There Is a Free Lunch for Hyper-Heuristics, Genetic Programming and Computer Scientists

Riccardo Poli and Mario Graff

School of Computer Science and Electronic Engineering, University of Essex, UK
{rpoli,mgraff}@essex.ac.uk

Abstract. In this paper we prove that in some practical situations, there is a free lunch for hyper-heuristics, i.e., for search algorithms that search the space of solvers, searchers, meta-heuristics and heuristics for problems. This has consequences for the use of genetic programming as a method to discover new search algorithms and, more generally, problem solvers. Furthermore, it has also rather important philosophical consequences in relation to the efforts of computer scientists to discover useful novel search algorithms.

1 Introduction

Hyper-heuristics are often defined as "heuristics to choose heuristics" [4]. A *heuristic* is considered as a rule-of-thumb or "educated guess" that reduces the search required to find a solution. The difference between meta-heuristics and hyper-heuristics is that the former operate directly on the problem search space with the goal of finding optimal or near-optimal solutions. The latter, instead, operate on the heuristics search space (which consists of all the heuristics that can be used to solve a target problem). The goal then is finding or generating high-quality heuristics for a problem, for a certain group of instances of a problem, or even for a particular instance. Put another way, hyper-heuristics are search algorithms that don't directly solve problems, but, instead, search the space of solvers, searchers, meta-heuristics or heuristics that can then solve the problems of interest.

The situation is depicted in Figure 1 where a hyper-heuristic (the point drawn in thick line in the leftmost circle in the figure) is sampling a subset of elements in the space of heuristics (the middle circle) in order to find a good solver for a problem, represented as a search space endowed with a fitness function (the rightmost circle in the figure). Each solver sampled by the hyper-heuristic needs to be executed in order to evaluate the quality of such a solver. This, in its turn, involves the sampling of a subset of elements of the space of solutions. Since fitness values are associated to these, information percolates up the hierarchy to guide the hyper-heuristic. Note that there are many hyper-heuristics in the space of hyper-heuristics. Each may sample the space of heuristics differently, which in turn may lead to sampling the solution space differently.

Genetic Programming (GP) [3,15,21] has been very successfully used as a hyper-heuristic. For example, GP has evolved competitive SAT solvers [1,2,8,14], evolutionary algorithms [16,17], bin packing algorithms [5,6,22], particle swarm optimisers [7,19,20], and travelling salesman problem solvers [11,12,13,18].

L. Vanneschi et al. (Eds.): EuroGP 2009, LNCS 5481, pp. 195–207, 2009.

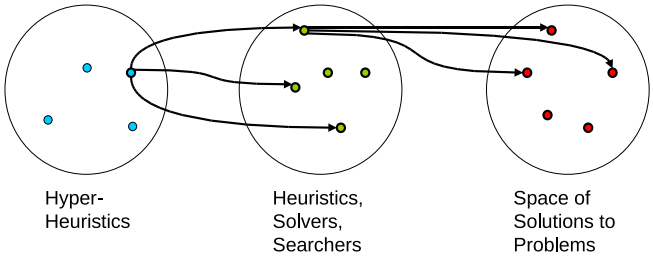

Fig. 1. A hyper-heuristic exploring the space of search algorithms (or more generally solvers) for a particular problem

The question we want to investigate in this paper is whether or not the No-Free-Lunch (NFL) theory of Wolpert and Macready [26] and its more modern reformulations have implications for hyper-heuristics and GP (when used to invent problem solvers and other search algorithms). In principle, there would seem to be no a priori reason why NFL would not apply to hyper-heuristics in the sense that NFL is said to apply to all forms of search. However, nobody has actually ever explored the specific applicability of NFL to meta-search (i.e., hyper-heuristics). In this paper we explore under what conditions NFL may apply to hyper-heuristics and what implications the availability or otherwise of a "free lunch" may have on researchers interested in developing search algorithms, including GP.

