Benchmarks That Matter For Genetic Programming

John Woodward
Computing Science and Mathematics
University of Stirling
FK9 4LA Scotland UK
jrw@cs.stir.ac.uk

Simon Martin
Computing Science and Mathematics
University of Stirling
FK9 4LA Scotland UK
spm@cs.stir.ac.uk

Jerry Swan
Computing Science and Mathematics
University of Stirling
FK9 4LA Scotland UK
jsw@cs.stir.ac.uk

ABSTRACT

There have been several papers published relating to the practice of benchmarking in machine learning and Genetic Programming (GP) in particular. In addition, GP has been accused of targeting over-simplified 'toy' problems that do not reflect the complexity of real-world applications that GP is ultimately intended. There are also theoretical results that relate the performance of an algorithm with a probability distribution over problem instances, and so the current debate concerning benchmarks spans from the theoretical to the empirical.

The aim of this article is to consolidate an emerging theme arising from these papers and suggest that benchmarks should not be arbitrarily selected but should instead be drawn from an underlying probability distribution that reflects the problem instances which the algorithm is likely to be applied to in the real-world. These probability distributions are effectively dictated by the application domains themselves (essentially data-driven) and should thus re-engage the owners of the originating data.

A consequence of properly-founded benchmarking leads to the suggestion of meta-learning as a methodology for automatically designing algorithms rather than manually designing algorithms. A secondary motive is to reduce the number of research papers that propose new algorithms but do not state in advance what their purpose is (i.e. in what context should they be applied). To put the current practice of GP benchmarking in a particular harsh light, one might ask what the performance of an algorithm on Koza's lawnmower problem (a favourite toy-problem of the GP community) has to say about its performance on a *very* real-world cancer data set: the two are completely unrelated.

Keywords

genetic programming (GP), evolutionary computation, function optimization, machine learning, meta-learning, hyper-heuristics, no free lunch theorems (NFL).

GECCO'14, July 12-16, 2014, Vancouver, BC, CANADA.
Copyright 2014 ACM todo Copyright is held by the owner/author(s). Publication rights licensed to ACM. ACM 978-1-4503-2881-4/14/07
http://dx.doi.org/10.1145/2598394.2609875 ...$15.00.

1. INTRODUCTION

The aim of this paper is to highlight an emerging crisis in GP by connecting a number of recently published papers: benchmarking is poorly practiced in the GP community. We describes a methodology [4] that offers a natural solution to this crisis (one to which GP itself is ideally suited). [11, 9, 20, 33]. We summarize each of these four papers in the following points.

- Machine learning has become largely disconnected from the communities of domain experts who provided the data sets in the first place [33].
- GP lacks decent benchmark problems and has been singled-out as having elevated the status of toy problems as an effective test bed, many papers being published do not even tackle real-world problems [20, 34].
- Recent theoretical results link the performance of an algorithm with the probability distribution from which the problem instances are drawn [9, 26].
- Meta-learning [11, 23], and in particular the use of GP as a hyper-heuristic to automatically design algorithms [4], offers a natural vehicle to address the issue of how to design an algorithm for a problem class (i.e. probability distribution over problem instances) at hand, and therefore provide a new set of interesting benchmarks.

The contribution of this paper is not a new algorithm, theorem or application, but to drive the current debate about GP benchmarks, and emphasize the emerging trend in hyper-heuristics for automated design as a way-forward. The suggestion is, in view of recent theoretical results, is that GP can be employed in a hyper-heuristics framework to automatically design algorithms for problem instances drawn from a probability distribution.

In the remainder of the introduction we consider the purpose of metaheuristics and how that purpose is demonstrated by the practice of benchmarking. We then consider a recent NFL result relevant to benchmarking [9, 26], and a related issue in GP, namely that GP cannot alter its own search bias from one problem instance to the next and therefore cannot tune itself to the benchmark problem class [37]. We use the terms algorithm and metaheuristic interchangeably, as we do with problem instance and (objective) function.

1.1 Metaheuristics.

Metaheuristics [31, 19] sample intractably large search spaces so there is a trade-off when deciding which metaheuristic to apply to given problem instances. It is intended that a metaheuristic returns a "good enough, soon enough, cheap enough" solution to the current problem instance [3]. However, metaheuristics offer little in the way of guarantees about either the *quality* of the solution, or the *quantity* of computational time required.

