The *Necessity* of Meta Bias in Search Algorithms

John R. Woodward
School of Computer Science
The University of Nottingham
Nottingham, UK
Email: john.woodward@nottingham.edu.cn

***Abstract*—Bias is necessary for learning [1], and is a probability over a search space. This is usually introduced implicitly by the designer of a search algorithm, for example by designing a new search operator [6]. This bias is does not change; each time a stochastic search algorithm is executed it will give a different answer. However, if executed repeated it will give the same solution *on average*. In other words, the bias is static (even if we include a self adaptive component to the search algorithm). One desirable property of search algorithms is that they converge (i.e. given enough time they will eventually reach the global optima). In terms of bias, this means that there is a non-zero probability of visiting each item in the search space.**

Search algorithms are intended to be reused on many instances of a problem. These instances can be consider to be drawn from a probability distribution. In other words, a search algorithm and problem class can both be viewed as probability distributions over the search space. If the bias of a search algorithm does not match the bias of a problem class, it will under perform, if however, they do match, it will perform well. Therefore we need some mechanism of altering the initial bias of the search algorithm to coincide with that of the problem class. This mechanism can be realized by a meta level which alters the bias of the base level. In other words, *if a search algorithm is to be applied to many instances of a problem, then meta bias is necessary*. This implies that *convergence at the meta level means a search algorithm shift its bias to any probability distribution*. Additionally, shifting bias is equivalent to automating the design of search algorithms.

KEYWORDS meta, search, machine learning, convergence, no free lunch theorems, optimization, bias, learning to learn.

I. Introduction

Search is the process by which items in a space are sampled in order to find high quality solutions [12] (see figure 1). This is a broad paradigm for machine learning [5]. The vast majority of search algorithms are stochastic, and therefore a probabilistic setting is appropriate. We will use the word algorithm to mean stochastic search algorithm. An algorithm therefore identifies with a probability distribution across the search space, and is often called bias [1]. The vast majority of search algorithms have no way to adjust their bias, and we will argue that this is a flawed approach purely because we intend the algorithm to be reused on multiple problem instances.

Almost all search algorithms are intended for use on many problem instances, and not just on a single instance of a problem [2], [13]. The set of problem instances can be characterized by a probability distribution and corresponds to a probability distribution over a search space. Just as we evolve programs to fit a given data set and make reliable predictions about future data [5], [6], we would be wise to adopt the same attitude to problem instances belonging to a problem class and being drawn from an underlying target distribution.

If an algorithm has a bias which does not match that of the problem class (i.e. the probability distribution over the search space), it will under perform (see figure 3), and will need to adjust its bias to that of the problem class (see figure 4). The central issue is that many search algorithms have a static bias over a number of problem instances. That is, an algorithm does not change its bias from one problem instance to the next. Even self adaptive search algorithms have a static bias across problem instances (see section II-A). If we intend our algorithms to be used on a number of problem instances, then it is imperative that a meta level exists which can shift the bias to the problem class (see figure 2).

The three contributions of this paper are;

1. If we apply our search algorithms to more than a single instance of a problem, then meta bias is necessary so the bias of the search algorithm converges towards the global optima. Automatically altering bias at the meta level is equivalent to automatically designing search algorithms.

2. At the meta level, convergence means that the bias can shift to any bias (i.e. probability distribution).

3. Problem classes defined as probability distributions are *an essential part of the machine learning methodology*. A problem class defines a niche in which a suitable search algorithm can fit. It does not make sense to consider an algorithm in the absence of a problem class which gives it context. Therefore algorithms should be tested on *problem instances which are drawn from this distribution* and NOT *randomly selected benchmark instances from the literature*.

II. Preliminaries

A *space* or *search space* is a set of items, which could be numbers in the domain of a function (in the case of function optimization), permutations of solutions to a combinatorial problem (i.e. routes to a traveling salesman problem) [12], or programs in the case of genetic programming [12]. It may also be called a "hypothesis space" in some areas of machine learning [5]. The items in a search space are also called (potential or candidate) solutions, points, members or individuals. One of the main issues with search is to decide how to sample it. The process of searching a space is equivalent to the "generate and test" paradigm. The definition of *learning* [5] is closely associated with the search process.

Mitchell [1] defines bias as *"any basis for choosing one generalization over another, other than strict consistency with the observed training instances"*. Bias is a probability distribution over the items in the search space and is just a sampling bias in the statistical sense. An unbiased search would just uniformly sample the space at random. Two syntactically different algorithms that yield the same probability distribution over the search space will behave indistinguishably.