The paper is organised as follows. In Section 2 we cover the basics of NFL. In Section 3 we consider the implications of NFL for hyper-heuristic search. Section 4 shows that free lunches are possible for hyper-heuristics. Exactly what this means is explained in Section 5. We then look at the consequences this has in a broader context in Section 6. We consider higher-order hyper-heuristic search in Section 7. Finally, we provide some conclusions in Section 8.

2 No Free Lunch

Informally speaking, the NFL theory [26] states that, when evaluated over all possible problems, all algorithms are equally good or bad irrespective of our evaluation criteria.

In the last decade there have been a variety of results that have refined and specialised NFL (see for example [25] for a comprehensive review and [24] for a discussion on its implications). One important result states that if one selects a set of fitness functions that are *closed under permutation* (more on this below) then the expected performance of any search algorithm over that set of problems is constant, i.e., it does not depend on the algorithm we choose [23] nor the chosen performance measure. In formulae, if $\mathcal{F}$ is a set of fitness functions closed under permutation, we have that

$$\sum_{f\in\mathcal{F}} P(f,a_1) = \sum_{f\in\mathcal{F}} P(f,a_2) \tag{1}$$

for any pair of pair of (non-resampling) search algorithms a_1 and a_2 and for any performance measure P. Furthermore, [23] showed that the connection between closure and

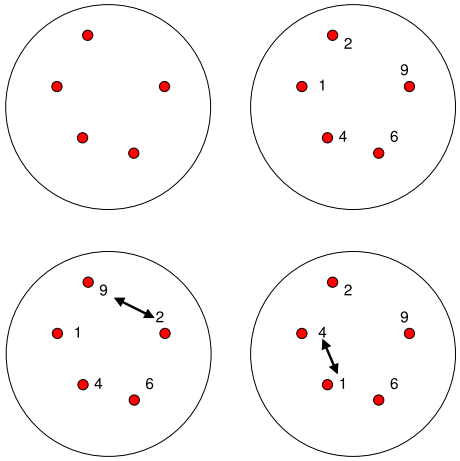

Fig. 2. A sample search space (top left), a problem (i.e., an assignment of fitness to the elements of the search space (top right), and two sample permutations of the fitness function (bottom)

f1	2
f2	1
f3	4
f4	6
f5	9

f1	1
f2	2
f3	4
f4	6
f5	9

f1	4
f2	1
f3	2
f4	6
f5	9

f1	6
f2	1
f3	4
f4	2
f5	9

f1	9
f2	1
f3	4
f4	6
f5	2

... (120) .

Fig. 3. A fitness function seen as a vector and its permutations

NFL is an "if and only if" one. That is, it is also the case that two arbitrary algorithms will have identical performance over a set of functions only if that set of functions is closed under permutation.

What does it mean for a set of functions to be closed under permutation? The NFL is limited to finite search spaces. A fitness function is an assignment of fitness values to the elements of the search space, as exemplified graphically in Figure 2 (top). A permutation of a fitness function is simply a rearrangement of the fitness values originally allocated to the objects in the search space. Examples of permutations are shown at the bottom of Figure 2. If we enumerate the elements of a search space according to some scheme, we can then represent a fitness function as a vector that stores the fitness associated to each point of the space. For example, if we enumerate the five points in Figure 2 (top left) in anti-clockwise order, the fitness function in Figure 2 (top right) can be represented as indicated in the leftmost vector in Figure 3. In this case, permuting a fitness function simply means shuffling the elements of the vector representing it. A subset of permutations of our sample fitness function are shown in Figure 3. A set of problems/fitness functions is closed under permutation if, for every function in the set, all possible shuffles of that function are also in the set. If all the possible 120

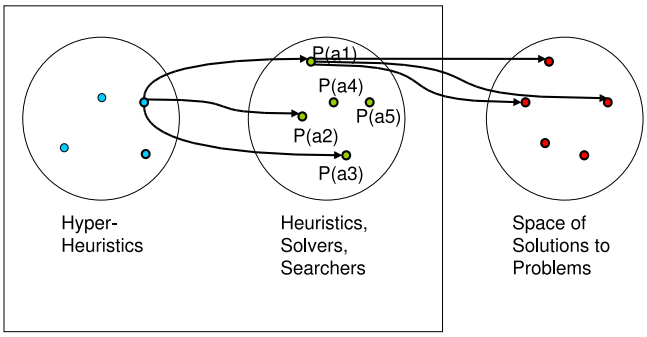

Fig. 4. In the space of hyper-heuristics (exploring the space of search algorithms/solvers for a particular problem), fitness is determined by how well each algorithm solves the original problem

permutations (including the identity permutation) of the elements of the leftmost vector in Figure 3 were considered, then the resulting set of fitness functions would be closed under permutation.

