means we can now make a more precise statement about the "zero-sum" nature of No Free Lunch (NFL). If algorithm **K** outperforms algorithm **Z** on any subset of functions denoted by β, then algorithm **Z** will outperform algorithm **K** over $P(\beta) - \beta$. This means that No Free Lunch results for search apply to finite sets. These sets can in fact be quite small.

Consider the following needle-in-a-haystack function. In this case NFL for search holds over 4 functions. All needle-in-a-haystack functions have a compact representation of size O(lg N), where $N = |\mathcal{X}|$.

$$\begin{aligned} f &= \langle 0,0,0,1\rangle \implies \\ P(f) &= \{\langle 0,0,0,1\rangle, \langle 0,0,1,0\rangle, \langle 0,1,0,0\rangle, \langle 1,0,0,0\rangle\} \end{aligned}$$

It is useful to view the permutation closure of a function as a table, where each row of the table is a function/permutation. Each row in the table also corresponds to the behavior of some optimization algorithm on some function. The *behavior* of an optimization algorithm with respect to some objective function describes the order in which the optimization algorithm samples the values that make up the codomain of the objective function. Schumacher et al. [13] refer to this as the "performance vector". Thus

```
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
```

could represents the permutation closure over a needle-in-a-haystack function for which the NFL theorems for search hold; each row also describes the behavior of some search algorithm applied to some function in the associated permutation closure.

The "table representation" makes it clear when NFL results hold and makes is clear that making a general declaration that one algorithm is better than another is in some sense meaningless. An algorithm can be better than another only on a subset of problems. It is also clear that there is no such thing as a generally robust algorithm [13].

Furthermore, consider the following table representing the permutation closure over a function defined over a codomain of 3 values.

1	2	3
1	3	2
2	1	3
2	3	1
3	1	2
3	2	1

Each column of the table represents the set of possible behaviors at a particular time step; the rows represent all possible performance vectors. But each column is identical in its composition. The notion of robustness implies that some algorithm yields relatively good performance over a broad range of problems compared to another algorithm. This would suggest that relatively good solutions are found within some fixed (e.g. polynomial) number of time steps. Yet, if NFL holds over a set of problems, the set of codomain values returned over all functions in the permutation closure is identical at each time step.

Culberson [4] points out that NFL theorems for optimization break down when we have knowledge about a problem. For example, he notes that NFL theorems do not hold over the class NP. No Free Lunch implies that no algorithm is better than another, and random enumeration is as good as any other search algorithm–which implies that no method is better than (exponential time) enumeration over all problems. But since we do not know if $NP = P$ we cannot know if NFL holds over the set of problems in NP.

There is also other evidence that NFL does not hold for specific problems in the set NP. The existence of ratio bounds for certain NP-Complete problems shows that NFL theorems do not hold for specific NP-Complete problems. For example, a greedy polynomial time algorithm exists for the Euclidean Traveling Salesman Problem which is guaranteed to yield a solution that is no worse than $2C$ where C is the cost of an optimal solution [3]. Branch and bound algorithms can use this information to compute bounds such that no solution with a cost greater than $2C$ is examined [7]. Thus, the existence of a ratio bound means that algorithms can select which performance vectors to explore, and this excludes some search behaviors (i.e., performance vectors) which are part of the permutation closure of the objective function.

Many researchers have dismissed NFL using the argument that NFL does not hold over real world problems. At the same time, researchers generally do not offer proofs that their algorithms are designed for particular subclasses of problems. Thus, we are left in the awkward situation where we need test problems and benchmarks to compare one algorithm to another, but we do so with the knowledge that when algorithm **K** outperforms algorithm **Z**, algorithm **Z** is better than **K** on a complementary finite set of problems. If we insist that some fixed percentage of values that make up the codomain are unique, then that complementary set will be composed of problems whose average description length is exponential in size [19] [13], but that still does not mean that algorithm **Z** is not better than algorithm **K** on some small subset of problems with a compact representation.

At least from a theoretical point of view, comparative evaluation is a dangerous, if not dubious, enterprise. But, the alternative of testing is to give up and say that all algorithms are equal–which means we have no way of recommending one algorithm over another when a search method is required to solve a problem of practical interest. The best we can do is build test functions that we believe capture some aspects of the problems we actually want to solve. But this highlights the critical question. Do benchmarks really test what we want to test? If an algorithm does well on a very simple problem–such as a linear objective function–is that good or bad? Many people have used the ONEMAX test function for testing search algorithms that use a binary representation. The objective function for ONEMAX is to count the number of bits set to **1** with the goal of maximization. But should we really believe that an algorithm that does well on ONEMAX really generalizes to other problems of practical interest? Theory would suggest caution, but provides no real answers.