The metaheuristic research literature is peppered with largely biologically inspired papers which begin "we propose a novel algorithm ... " [29], but rarely is it said what purpose the algorithm serves, that is, on what problem class (probability distribution of problem instances) will the proposed algorithm outperform other algorithms. While there has been some theoretical progress in understanding the behaviour of metaheuristics and GP in particular [24], the field is still largely empirical and the most common way to compare metaheuristics is still benchmarking.

1.2 Benchmarking.

Benchmarking is the practice of taking a set of problem instances, often from a diverse set of domains, and demonstrating an algorithm's performance on this set, comparing it with the performance of other algorithms. The typical approach by academic researchers is to propose a new algorithm and then compare it to other state-of-the-art algorithms in the research literature by comparing their performance on benchmarks.

A central assumption of machine learning is the test data is drawn from the same distribution as the training data [10]. It will become apparent in this paper that benchmark problem instances should be drawn from the same distribution as the real-world problem instances. In other words, if we employ GP as a hyper-heuristic to automatically design algorithms the test benchmark problem instances should be drawn from the same distribution as the training instances. It is stated that there is a hyper-focus on benchmark data sets [33], and that GP has a "toy-problem problem"; often benchmark problem instances are too simple [20]. In a similar vein to [20], this paper is not a list of benchmark recommendations, but is instead aimed at contributing to the debate.

1.3 No Free Lunch.

NFL results have been expressed informally as statements such as "over all problems, no optimization algorithm outperforms any other", or " no machine learning algorithm generalizes better than any other over the space of all problems". These results have been largely ignored by practitioners as we are rarely interested in the set of all problems. However, a new result [9, 26] (which we formally present in Section 3) states that for a given algorithm and *any* probability distribution over a set of functions, that there exists another algorithm and probability distribution with identical performance. Importantly, this result *does not depend on a uniform distribution*, as many previous NFL interpretations have done. We can ask ourselves what are the implications of this result regarding benchmarking. One implication of this result it that we should state (in some way) what probability distribution of problem instances the metaheuristic is intended for. Or to phrase it another way, a metaheuristic should not be proposed without being associated with a problem class (probability distribution over problem instances). A metaheuristic should be fit-for-purpose, that purpose being defined by the problem class. Therefore a metaheuristic should not be proposed in isolation but in the context of a problem class.

1.4 Genetic Programming.

GP is a metaheuristic which operates on a space of programs. As GP offers a way to automatically generate programs (with loops and conditional statements), [36] there are many potential applications, perhaps more so than other machine learning techniques because GP can evolve anything representable on a computer (cf. decision trees which are a limited representation).

GP borrows concepts from natural evolution: organisms *adapt* to fit their environment. In the case of GP, this means that programs evolve to fit the problem instance and the fittest programs survive. It is true that the *population of programs* adapts when GP is applied to a single problem instance, but *the way* GP searches from one problem instance to the next typically does not (cannot) alter [37]. Therefore, if we select a GP algorithm, it may perform poorly on those particular problem instances, and the GP algorithm has no way to improve its performance on future problem instances (except for intervention by a human programmer). In other words, the behaviour of GP is completely determined by the bias provided by its human designer. However, a meta-learning system, such as GP as a hyper-heuristic [4] can do this off-line. On a training set GP can alter bias, which is then fixed in a testing phase [39, 38, 12].

The outline of the remainder of this article is as follows. In Section 2 we connect together a number of recent papers relating to metaheuristics, GP, meta-learning and NFL. Section 3 presents a mathematical framework and theorems illustrating the trade-off between metaheuristic performance and problem classes. In Section 4 we give examples of problem classes pertaining to bin-packing and the traveling salesman problem. In Section 5 we summarize the argument that benchmarks need to be relevant, not isolated, and that meta-learning and hyper-heuristics to generate heuristics can address this.

2. BACKGROUND WORK

2.1 Reconnecting Machine Learning To The Real-world.

The academic practice of machine learning has become disconnected from the communities which originally supplied the data sets [33]. Thus, as these communities rarely communicate, there is little long-term benefit to the application-domain community (or in fact the machine learning community itself). Wagstaff [33] concludes that progress in the field of machine learning has little impact on the community that initially supplied the actual data set. Data sets are simply treated as rows and columns of numerical data to be freely downloaded and the real-world *context* of the data has been lost.