3 Implications of No-Free Lunch for Hyper-Heuristics

So, let us see exactly what the applicability of NFL would imply in the case of hyper-heuristics. Firstly, let us consider what guides the search of hyper-heuristics.

Hyper-heuristics require some feedback from the problem domain in order for them to have any hope to discover good heuristics. As illustrated in Figure 4, they are guided by a performance measure, $P(a)$, that indicates how good each heuristic a (for "algorithm") is in relation to the problem at hand. Note that while in practice $P(a)$ is computed (in fact, it can often only be estimated) only for the algorithms actually sampled by the hyper-heuristic, from a logical point of view we can imagine that all possible algorithms in the heuristics space have a precomputed performance measure. In this situation, we can forget about the original problem space, and, instead, look at the algorithm space as an ordinary search space endowed with an objective measure. That is, we can focus on the rectangular region marked out in Figure 4.

In order to make further progress, we need to generalise and formalise our treatment a little. Let $\mathcal{P}$ be a set of problems for which we want to find a good solver via hyper-heuristic search. The set $\mathcal{P}$ may include a single problem instance (as exemplified in Figures 1 and 4), a set of problem instances, or even a whole class of problems (as illustrated in Figure 5). However, for simplicity, let us imagine that all such problems share the same solution space Ω and that Ω is finite. If, for example, we were interested in solving SAT problems, we imagine that $\mathcal{P}$ is a set of SAT instances with the same number of variables, n, so that $\Omega = \{0,1\}^n$ for all problems in $\mathcal{P}$.

What distinguishes a problem from another is the assignment of fitness to the elements of Ω. Let $\mathcal{G}$ be the set of fitness functions associated to the problems in $\mathcal{P}$. $\mathcal{G}$ corresponds to the rectangular area to the right of Figure 5. We can think of the elements of $\mathcal{G}$ graphically as in the figure or as vectors/tables as indicated in Section 1.

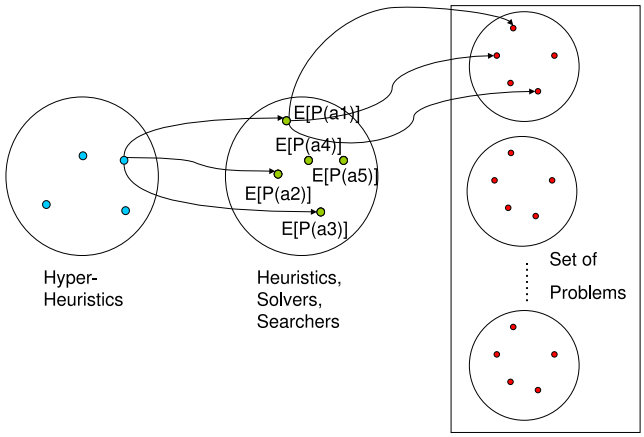

Fig. 5. A hyper-heuristic exploring the space of search algorithms (or more generally solvers) for a particular set of problems

As we discussed above, a hyper-heuristic is a search algorithm that explores the space of search algorithms. Let us call this latter space Λ.[1] Λ corresponds to the circle in the middle of Figure 5. In order to make decisions as to which heuristics in Λ to explore and in which order, a hyper-heuristic needs to be guided by some objective function that evaluates how well each of the heuristic it samples solves the problems in $\mathcal{P}$. Let us look at this more closely.