We often compare algorithms in terms of the number of evaluations of items in the search space. Bias is a reasonable way to compare search algorithms. as the time taken to generate a new candidate solution takes a fraction of the time that it does to test it. This is a realistic assumption for the majority of search algorithms and real-world problems thought we acknowledge the fact that a computation complexity setting is more robust. Thus, comparing search algorithms as probability distributions over a search space is reasonable. Woodward [8] has showed that search and bias are intrinsically linked.

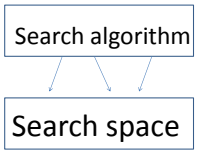

Fig. 1. A search algorithm conducts a sampling over a search space.

1) What are the sources of bias in a search algorithm?: Genetic programming [6] uses of different search operators such as crossover to affect the bias. It is important to report these parameters for the purpose of replication as it is the bias which we are trying to replicate. In short, every parameter or choice that can be made about a search algorithm will affect its bias. Any parameter which does not affect the bias of an algorithm is redundant. Some parameters will have greater influence than others on the bias of the algorithm. For example, the temperature in simulated annealing algorithms [12].

2) An Illustration of Bias: Langdon [7] generated all of the programs up to a given size. Programs are build up from different sets of logical functions. He plots histograms showing the numbers of programs that represent different functions. The first point to note from the figures is that the distribution of functions is far from uniform, with some functions being represented much more frequently than others. Secondly, different choices of function set, for example {NAND} and {AND, OR, NAND, NOR}, highly influences the distribution over the set of logical functions. Of course the choice of function set in genetic programming is just one component of how bias can be affected, but this paper illustrates this very well.

3) Meta Bias: An algorithm has a bias over a search space. We can call this the *base bias* (see figure 1). If the algorithm is presented with different instances of a problem class, it will on average produce the same performance if presented the same set of instances. as the base bias does not change. While an algorithm is demonstrating learning during its execution on a single problem instance, it is not demonstrating learning about the problem class This is important, as we typically do not care about the performance of an algorithm on a single instance, but its performance on a class of problem. This is the situation with most search algorithms in the literature today.

The notable exception to this is algorithms that "learn how to learn" [2]. An algorithm needs some method of altering its base bias as it is presented with sets of instances. The major claim of this paper is that, as the vast majority of algorithms are intended for use on a collection of problem instances, that meta bias is therefore necessary. Evidence for the fact that algorithms are intended to be reused is that the proponents of newly proposed algorithm test them on a number of problem instances which are reported in their papers. *Meta bias* is any mechanism which can adjust the base bias (see figure 2 i.e. it is a probability distribution over a set of base biases).

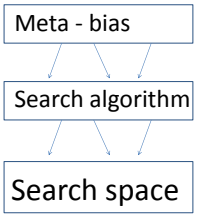

Fig. 2. Meta bias can alter the bias of a search algorithm.

A. Self Adaptation is not Meta bias

Self adaption is a technique used in a number of algorithms, where the algorithm "takes on the responsibility" of altering itself during the search process. This has the advance that the algorithm has the potential of recovering from a poor initial choice of parameter setting, but also the drawback that it may alter a good initial choice for a poorer value. This trade off is a consequence of the NFL theorem [4].

Self adaptation does not provide meta bias. When the same algorithm is executed again it will do the same thing on average as there is no mechanism by which it can remember from one problem instance to the next. While it will change the way it operates during its execution on a single problem instance (in comparison with a none adaptive version of the algorithm), if it is applied to another problem instance, it will not have learn anything from one instance to the next. As it does the same thing, it has by definition, not learnt.

B. Problem Classes

A problem instance is a single example of a problem which could be used for training or testing purposes, and is often called a (test or training) case. A training or test set in machine learning consists of a set of problem instances.

A problem class is a probability distribution over problem instances. A problem class is usually presented to a learning algorithm as representative set of problem instances. We rarely have access to the exact probability distribution, unless we are dealing with synthetically manufactured problem classes. Thus a set of test cases is a sample from the problem class.

Any source of problem instances in the real world will have the potential to generate an infinite sequence of problem instances. In reality, problem classes are probability distributions over a set of problem instances.

C. Property of Convergence

Convergence is the property that a search algorithm will eventually sample the global optima. In terms of bias, this simply means all items in the search space have a non-zero probability. The property of convergence is often achieved in one of two ways (but can be both). Firstly by introducing a "global" search operator which can generate any solution in the search space. For example, with genetic programming, the primary search operator is crossover which shuffles existing genetic material around in the population, while the mutation operator introduces new genetic material into the gene pool, meaning that there is always a non-zero chance that an individual can jump to any point in the search space. Secondly, a "non-greedy" acceptance condition is used instead of one which only allows better solutions to replace the current best. For example, with simulated annealing there is the chance of accepting a lower quality individual according to a "cooling schedule", allowing the current solution to escape from local optima, which its cousin Hill Climbing suffers from [12].