2.2 Realistic Benchmarks for Genetic Programming.

McDermott et al. [20] criticize the lack of rigor in the choice of benchmark problem instances. The standard, often-used problem instances (so-called toy problems, where the optimal solutions are often *known* in advance) do not reflect the complexities which are experienced when tackling real-world problems, where the optimal solutions are often *not known.* This defeats the whole purpose of applying a metaheuristic to a problem as the solution (global optimum) is already known.

The development of a machine learning algorithm is typically a three stage process; training, testing and application [22]. In the training stage, the algorithm learns by adjusting its bias (probability distributions). Its performance is confirmed on an independent test set to check that over-fitting is not a problem (i.e. we have generalized beyond the training set). Once this stage has been passed to the satisfaction of the GP researcher, then the algorithm can be applied problem instances from the application domain. The set of training and testing instances should represent the application instances, and therefore we should make some attempt to describe what characterizes these problem instances, for example their source (see 4). Johnson [18] states; "use instance test-beds that can support general conclusions". So if we demonstrate the utility of a metaheuristic on a set of test problem instances, then it follows that this set should be drawn from the same probability distribution as in the application stage as is common practice in the machine learning community, but no so much the optimization community. It seems as if GP has become separated from the third, and arguably the most important part of the developmental process; the application.

2.3 No Free Lunch And Benchmarking With Problem Classes.

NFL effectively states that given a uniform distribution over the set of *all* problem instances, no single algorithm outperforms any other [35]. This sounds like a negative and defeatist result, as it comprehensively covers *all* functions. NFL holds for cross-validation [40] and early-stopping [6], two major sources of bias in machine learning. We are not aware of a NFL result for feature selection [2], for example, but nor would we be surprised if one existed. However feature selection, which is based on information theory, remains a useful bias in practice, as are cross-validation and early-stopping.

NFL has become the holy grail in some research communities [14], and been called a "show-stopper" [11] (which was firmly answered in the negative and on which this paper builds its implications). NFL applies to a vanishingly small number of the subsets of all functions and is unlikely to apply to real-world situations [15, 16, 17]. It has conclusively been shown by theoretical analysis, that on restricted sets of problems, one algorithm *does* outperform another [7, 8]. We advocate that these restrictions should be imposed by the application domain itself. In this paper we interpret "restricted sets of problems" as probability distributions over problem instances (i.e. problem classes).

Links have been made between the performance of an algorithm and the probability distribution over benchmark problem instances [9, 26]. An earlier similar result connects the performance of an algorithm on a single problem instance [28]. Essentially two search algorithms (which are permutations of one another), and two functions (which are the inverse permutations of one another), have identical and therefore indistinguishable performance. Further results have been obtained concerning the use of arbitrary benchmarks to evaluate randomized algorithms [9, 26].

2.4 Bias In Algorithms And Problem Classes.

Mitchell [21] defines bias as any basis (probability) for choosing one generalization over another, other than strict consistency with the observed training problem instances. Specifying a bias is equivalent to specifying an algorithm as we are only interested in the output of the algorithm (i.e. what it does, not how it does it). Bias determines which items in the search space the algorithm will visit. Mitchell [21] states that there needs to be a bias in order to learn. Similarly, there must be bias over the problem class if there is anything to be learnt about the problem instances on *average.*

A problem class F is a probability distribution over problem instances $F = (p_f(f_1), \dots, p_f(f_i), \dots)$. Each time an algorithm is executed, it will return the best solution it has found. If we consider an algorithm as a stochastic search process then when the is algorithm executed again, it may yield a different solution. The *bias* of an algorithm a on a function f is the probability that a will return a given solution. This notion of problem classes and algorithms seen as probability distributions corresponds to the geometric perspective in [35] and Lemma 1 in [11]. Essentially, to gain good performance of an algorithm on a problem class, we want the bias of the algorithm to align with the bias of the problem class. We want the algorithm to sample the space with high probability, were we expect solutions to be found. If we increase the probability of an algorithm sampling some region(s) of the search space, then it follows that the probability of sampling the remainder of the search space will decrease (as probability is conserved). We want the algorithms to visit part of the search space solutions are likely to reside, given the problem class at hand.