Given a problem with fitness function $g \in \mathcal{G}$, a searcher/heuristic will sample Ω to achieve some goal. Often the goal is to find good values of fitness and we would like the heuristic to find a member of Ω that has the largest (or smallest, if minimising) value in g using the smallest possible number of trials. However, one may want to adopt other performance measures. Let P be one such performance measure. Because we don't have just one problem, but a set of problems, $\mathcal{P}$, the performance measure $P(a)$ for an algorithm $a \in \Lambda$ will need to be applied to all such problems and then averaged to produce an estimation of how good an element of the space of searchers, Λ, is. Naturally, in the presence of stochasticity, P will be noisy. For the purpose of this argument, however, we will imagine that we can associate to each element a of Λ the exact expected value of $P(a)$, $E[P(a)]$, where the expectation is taken over all problems in $\mathcal{P}$, i.e., all functions in the set $\mathcal{G}$. As we indicated above, once the performance of each algorithm/solver is available, we can forget about the problem space, and, instead, look at the algorithm space as an ordinary search space endowed with a "fitness measure" $E[P(a)]$, as illustrated in Figure 6 (top).

Of course, as we know from the NFL theory summarised in Section 2, if the set of fitness functions, $\mathcal{G}$, characterising our problem set, $\mathcal{P}$, is closed under permutation, then all non-resampling search algorithms will show identical average performance over $\mathcal{G}$, i.e., $\sum_{p \in \mathcal{P}} P(a,p) = \text{constant}$. So, $\forall a : E[P(a)] = \text{constant}$.

[1] Naturally, Λ may depend on the problems in $\mathcal{P}$, the solution space Ω, the adopted representation for search behaviours, etc. We will not investigate these details here.

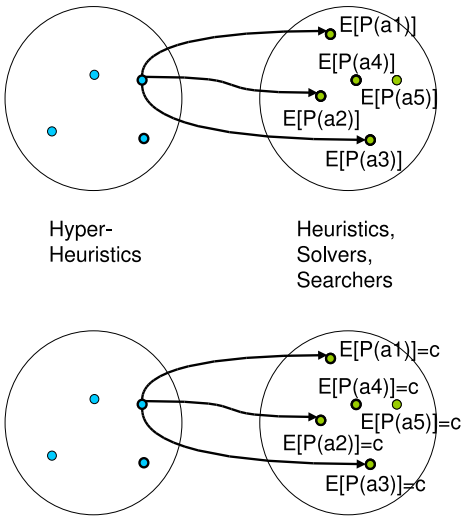

Fig. 6. The search as perceived from the point of view of a hyper-heuristic (top). If the problem set is closed under permutation, the "fitness function" over the heuristic search-space (Λ) is flat (bottom). In this case, searching for hyper-heuristics is futile, since every single one of them is equally good.

This has a simple and important implication. A hyper-heuristic using $E[P(a)]$ (for any P) as its fitness measure over Λ will find that the fitness landscape is flat! This situation is illustrated in Figure 6 (bottom). So, the hyper-heuristic has nothing to guide it. In fact, no hyper-heuristic can have any guidance whatsoever. Therefore, a hyper-heuristic that chooses a random solver from Λ is as good as any other. In other words, *hyper-heuristic search makes no sense if the set of problems for which we want to find a solver is closed under permutation*.

4 Free Lunches for Hyper-Heuristics

How likely is the situation depicted above? Igel and Toussaint [9] were able to show that if one considers all possible sets of functions with a given domain and a given co-domain, in most conditions *the sets that are closed under permutation, and, hence, over which NFL applies, represent a tiny fraction of the whole*. From the point of view of hyper-heuristic search, this would suggest that sets of problems $\mathcal{P}$ for which there is no point in using hyper-heuristics are in fact quite rare, which is encouraging. However, this is no guarantee that NFL will not be applicable for a particular set of problems.