One of the contributions is to say what it means to have convergence at the meta level, and the many search algorithms, while demonstrating the property of convergence at the base level, do not demonstrate it at the meta level.

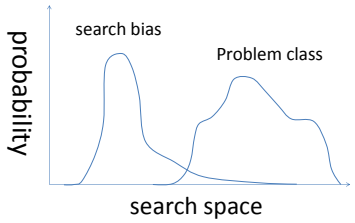

Fig. 3. The bias of the problem class and search algorithm are not aligned. Therefore the search algorithm will perform poorly on this problem class.

III. Discussion and Future Work

A. Meta Learning is Necessary and Sufficient

Rarely is it the case that an algorithm is intend for a single problem. The "one shot scenario" in [13], is quoted here *"It makes no sense to compare optimization techniques in this scenario. One may be lucky and start with some optimal value. Therefore, a problem (from a scientific point of view) has to have a lot of problem instances"*. If an algorithm is intended to tackle more than a single problem, it *must* have meta level. A problem class is equivalent to defining a probability distribution over the set of problems. An algorithm defines a probability distribution over the set of solutions (the bias [2], i.e. each time a stochastic algorithm is executed it produces a different solution, and therefore a probability distribution over the possible solutions). Ideally we want the bias of the algorithm to match the probability distribution of the problem class. For learning to occur at the base level, a learner must improve at that problem (almost all algorithms satisfy this). For a learner to improve over a set of problems, its bias must alter at the meta level. i.e. the probability distribution over solutions produced by the algorithm must converge towards the probability distribution over the problem class (see figure 4). Therefore meta learning is essential if an algorithm is intended to be used on more than a single problem, and most algorithms described in the machine learning literature lack this ability.

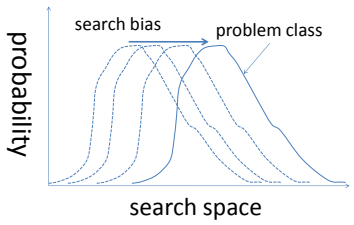

Fig. 4. The bias of the search algorithm converges towards that of the problem class (i.e. they move closer together).

B. Useful Meta Biases

One meta bias which makes sense in machine learning is Occam's Razor [5]. Occam's Razor is the tenancy toward simpler explanations. Interestingly Occam's Razor is mutually exclusive to No Free Lunch [10]. It has been argued that Occam's Razor is a sensible bias from a probabilistic perspective rather than the traditional complexity perspective [9].

Physicists and mathematicians are usually interested in continuous and differentiable functions. Indeed we are usually interested in the class of primitive recursive functions which is a subset of the computable functions [11]. Thus we are not usually interested in "all functions" and this could be a starting point for introducing useful meta biases.

Finally, with some search spaces such as program spaces in genetic programming [6], we have some build-in knowledge. For example, if two programs compute the same function, then we can reduce the probability of sampling one program to zero, as we still have the possibility of sampling the second program with the same functionality.

C. On Line Bin Packing as an Illustration

In this subsection we describe a system which effectively evolves a probability distribution, though not over a search space, will serve as an illustration. It also encapsulates the

idea of problem classes in the approach. In [3], algorithms for the on-line bin packing problem are evolved. The on-line bin packing involves packing a sequence of items as they arrive into the least number of bins. The item sizes are drawn from a probability distribution, and this defines a problem class. Genetic programming is used to evolve a function which decides in which bin to place the current item. The evolved function is evaluated for each bin and the item is placed in the bin which obtained the maximum score. This function can be interpreted as a probability distribution (after adding a constant, to make it non-zero, and multiplying by a scaling factor, to make the sum equal to one), and the placement operation can be interpreted as "place the item in the bin with maximum probability". The point of including this paper here is that the function set used by the genetic programming system is capable of producing any "reasonable" probability distribution (i.e. continuous and differentiable). Thus this paper representant a problem class which is defined by a probably distribution over the items to be packed, and a solution which is effectively a probability distribution function over the bins. Thus, as the genetic programming system has the ability to express any probability distribution, then it is effectively shifting its bias, from some initial bias towards that of the problem class i.e. it is converging towards an optimal probability distribution.