1.2.1 No Free Lunch, Generalization, and Learning

The topic of over generalization has been been examined much more extensively in the learning literature than in the literature on optimization. As with optimization algorithms, we typically assume that our learning methods work well across a large number of problem domains of practical interest.

No Free Lunch and learning is briefly discussed here to highlight differences between learning and optimization. While the *permutation closure* is critical to the NFL results for optimization, for Boolean classification learning the *power set* is critical.

This is illustrated by the simple example shown in figure 1.1. The task is to assign a **0** or **1** classification to every square in a 2 dimensional grid. The cases where a **0** or **1** already appears in the grid can be taken as training data. The learner must learn to classify the remaining squares. A concise hypothesis might assume that all squares in the top half of the grid should be classified as **1** and all squares in the bottom half of the gird should be **0**. Algorithms such as a neural network or a decision trees produced by ID3 (or C4.5)

1	1	1	1
1			1
0			0
0	0	0	0

Fig. 1.1. A simple Boolean classification task.

would almost certainly arrive at this solution because a simple linear decision boundary suffices to classify all the training data. However, in the *space of all possible problems* there are 16 possible classifications corresponding to the set of all 4-bit binary strings. Viewed another way, the possibilities correspond to the power set when we ask which subset of the unclassified squares should be classified in class **1**.

This small example illustrates two ideas. First, NFL results for learning are somewhat different in detail from the NFL results for optimization–and they are also alike in that all possible classifications (e.g. the power set for Boolean classification problems) and all possible search patterns (the permutation closure for optimization problems) are considered.

The second idea that relates to both optimization and learning is that every optimization algorithm and every learner has a bias. Algorithms that perform random search, or learners that output random classifications, have a random uniform bias–which is typically not what we want. To the degree that an algorithm makes a non-random decision, the algorithm operates on the assumption that there is structure in the problem that can be exploited. The notion of bias has been widely discussed in the machine learning literature, but has received little attention in the search and optimization literature. But even in the machine learning literature, the notion of bias is often treated in very general terms. The bias of an ID3 decision tree is described by Mitchell ([10], p. 63) as follows:

> Shorter trees are preferred over longer trees. Trees that place high information gain attributes close to the root are preferred over those that do not.

Similar descriptions of bias can be constructed for optimization algorithms such as Tabu search or Genetic Algorithms. Tabu search, as a rough approximation, assumes that good solutions are reached by finding and then escaping from optima to explore other nearby local optima. Genetic algorithms, as a rough approximation, assume that solutions tend to be quasi-decomposable into building blocks and that the decomposition and reconstitution of good solutions via recombination leads to better solutions. (Note the bias of mutation is a bit different: good solution tend to be near other good solutions in the representation space in which the mutation operates.)

But it is difficult to say whether the bias of a neural networks or the bias of a decision tree is a better reflection of the structure which we want to model in the world. And it is difficult to say whether the bias of Tabu search or the bias of a genetic algorithm is a better reflection of the type of structure encountered in search problems.

1.3 Testing Scheduling Algorithms

In this section, we return to the topic of optimization. Can we escape the trap imposed by No Free Lunch? Comparing over all possible problems is meaningless, but what problems should we be using? It would appear that NP-Complete problems such as scheduling would be a safer domain for comparative evaluation. Proofs can be derived for certain NP-Complete problems where we can escape the NFL dilemma. However, even within the set of NP-Complete problems there are questions concerning the use of random test problems and structured test problems. Is the use of "difficult" randomly generated test problems reasonable?

SAT problems are a commonly used testbed for search algorithms [5]. These problems are logical expressions, often expressed in conjunctive normal form (CNF) with k variables (some of which may be negated) in each disjunctive clause. A problem is *satisfiable* if there exists an assignment of truth values to the variables that will make the logical expression true. When stochastic search algorithms are applied to SAT problems they need a gradient to climb which is not provided by a Boolean function. So rather than using logical ANDing to combine disjunctive clauses, the Boolean evaluations associated with the individual clauses are summed with *true* being 1 and *false* being 0. The maximum is then sought to solve the problem. This evaluation function is called **MAXSAT** [11].

Like MAXSAT test problems, Permutation Flowshop Scheduling Problems are often randomly generated. For MAXSAT, these problems are chosen with a particular clause/variable ratio that empirically results in difficult problems [9] [14]. For permutation flow-shop problems, there appears to be even less motivation for the use of random functions. In both cases, there is a general assumption that algorithms which work well on random problems also work well on other types of problems. We examine this assumption in more detail for permutation flowshop problems.