2.5 From No Free Lunch To Meta-learning.

NFL is of little relevance to research in machine learning (who is interested in learning or optimizing a white noise effectively random function [13]). However, NFL *does* make the contribution that whenever an algorithm performs well on some problem instances, then it must perform poorly on some other problem instances [11]. In the absence of bias, enumerating a set of fixed-arity binary functions demonstrates that, given a subset of input-output pairs (i.e. the training set) then it is impossible to generalize over the remaining input-output pairs, (i.e. the testing set) [11]. This result is referred to as Conservation of Generalization [27]. [11] further make the claim that this theorem supports the argument for meta-learning.

Let us take the example of classification [10]. The task is to design a classifier, which maps an example e to a class c (i.e. $f(e) = c$). Domain experts may well be interested in the problem of designing a classifier for a single problem (e.g. automatic number plate recognition: mapping photographs of vehicle licence plates to the plate number). However, as machine learning researchers we are usually interested in a method of producing classification algorithms [22, 25]. Given that we (metaheuristic researchers) are interested in, not a single classification problem, but many, this gives us a probability distribution of problem instances F (i.e. a problem class). In [11] the following Lemma is proved: knowing F, the probability of encountering an arbitrary function f, is equivalent to knowing $p(c|e)$, the probability of class membership for an arbitrary example e. The question then is how to arrive at $p(c|e)$, and one answer is meta-learning to which we now turn our attention.

2.6 Meta-learning

Meta-learning (also called 'learning to learn' [32] and hyper-heuristics [4]) is the idea that the base-level learns about single problem instances, and the meta-level learns about the problem class (i.e. a sample of problem instances drawn from the probability distribution associated with the problem class) [32]. In principle there is no difference between learning at the base-level and meta-level. Therefore just as a central assumption in machine learning is that the training and test data at the base-level are drawn from the same probability distribution, so too should the training and test instances be drawn from the same probability distribution (i.e. problem class). In both cases the search process can be thought of as being conducted over a set of items in a search space. In the former, the set of items being solutions to a problem instance, and the latter either being learning algorithms or some component (e.g. parameters) of the learning algorithm [1]. The meta-level learns invariants of the problem class. Consider the example of problem instances of face recognition of individual people. An invariant common across all face recognition tasks include translational and rotational invariance (i.e. if any face, is rotated or translated, it is still the same face) [32]. If an automated learning system is to learn about a set of related problem instances, then we should ask what we suspect these instances have in common. Just as we would not expect a machine learning algorithm to learn a source of random noise, nor should we expect algorithms to perform well on unrelated (i.e. randomly selected) problem instances. Meta-learning offers a way of automatically building learning algorithms in response to a specific problem class.

3. A MATHEMATICAL FRAMEWORK

We begin this section with a presentation of an established framework for search operators [28, 30, 17]. The two theorems below are taken from [26, 9]. Let X and Y be finite sets and $f : X \to Y$ be *an objective function* (i.e. $y = f(x)$), and $\mathcal{F}$ is the set of all such functions. The domain X and co-domain Y are fixed but f may vary. Let σ be a permutation. As $|X|$ and $|Y|$ are finite, so too is $|\mathcal{F}|$. Define a *trace* T_m (of size m $(0 \leq m \leq |X|)$) corresponding to f to be a sequence of pairs $(x, y) \in X \times Y$ where x components are unique. Note $T_{|\mathcal{F}|} \equiv f : X \to Y$ as $T_{|\mathcal{F}|}$ contains all the information about f i.e. $T_{|\mathcal{F}|}$ is effectively a look-up-table for f. X^m has corresponding projections $T^x_m : T_m \to X^m$ and $T^y_m : T_m \to Y^m$.

$T_m = [(x_0, f(x_0)), \ldots, (x_m, f(x_m))]$
$T^x_m = [(x_0), \ldots, (x_m)]$
$T^y_m = [(f(x_0)), \ldots, (f(x_m))]$

A trace is a list of the sampled items x_i in the search space X, together with their associated objective values $f(x_i)$. T^x_m is a list of unique items visited (i.e. a permutation of X) and T^y_m is the list of the corresponding objective values.