Proving whether or not a set of fitness functions $\mathcal{G}$ is closed under permutation may be difficult in general. However, there are cases where it is trivial to see that $\mathcal{G}$ cannot be closed under permutation via basic counting argument. For example, if $\mathcal{P}$ includes only one problem instance (which is sometimes the case in hyper-heuristic search), and the fitness function associated to that problem instance, f, is non-flat (i.e., there is at least one point $x \in \Omega$ with higher than average fitness, which is typically the case),

then $\mathcal{P}$ cannot be closed under permutation. This is because with f there is always a $y \in \Omega$, with $y \neq x$, such that $f(y) \neq f(x)$. It is, therefore, possible to find a way to shuffle f to produce a new fitness function $\tilde{f}$ in which the fitness originally associated to x is now assigned to y, that is $\tilde{f}(y) = f(x)$. But then, since $f(x) \neq f(y)$, we have that $\tilde{f}(y) \neq f(y)$, so $\tilde{f}$ is distinct from f. Since $\mathcal{P}$ is a singleton set, so is $\mathcal{G}$. However, $f \in \mathcal{G}$ and, so, $\tilde{f} \notin \mathcal{G}$, which implies $\mathcal{G}$ is not closed under permutation. So, *when a hyper-heuristic approach is applied to finding a solver for a specific problem, there can be a free lunch*.

This counting argument can be made more general. If we know that one problem has at least n distinct fitness values $f_1, \ldots, f_n$ at points $x_1, \ldots, x_n$, then it is possible to shuffle the assignments of fitness values to $x_1, \ldots, x_n$ in at least $n!$ ways. If our problem set $\mathcal{P}$ contains fewer that $n!$ problems, then it cannot be closed under permutation and there is a free lunch for hyper-heuristic search. It stands to reason that these conditions are easily met by any practical problem set, since a) the size, n, of the co-domain for realistic fitness functions will be relatively large (i.e., the fitness function can return a variety of different values), and b) practical problem sets can never be too large because one needs to calculate the performance of a in each of the problems which is usually done by running a on each problem thereby the time required for this procedure would limit the size of $\mathcal{P}$. So, the condition $|\mathcal{G}| < n!$ should easily be satisfied.[2] So, *in practice, when a hyper-heuristic approach is applied to finding a solver for a not-too-large set of problems, there will likely be a free lunch*.

5 What's a Free Lunch?

So far we have not looked very closely at what having a "free lunch" means. One might think, for example, that having a free lunch means that for any performance measure and any two algorithms, one algorithm is superior to the other. Or perhaps having a free lunch might mean that for any performance measure, there is at least one algorithm that is superior to some other. Unfortunately, neither interpretation is correct.

As we indicated above, [23] showed that two arbitrary algorithms have identical performance (irrespective of the chosen performance measure) over a set of functions if and only if that set is closed under permutation. This result was proven by designing a performance measure such that if NFL held over a set of functions $\mathcal{F}$ that is not closed under permutation, it would then be possible to identify an algorithm whose average performance over $\mathcal{F}$ is inferior to that of some other algorithm, leading to a contradiction. So, another way to look at this result is to say that if the set of functions $\mathcal{F}$ is *not* closed under permutation, then *there exists at least one performance measure under which at least one algorithm has better (worse) than average performance* at optimising functions from $\mathcal{F}$.

So, the fact that NFL does not apply to hyper-heuristic search when the set of fitness functions $\mathcal{G}$ associated to a problem set $\mathcal{P}$ is not closed under permutations means that

[2] In fact this condition is always satisfied in the case of Boolean induction problems with $k > 1$ inputs, since there can be at most 2^{2^k} different Boolean functions in $|\mathcal{G}|$ and, normally, there are $2^k + 1$ different possible fitness values, but $2^{2^k} < (2^k + 1)!$ for $k > 1$.

the search for superior search algorithms is meaningful, for at least some performance measures. We know nothing about which and how many algorithms exist that perform better/worse than average. We know nothing about which and how many performance measures exist for which this is possible. All we know is that answers to these questions exist. Finding them may be a totally different matter.

While this may sound discouraging, in fact testing whether any particular performance measure is suitable is not difficult. All we need to do is find two points in the search space for which the performance measure gives different outputs. We suspect in many practical situations this may just require testing a handful of points.