1.3.1 Random versus Structured PFSPs

In an $n \times$m instance of the Permutation Flow-Shop Scheduling Problem (PFSP), n jobs must be processed in the same order on each of m machines. Concurrency and preemption are not allowed, and each job must be

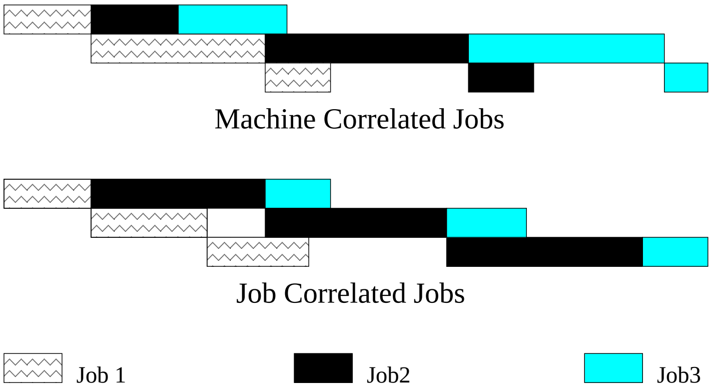

Fig. 1.2. Job correlated and machine correlated processing times are illustrated using three jobs on three machines.

processed on a machine for a pre-specified duration. Solutions to the PFSP are a permutation of the n jobs, and the most commonly considered objective is to minimize the makespan, or the time required to complete all jobs.

Algorithms for the PFSP are typically evaluated using well-known benchmark problem instances in which the job processing times are randomly generated (e.g., via uniform sampling from the interval $[1, 99]$). However, real-world scheduling problems generally possess some form of structure, and it is unclear whether algorithms that perform well on random problems will also perform well on more realistic, structured instances.

Kan [8] introduced two methods of creating structure: job correlation and time gradients. In *job-correlated* problems, the processing times of a given job are 'correlated' in the sense that they are sampled from a relatively tight distribution specific to that job. *Time gradients* impose non-random structure over the job processing times on different machines; processing times on early machines tend to be less than those of later machines. Watson et al. [17] [18] introduce a third method to create non-random structure: *machine correlation*. In *machine-correlated* problems, the processing times of all jobs on a given machine are sampled from a relatively tight distribution specific to that machine. Job correlated and machine correlated problems are illustrated in figure 1.2.

Note that on machine correlated problem instances, the machine with the larger processing times (in this case Machine 2) has the largest impact on the total processing time (or the "makespan" of the schedule). But on the job correlated problem instances the job with the largest processing times (in this case Job 2) has the largest impact on the processing time. Experiments show that the lower bound calculation found in Taillard [15] [16] perform very well for random and machine correlated problems. But Taillard's lower bound calculation breaks down and yields an extremely poor bound on job correlated problems. Kan [8] gives a more complete lower bound calculation

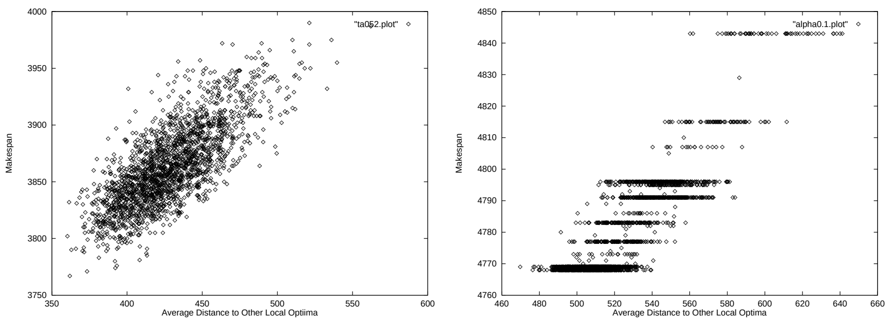

Fig. 1.3. Local optima distribution for a random (left figure) and a structured (right figure) PFSP instance.

that cover both machine correlated and job correlated problems. But this again illustrates how there are biases, not only in the test problems that are used for evaluating algorithms, but also in commonly used lower bound calculations that are in some sense used to measure problem difficulty. (Problem instances with solutions that are far from the lower bound are often assumed to be more difficult.) Here again, commonly used benchmarks introduce an unexpected bias.

1.3.2 Differences in Search Space Topology

Creating structured PFSPs enables experiments that explore changes in the search space topology and the performance of optimization algorithms as one transitions from random to structured problem instances.