We define a *search operator* as a function $a : T_m \to X$ which takes a trace T_m and returns x with $x \notin T^x_m$. Thus, application of the search operator a on the objective function f builds a longer trace.

$$T_{m+1} = T_m \oplus (a(T_m), f(a(T_m)))$$

where $\oplus$ appends $(a(T_m), f(a(T_m)))$ to the end of T_m We define a *performance vector* $v_m(a, f) = T^y_m$ (i.e. the performance vector is a list of values in Y, corresponding to the order in which the domain values are sampled in X). We define a *performance measure* $M : v_m \to \mathbb{R}$. We define a *problem class* F to be a probability distribution over $f \in \mathcal{F}$, i.e. the probability $F(f)$ of encountering f as a function optimization problem. The overall performance measure $M_0(a)$ is given by:

$$M_0(a) = \sum_{f \in \mathcal{F}} F(f) M(v(a, f))$$

[30] (i.e. the performance measure is now simply the weighted sum of the performance on each individual function, weighted by the probability of sampling that function). We define a *probabilistic performance vector* $V_m(A, F)$ where the i component of $V_m(A, F)$, is the average of the ith component of T^y_m, generated from the pair (a, f) with a and f drawn from A and F We say an NFL result applies to F if and only if $M_0(a) = M_0(b)$ for any pair of search operators a and b.

Theorem 1. Given an objective function f, a search operator a, a permutation σ and m $(0 \leq m \leq |X|)$

$$v_m(\sigma(a), f) = v_m(a, \sigma(f))$$

This result is summed up as follows. a search operator $\sigma(a)$ applied to a function f and a search operator a applied to a function $\sigma(f)$ have the same performance as they produce identical y-component traces T^y_m.

Theorem 2. Given a probability distribution F over the set of all functions $\mathcal{F}$, and a probability distribution A over the set of all search operators $\mathcal{A}$, and m with $(0 \leq m \leq |X|)$, there exists a probability distribution $G = F(\sigma)$ over all functions, and a probability distribution $B = A(\sigma)$ over the set of all search operators such that

$$V_m(A, F) = V_m(B, G)$$

In short, the implication of this theorem is the bias of an algorithm needs to align with the bias of the probability distribution over functions.

4. DISCUSSION

In this section we discuss two examples (traveling salesman problem (TSP) and bin packing) of application domains that naturally give rise to problem classes. As it has been established in the previous section, from a theoretical point of view it only makes sense to compare algorithms on problem instances drawn from a given probability distribution. As we do not have explicit access to these distributions, we have to do with samples taken from them, as is done in machine learning.

4.1 Bin Packing As An Example Of Problem Classes.

Let us consider how problem classes arise naturally in the real-world. Imagine two logistics companies, each with a different customer base. These will each give rise to two different sets of bin-packing problem instances (associated with two different probability distributions of item sizes) and therefore can be considered to be drawn from two bin-packing problem classes. The two companies are therefore the source of two different problem classes.

To put benchmarking in the specific terms of the domain of online bin-packing, there should not be a "standard" set of benchmarks, as the problem instances we solve will belong to specific probability distributions of item sizes. While we can have a "general" (all round) bin-packing solver i.e. one trained on a uniform distribution of item sizes in the range, there is always room for specialization if we have information about the problem class (i.e. given we have items in the range [lower bound, upper bound]). It has been shown that bin packing algorithms that were automatically designed for a specific problem class outperform bin packing algorithms that were designed for different problem classes [5]. Furthermore, we can train our algorithm on bin packing problem instances of a known range of sizes, and test the algorithm on problem instances with the *same* range of sizes but different number of items while still achieving scalable performance.

4.2 Traveling Salesman Problem As An Example Of Problem Classes.

The Traveling Salesman Problem (TSP) is the problem of finding a tour that visits all cities in a set only once, while minimizing the total distance travelled. TSP problem instances could be generated by the following two different methods. One TSP problem class corresponding to points uniformly distributed in the unit square. It turns out, one can obtain very good estimates of what the expected value of that bound (i.e. tour length) is for any number of cities [18]. An alternative TSP problem class could be cities distributed according to a Gaussian distribution over the unit circle. Either a theoretical or empirical analysis would give expectation values for any number of cities assuming a given TSP problem class (i.e. probability distribution over TSP instances).