6 Implications for Computer Scientists and Other Searchers

Until now we have focused on the situation where we are evaluating one particular hyper-heuristic. As indicated in Figures 4–6, however, such a hyper-heuristic is just one element in the space of hyper-heuristics. In this section we want to move one level up in the "search hierarchy" to explore important questions such as: Are there better (non-resampling) hyper-heuristics than others? Are computer scientists designing hyper-heuristics wasting their time?

Let us start by considering a special (but typical) case. Let us imagine that we have fixed the set of problems $\mathcal{P}$ we want to solve and that we are interested in finding solvers for these problems that maximise $E[P(a)]$ for a given performance measure P. That is, we are not looking at all possible such measures, but just one particular P that we think works well for our objectives. For example, P might just be the success rate for an algorithm or the best fitness found in n fitness evaluations. As we noticed above, $E[P(a)]$ can be seen as a fitness function over the elements of Λ. We also noticed that if $\mathcal{G}$ is closed under permutation, such a fitness function is flat. Note that even if the set is not closed under permutation, the fitness function on Λ may look flat. It will not be flat only if, additionally, P is a suitable performance measure (i.e., one will show differences in performance).

If we enumerate Λ we can store the values of $E[P]$ associated to each heuristic in Λ in a vector h. If we fix every detail of our search (performance measure, set of problems, fitness measure over the solution space Ω, etc.), our hyper-heuristic is simply faced with one problem which is to find the maximum value in h. This is a very common situation in hyper-heuristics. The question is: which hyper-heuristic should one use for the job?

If $\mathcal{P}$ is not closed under permutation, and we have chosen a performance measure that will show differences in performance, then h is non-flat, and so there are better elements of Λ than others. Because of this, the singleton set $\{h\}$ is not closed under permutation (see argument in Section 4). As a result, there exists at least one hyper-heuristic that is better than average for at least one performance measure. In other words, if we consider problem sets that are not closed under permutation, looking for good hyper-heuristics is not a waste of time. Connecting this observation with the fact (highlighted in Section 4) that sets of problems which are closed under permutation may in practice be a rarity, we can conclude that *there may be a free lunch for computer scientist developing hyper-heuristics*.

While, clearly, not every user of hyper-heuristics is directly involved in developing new ones, the modification of hyper-heuristics and the tuning of their parameters is very

common. For example, users are likely to try to optimise the rates of application of the hyper-heuristic's search operators (e.g., with some sort of trial and error approach) in an attempt to obtain a hyper-heuristic that is able to locate elements of Λ associated with good h performance values using the smallest number of samples from Λ. This *tuning and tweaking is a form of search, where, we, the humans, are the searchers*. This process is guaranteed to be a waste of time if h is flat. However, as we noted above, this situation is unlikely and, so, *there may be a free lunch for users tuning hyper-heuristics*.

7 Higher-Order Hyper-Heuristics

Looking into the future a little, we may imagine that at a not-too-distant point, we will have enough computer power to automate the search for hyper-heuristics via hyper-hyper-heuristics. Is there hope for (future) hyper-hyper-heuristics? Does NFL hold for hyper-hyper-heuristics?

Clearly hyper-hyper-heuristics will search the same space currently searched by computer scientists who try to design good hyper-heuristics and users who tune their hyper-heuristics before using them. So, the search will be hopeless whenever human search is hopeless and *vice versa*. In other words, *in the right conditions*[3]*, there is a free lunch for hyper-hyper-heuristics*.

Looking further into the future, one may wonder what would happen if we considered hyper-hyper-$\cdots$-heuristics. The question is not purely theoretical, since hyper-hyper-heuristics are just round the corner, and researchers interested in designing them will in fact find themselves exploring the space of hyper-hyper-hyper-heuristics. Does it make sense to explore that space or does NFL hold there? As we have seen above, if the set of problems $\mathcal{P}$ is closed under permutation, then all heuristics are equally good, and, so, there is no point in using hyper-heuristics. The reason is that every time a hyper-heuristic samples the space of heuristics, it gets the same performance value back. Therefore, irrespective of what performance measure we may have in mind for hyper-heuristics, they will all appear to be equally good. That is the hyper-heuristics fitness landscape is flat too. It follows from this, that the same is true for higher order hyper-heuristics too. To sum up, *if we are working with a set of problems that is closed under permutation, there is no point in using hyper-hyper-$\cdots$-heuristics*.