IV. Summary and Conclusion

The bias of a stochastic search algorithm is just a probability distribution over a search space. A problem class defines a probability distribution over the search space too. It is important to consider a problem class rather than a small set of problem instances as we usually intend our algorithm to be executed on a large number of problem instances. This is also important from an application point of view, as most applications are sources of a large set of problem instances. We require the bias of the search algorithm to match the bias of the problem class (see figure 4).

A search algorithm typically has a static bias. If it is run on the same problem instance, it will give different solutions as it is a stochastic algorithm. However, if we run the algorithm many times on a set of problem instances, it will give the same distribution of solutions *on average*. The probability distribution over the search space does not change as the algorithm has no method to alter its bias from one problem instance to the next.

If the bias of an algorithm matches the problem class, we do not need to tailor the bias of the algorithm any further. However, if the bias of our algorithm does not match the bias of the problem class, then the algorithm will perform sub-optimally on that problem class. Hence there is a need to shift the bias of the search algorithm towards that of the problem class (see figure 4). If there is not mechanism for this, then it cannot occur (which is the case with many algorithms). A mechanism is required which will allow the bias of a search algorithm to alter as it is exposed to more and more problem instances. Shifting the bias of the search algorithm, is equivalent to automatically designing search algorithms, as all search algorithms can be expressed as probability distributions.

Convergence at the base level means that there is a non-zero probability of visiting all the items in a search space. There is always a chance of hitting the global optima. At the meta level, we would like our algorithm to have the property that it can shift its bias towards that of *any* probability distribution. That is, the bias of the search algorithm converges to the bias of the problem class.

In many papers concerning search algorithms, the performance is bench marked against a set of isolated, unrelated problem instances. It is not surprising then that they report mixed results such as "our new algorithm performs better on some problem instances, and worse on others". Rarely is it the case that results are obtained across a set of problem instances which universally outperform another algorithm. This points to a flaw in the current methodology of comparing algorithms on a set of isolated problem instances. It is also the case that we intend an algorithm to be used with a problem class (i.e. a specific application), and therefore we should report its performance on a problem class [3].

Meta bias is not new [2], but our central claim is that it is essential because we use our algorithms on many problem instances.

References

[1] T.M. Mitchell, The Need for Biases in Learning Generalizations, Rutgers Computer Science Department Technical Report CBM-TR-117, May, 1980. Reprinted in Readings in Machine Learning, J. Shavlik and T. Dietterich, eds., Morgan Kaufmann, 1990.

[2] S. Thrun and L. Pratt, Learning To Learn, S. Thrun and L. Pratt, ed., Kluwer Academic Publishers, 1998, 354 pages.

[3] E. K. Burke, J. Woodward, M. Hyde, G. Kendall, Automatic heuristic generation with genetic programming: Evolving a Jack of all trades or a master of one. Genetic and Evolutionary Computation Conference, GECCO 2007.

[4] C. Schumacher, M. D. Vose, and L. D. Whitley. The no free lunch and problem description length. In proceedings of the Genetic and Evolutionary Computation Conference, 565-570, California, USA, 7-11 July 2001. Morgan Kaufmann.

[5] T. M. Mitchell. Machine Learning. McGraw-Hill 1997.

[6] Riccardo Poli, William B. Langdon and Nicholas Freitag McPhee, A Field Guide to Genetic Programming, Lulu.com, freely available under Creative Commons Licence from www.gp-field-guide.org.uk, March 2008.

[7] William B. Langdon: Scaling of Program Fitness Spaces. Evolutionary Computation 7(4): 399-428 (1999)

[8] Woodward J. Computable and Incomputable Search Algorithms and Functions. IEEE International Conference on Intelligent Computing and Intelligent Systems (IEEE ICIS 2009) November 20-22,2009 Shanghai, China.

[9] Woodward, J., Evans A., Dempster, P. 2008, A Syntactic Justification of Occam's Razor. October 31 to November 2, 2008 Midwest, A New Kind of Science Conference Indiana University Bloomington, Indiana

[10] Marcus Hutter, "Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability" Springer,2004, http://www.hutter1.net/ai/uaibook.htm.

[11] Hartley Rogers, Theory of Recursive Functions and Effective Computability, The MIT Press (April 22, 1987) ISBN-10: 0262680521

[12] Edmund K.Burke and Graham Kendall (editors), Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, Springer 2005.

[13] Stefan Droste, Thomas Jansen, Ingo Wegener: Perhaps Not a Free Lunch But At Least a Free Appetizer, 13-17 July 1999: 833-839 Proceedings of the Genetic and Evolutionary Computation Conference, Orlando, Florida, USA, Morgan Kaufmann,