Let us extend the artificial example of square and circular TSP problem classes to a real-world situation. An example of a real-world TSP problem class would be having delivery points distributed according to a population density of customers for a given company. A set of TSP problem instances generated by different company (possibly is a different location) will, more than likely, have a different probability distribution associated with it. And it is this difference in probability distributions which will give one algorithm leverage over another. To really emphasize the point that real problem instances are grounded in a problem class, it is hard to see how a TSP instance could arise without being in the context of other TSP instances. There are rarely one-off TSP instances we want to solve, but typically a whole sequence of TSP instances generated by a single source (e.g. a logistics company).

4.3 The Comparison of Algorithms On Problem Classes.

One measure of the performance of an algorithm is to examine its behaviour on a single problem instance. However, it does not make sense to compare single runs of an algorithm, as they are stochastic so need to be executed multiple times. Neither are we typically interested in a single problem instance anyway [7], but typically a set of problem instances [32]. We could compare two stochastic algorithms by repeatedly executing them on a single problem instance. However, this is somewhat artificial as we only want to execute an algorithm once on a given problem instance. It seems to make more sense to compare our algorithms on multiple problem instances drawn from a problem class (i.e. one algorithm is only run once on one problem instance).

Also when developing metaheuristics, it is tempting to test them on benchmark problem instances during the development (training) stage. And if the performance is poor we reengineer the metaheuristic. However, if a problem instance is used in the development stage, it should not later be used again to demonstrate the utility of the metaheuristic. In other words we should adopt a strict machine learning regime, which separates training and testing problem instances. A set of benchmark instances should therefore be representative of the underlying problem class of interest, which are used in the development stage to learn about the problem class, and a separate set of test problem instances are used to show something of use has been learned in the training phase which can be used in the testing phase. In short, training and testing instances should be separate and not contaminate one another. Developing a metaheuristic by using problem instances in the training phase, and then using those instances in the testing phase to demonstrate the performance of the algorithm for the purposes of publication is equivalent in machine learning to over-fitting.

4.4 Sampling To Estimate Probability Distributions.

We usually do not have explicit access to the probability distribution over the set of problem instances, except in the case of toy problems (where we can define our own probability distribution). We can only obtain samples of problem instances from real-world problem classes. By the law of large numbers, as we sample more and more problem instances we will gain more information and tend towards the expected probability distribution. Thus, while we do not have explicit access to the probability distribution responsible for the problem class, increasing the sample size will give a better and better approximation.

5. SUMMARY AND CONCLUSION

We have argued for the utilization of realistic benchmarks for GP based on a number of recent papers. Firstly [33] observes a proliferation of machine learning papers that evaluate new algorithms on a handful of *isolated* benchmark data sets and those improvements are not fed-back to the originators of the data. As machine learning researchers, we should remind ourselves of the importance of feedback, as it is a fundamental mechanism in the machine learning process itself.

Secondly, GP is singled out as a particularly bad example of machine learning practice where toy problems have enjoyed attention they do not rightly warrant [20]. They make a number of recommendations for establishing properties that a suite of benchmark instances should have, some of which are inconsistent with the approach advocated in this paper. The main objection being that there should not be a standard benchmark suite, but a set of problem instances dictated directly by the real-world problem instances one is attempting to solve. In other words, isolated context-free individual benchmark problem instances of are little use. What is needed are problem instance generators (i.e. a probability distribution from which problem instances are drawn). This would reduce the chances of over-tuning a metaheuristic to a given problem instance.

Thirdly, a theoretical result in [9, 26] links the performance of an algorithm with instances drawn from a probability distribution. One of the consequences of this is that the bias of an algorithm needs to align with the bias of a probability distribution over problem instances [11, 35]. The outcome of this is that an algorithm should be tuned to a problem class to obtain better than average performance. When the bias of an algorithm precisely aligns with the problem class, then that is the best algorithm for that problem class.

Finally, a solution is provided to the problem of designing algorithms tailored to a specific problem class; meta-learning [11] and hyper-heuristics [4]. While base-level learning is concerned with solving a specific problem instance, meta-level learning is concerned with solving many problem instances from a problem class (i.e. learning about the problem class). In other words, a meta-level provides a channel through which learning from problem instance in a training set can be retained to improve performance on problem instances in the application stage.