It is generally assumed that the structure of the search space dictates algorithm performance. In particular, several researchers have investigated the distribution of local optima, and how local optima topology impacts algorithm performance. Reeves and Yamada [12] show that the local optima of random PFSPs are distributed in a *big-valley* structure [2], in which (1) local optima tend to be relatively close to other local optima, (2) better local optima tend to be closer to global optima, and (3) local optima near one another have similar quality. A prototypical example of a big-valley local optima distribution is shown in the left side of Figure 1.3. Local optima with better evaluations are nearer to each other than local optima with poorer evaluation. This suggests that algorithms which “jump” from optimum to optimum can exploit this structure. A big-valley distribution is therefore well-suited to a number of local search techniques, and many state-of-the-art algorithms for the PFSP are designed to exploit such a distribution [12].

A natural question then arises: Do structured PFSPs possess big-valley local optima distributions? If not, there is little reason to expect that superior performance on random PFSPs will transfer to more structured PFSPs. Watson et al.[17] show that the local optima of structured PFSPs are *not* distributed in a big-valley; rather, local optima are generally members of large plateaus of equally-fit solutions. An example of the local optima distribution in structured PFSPs is shown in the right side of Figure 1.3. Given strong differences in search space topology, we would anticipate differences in the performance of PFSP algorithms on random and structured instances.

1.3.3 Differences in Algorithm Performance

Watson et al. [17] analyze the performance of several PFSPs algorithms along a continuum from highly random to highly structured problem instances. In Figures 1.4 and 1.5, we show the performance of several of these algorithms. In the current figures we identify only the Path Relinking algorithm. (For details on the other specific algorithms, see [17].)

Path relinking is a general search strategy in which the search space is explored by looking for additional optima near two known local optima. During the process of 'linking' or constructing a path between two local optima, the algorithm can check the intervening area for other optima. Path relinking is the basis for the Reeves/Yamada PFSP algorithm [12], which we denote by *pathrelink*.

Informally, the x-axis indicates the degree of randomness present in problem instances; purely random and highly structured instances are found at 0.0 and 1.0, respectively. The average deviation from the best-known solution is indicated by the y-axis. In figure 1.4 job correlated problem are used; performance begins to "level" as problems become more structured; differences in performance are negligible, and simple heuristic algorithms often perform as well as more complex, state-of-the-art algorithms. In figure 1.5 machine correlated problem are used; here path relinking is actually poorer than several other methods as problems become more structured. Clearly there are subsets of problems where path relinking does very well, and other subsets of problems where other algorithms yield better performance.

The point of these experiments in the current context is not that one algorithm is better than another, but rather that changes in relative performance can result from small changes in problem structure.

1.4 Conclusions

The current paper discusses the difficulty of using benchmarks for evaluating competing algorithms, especially for search and optimization problems. The current state of theory provides little guidance. No Free Lunch results prove that testing is a zero-sum game where winning on one subset of problems implies that we must lose on another subset of problems. This makes it difficult to interpret experimental results on general parameter optimization problems. And yet, testing is critical to building real-world applications.

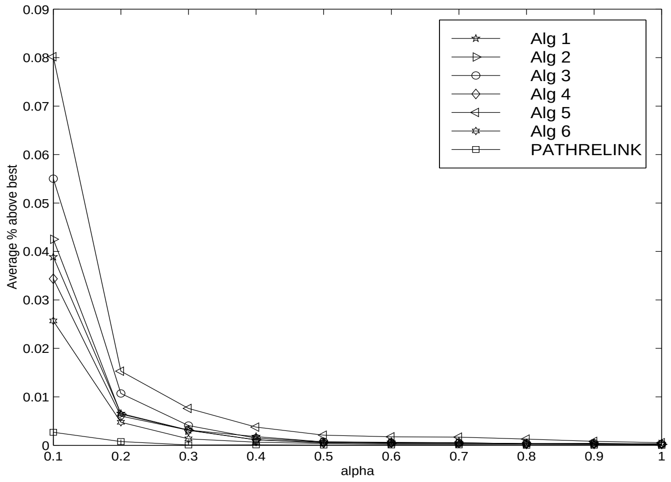

Fig. 1.4. Performance of optimization algorithms for the PFSP on job correlated problems. The degree of randomness is indicated along the x-axis, while the deviation from the best-known solution is indicated along the y-axis. On more random problems, path relinking does extremely well. But as more structure is introduced, all algorithms perform nearly the same.