In short, it does not make sense to nominate an algorithm without simultaneously nominating the set of problem instances to which it is to be applied (i.e. an algorithm is appropriate *for* a problem class). By analogy with the inspiring metaphor of GP, just as organisms evolve in response to environmental changes (i.e. survival of the fittest), and objects are engineered in response to needs (i.e. fit for purpose), so too should we automatically design algorithms in response to grounded benchmark instances (i.e. drawn from a probability distribution).

There are at least two reasons why we would want to employ meta-learning. Firstly, we could manually generate new algorithms (i.e. 'hand-coding'), and test them on said benchmarks (which seems to be the most widely adopted approach). However, this itself is just a generate-and-test approach (no different to GP itself) that can be automated. Secondly, with real-world problems, the probability distribution of problem instances is likely to change over time, so rather than having to hand-code new algorithms in response to this change, they can be generated on demand for a new set of benchmark instances.

We hope that this paper goes some way of addressing the issues raised in [33, 20]. We provided support for engaging with real-world problems via the adoption of meta-learning as a problem domain for GP.

6. REFERENCES

[1] Juergen Branke and Jawad Asem Elomari. Meta-optimization for parameter tuning with a flexible computing budget. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, GECCO '12, pages 1245–1252, New York, NY, USA, 2012. ACM.

[2] Gavin Brown. A new perspective for information theoretic feature selection. In *International Conference on Artificial Intelligence and Statistics*, pages 49–56, 2009.

[3] Edmund Burke, Graham Kendall, Jim Newall, Emma Hart, Peter Ross, and Sonia Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In Fred Glover, Gary Kochenberger, and Frederick S. Hillier, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research and Management Science*, pages 457–474. Springer New York, 2003.

[4] Edmund K Burke, Mathew R Hyde, Graham Kendall, Gabriela Ochoa, Ender Ozcan, and John R Woodward. Exploring hyper-heuristic methodologies with genetic programming. In *Computational intelligence*, pages 177–201. Springer Berlin Heidelberg, 2009.

[5] Edmund K. Burke, Matthew R. Hyde, Graham Kendall, and John Woodward. Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 1559–1565, London, 7-11 July 2007. ACM Press.

[6] Zehra Cataltepe, Yaser S. Abu-Mostafa, and Malik Magdon-Ismail. No free lunch for early stopping. *Neural Comput.*, 11:995–1009, May 1999.

[7] Stefan Droste, Thomas Jansen, and Ingo Wegener. Perhaps Not a Free Lunch But At Least a Free Appetizer. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-1999*, pages 833–839, San Francisco, CA, 1999. Morgan Kaufmann Publishers, Inc.

[8] Stefan Droste, Thomas Jansen, and Ingo Wegener. Optimization with randomized search heuristics - the (a)nfl theorem, realistic scenarios, and difficult functions. *Theor. Comput. Sci.*, 287(1):131–144, 2002.

[9] Edgar A. Duéñez Guzmán and Michael D. Vose. No Free Lunch and Benchmarks. *Evolutionary Computation*, pages 1–20, March 2012.

[10] Peter Flach. *Machine Learning: The art and science of algorithms that make sense of data*. Cambridge University Press, September 2012.

[11] C. Giraud-Carrier and F. Provost. Toward a Justification of Meta-learning: Is the No Free Lunch Theorem a Show-stopper? In *Proceedings of the ICML-2005 Workshop on Meta-learning*, pages 12–19, 2005.

[12] Libin Hong, John Woodward, Jingpeng Li, and Ender Ozcan. Automated design of probability distributions as mutation operators for evolutionary programming using genetic programming. In Krzysztof Krawiec, Alberto Moraglio, Ting Hu, A. Sima Uyar, and Bin Hu, editors, *Proceedings of the 16th European Conference on Genetic Programming, EuroGP 2013*, volume 7831 of *LNCS*, pages 85–96, Vienna, Austria, 3-5 April 2013. Springer Verlag.

[13] Marcus Hutter. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability*. EATCS. Springer, Berlin, 2005. 300 pages, http://www.hutter1.net/ai/uaibook.htm.

[14] Marcus Hutter. A complete theory of everything (will be subjective). *Algorithms*, 3(4):329–350, 2010.

[15] Christian Igel and Marc Toussaint. On classes of functions for which no free lunch results hold. *Inf. Process. Lett.*, 86(6):317–321, 2003.

[16] Christian Igel and Marc Toussaint. Recent results on no-free-lunch theorems for optimization. *CoRR*, cs.NE/0303032, 2003.

[17] Christian Igel and Marc Toussaint. A no-free-lunch

theorem for nonuniform distributions of target functions. *Journal of Mathematical Modeling and Algorithms*, page 313, 2004.

[18] D. S. Johnson. A Theoretician's Guide to the Experimental Analysis of Algorithms. In *5th and 6th DIMACS Implementation Challenges*. American Mathematical Society, 2002.

[19] Sean Luke. *Essentials of Metaheuristics*. Lulu, second edition, 2013. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[20] James McDermott, David R. White, Sean Luke, Luca Manzoni, Mauro Castelli, Leonardo Vanneschi, Wojciech Jaskowski, Krzysztof Krawiec, Robin Harper, Kenneth De Jong, and Una-May O'Reilly. Genetic programming needs better benchmarks. In *GECCO '12: Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, pages 791–798, Philadelphia, Pennsylvania, USA, 7-11 July 2012. ACM.

[21] Tom M. Mitchell. The need for biases in learning generalizations. Technical report, Rutgers University, New Brunswick, NJ, 1980.

[22] Gisele L. Pappa and Alex A. Freitas. *Automatically Evolving Data Mining Algorithms*, volume XIII of *Natural Computing Series*. Springer, 2010.

[23] Gisele L Pappa, Gabriela Ochoa, Matthew R Hyde, Alex A Freitas, John Woodward, and Jerry Swan. Contrasting meta-learning and hyper-heuristic research: the role of evolutionary algorithms. *Genetic Programming and Evolvable Machines*, pages 1–33.

[24] Riccardo Poli, Leonardo Vanneschi, William B. Langdon, and Nicholas Freitag Mcphee. Theoretical results in genetic programming: The next ten years? *Genetic Programming and Evolvable Machines*, 11(3-4):285–320, September 2010.

[25] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[26] Jonathan E. Rowe and Michael D. Vose. Unbiased black box search algorithms. In *GECCO*, pages 2035–2042, 2011.

[27] C. Schaffer. A conservation law for generalization performance. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 259–265. Morgan Kaufmann, 1994.

[28] 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 (GECCO-2001)*, pages 565–570. Morgan Kaufmann, 2001.

[29] Kenneth Sörensen. Metaheuristics the metaphor exposed. *International Transactions in Operational Research*, 2013.

[30] Matthew J. Streeter. Two broad classes of functions for which a no free lunch result does not hold. In *Proc. Genetic and Evolutionary Computation Conference GECCO-2003*, pages 1418–1430. Morgan Kaufmann, 2003.

[31] El-Ghazali Talbi. *Metaheuristics - From Design to Implementation*. Wiley, 2009.

[32] Sebastian Thrun and Lorien Pratt, editors. *Learning to learn*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[33] Kiri Wagstaff. Machine learning that matters. *CoRR*, abs/1206.4656, 2012.

[34] David R White, James McDermott, Mauro Castelli, Luca Manzoni, Brian W Goldman, Gabriel Kronberger, Wojciech Jaśkowski, Una-May Oï£¡Reilly, and Sean Luke. Better gp benchmarks: community survey results and proposals. *Genetic Programming and Evolvable Machines*, 14(1):3–29, 2013.

[35] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, April 1997.

[36] John R Woodward. Evolving turing complete representations. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 2, pages 830–837. IEEE, 2003.

[37] John R Woodward. The necessity of meta bias in search algorithms. In *Computational Intelligence and Software Engineering (CiSE), 2010 International Conference on*, pages 1–4. IEEE, 2010.

[38] John R. Woodward and Jerry Swan. The automatic generation of mutation operators for genetic algorithms. In Gisele L. Pappa, John Woodward, Matthew R. Hyde, and Jerry Swan, editors, *GECCO 2012 2nd Workshop on Evolutionary Computation for the Automated Design of Algorithms*, pages 67–74, Philadelphia, Pennsylvania, USA, 7-11 July 2012. ACM.

[39] John Robert Woodward and Jerry Swan. Automatically selection heuristics. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, GECCO '11, pages 583–590, New York, NY, USA, 2011. ACM.

[40] Huaiyu Zhu and Richard Rohwer. No free lunch for cross-validation. *Neural Comput.*, 8:1421–1426, October 1996.