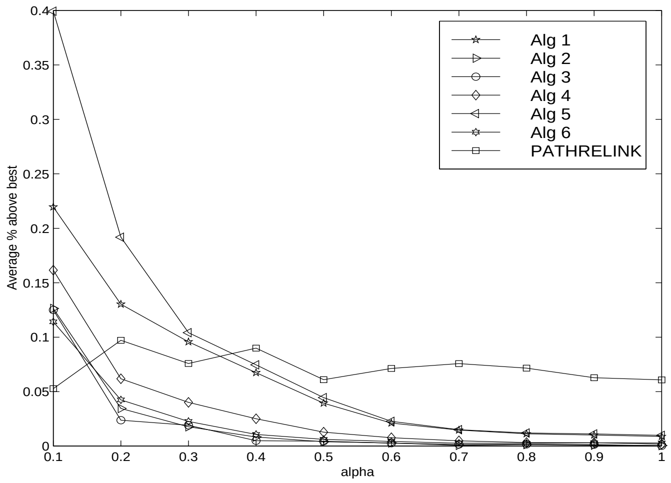

Fig. 1.5. Performance of optimization algorithms for the PFSP on machine correlated problems. The degree of randomness is indicated along the x-axis, while the deviation from the best-known solution is indicated along the y-axis. On more random problems, path relinking again does well. But as more structure is introduced, path relinking is actually poorer than other algorithms.

We can partly escape No Free Lunch by looking at specific types of problems, such as NP-Complete problems. But differences between randomly generated problems and structured problems can also impact algorithm performance for NP-Complete problems.

Theoretical results such as No Free Lunch should motivate researchers to more carefully explore when their algorithms work and when they fail. Ideally, theory will help to explain which methods work well for particular types of problems. But historically, experimental research has often helped to sort out what works and what does not. Researchers need to spend as much time on designing good test problems as they spend creating new search algorithms.

1.5 Acknowledgments

This work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant numbers F49620-97-1-0271 and F49620-00-1-0144. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

References

1. R. Barr, B. Golden, J. Kelly, M Resende, and Jr. W. Stewart. Designing and Reporting on Computational Experiments with Heuristic Methods. *Journal of Heuristics*, 1:9–32, 1995.
2. K. D. Boese, A. B. Kahng, and S. Muddu. A new adaptive multi-start technique for combinatorial global optimizations. *Operations Research Letters*, 16/2:101–113, 1994.
3. T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms.* McGraw Hill, New York, 1990.
4. J. Culberson. On the Futility of Blind Search. *Evolutionary Computation*, 6(2):109–127, 1999.
5. K. DeJong, M. Potter, and Wm. Spears. Using Problem Generators to Explore the Effects of Epistasis. In T. Bäck, editor, *Proc. of the 7th Int'l. Conf. on GAs*, pages 338–345. Morgan Kaufmann, 1998.
6. J.N. Hooker. Testing Heuristics: We Have it All Wrong. *Journal of Heuristics*, 1:33–42, 1995.
7. E. Horowitz and S. Sahni. *Fundamentals of Computer Algorithms.* Computer Science Press, 1978.
8. Alexander R. Kan. *Machine Scheduling Problems: Classification, complexity and computations.* Martinus Nijhoff, The Hague, 1976.
9. David Mitchell, Bart Selman, and Hector Levesque. Hard and easy distribution of sat problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, San Jose, CA, 1992.
10. Tom Mitchell. *Machine Learning.* McGraw Hill, 1997.
11. Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley Publishing, Co., 1994.

12. C.R. Reeves and T. Yamada. Genetic Algorithms, Path Relinking, and the Flowshop Sequencing Problem. *Journal of Evolutionary Computation*, 6(1):45–60, 1998.
13. C. Schumacher, M. Vose, and D. Whitley. The No Free Lunch and Problem Description Length. In *Genetic and Evolutionary Computation Conference GECCO-00*, pages 565–570. Morgan Kaufmann, 2001.
14. B. Selman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In Trick and Johnson, editors, *Second DIMACS Challenge on Cliques, Coloring and Satisfiability*, 1993.
15. E. Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operations Research*, 47:65–74, 1990.
16. E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operations Research*, 64:278–285, 1993.
17. J.P. Watson, L. Barbulescu, D. Whitley, and A. Howe. Artificial test suites for flowshop scheduling and algorithm performance. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, 1999.
18. J.P. Watson, L. Barbulescu, D. Whitley, and A. Howe. Contrasting Structured and Random Permutation Flow-Shop Scheduling Problems. *INFORMS Journal on Computing*, 2002.
19. D. Whitley. Functions as Permutations: Regarding No Free Lunch, Walsh Analysis and Summary Statistics. In Schoenauer, Deb, Rudolph, Lutton, Merelo, and Schwefel, editors, *Parallel Problem Solving from Nature, 6*, pages 169–178. Springer, 2000.
20. David H. Wolpert and William G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute, July 1995.
21. David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 4:67–82, 1997.