The converse is not necessarily true though. That is, even if the set $\mathcal{P}$ is not closed under permutation, in order for hyper-hyper-$\cdots$-heuristics to make sense, we need to make sure at every level in the chain an objective measure is used that reveals performance differences at the level immediately below. While we suspect that in practice this may not be difficult to achieve (see discussion at the end of Section 5), there is no guarantee that such performance measures could be found without incurring in substantial overheads.

Can we imagine situations more general than this? Actually, yes, and in fact these may be quite important for the future of hyper-heuristics. Consider the case depicted in Figure 7 where a hyper-heuristic is asked to explore not one space of solvers with their associated performance values, but many such spaces (those in the rectangular region in the middle of the figure). For simplicity, let us imagine that these spaces share the

[3] See Section 4.

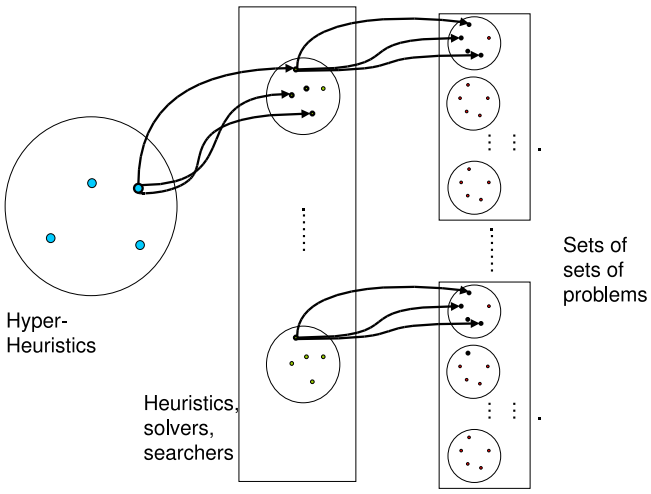

Fig. 7. A hyper-heuristic required to explore multiple spaces of search algorithms where each space has potentially a different set of problems associated to it

same structure and include the same algorithms/solvers but that they are characterised by different assignments of fitness (i.e., performance) to their points. In essence, from the point of view of the hyper-heuristic, these are different problems. Note that each heuristic space has potentially a different set of problems associated to it (one of the rectangles on the right of Figure 7).

Why would one want to consider this scenario? For example, we might want to evaluate how good our hyper-heuristic is at finding solvers for different classes of problems. In this case, each problem class induces a different h function over the space of heuristics, Λ. Another important case in where one is given only one set of problems (in this case all the rectangles on the right of Figure 7 would be identical), but we want to test whether our hyper-heuristic can work well with different performance measures P (e.g., best end of runs fitness, success rate, etc.). Then each performance measure would produce a different h over Λ. Alternatively, one might want the hyper-heuristic to work well both across problem classes and performance measures. Finally, one might even want to test a hyper-heuristic when searching different classes of heuristics (e.g., genetic algorithms, particle swarm optimisers and tabu search).

In all these cases, instead of having to deal with one fitness function over Λ, h, we will have a set of such functions, $\mathcal{H}$. In principle such a set could be closed under permutation. However, following arguments similar to the ones we have put forward for heuristics, it is easy to convince oneself that in many practical situations there will be a free lunch for this type of search and for its higher-order variants.

8 Conclusions

Hyper-heuristics are an important class of problem-solving techniques that find solvers for problems or classes of problems rather than attempting to solve the problems

themselves. The last few years have seen an increased effort and many successes in applying GP as a hyper-heuristic.

In this paper we have shown that NFL does not automatically apply to hyper-heuristic search. NFL is guaranteed to apply to hyper-heuristics and higher-order hyper-heuristics only if it applies to the lowest levels in the search hierarchy, i.e., if the set of problems of interest is closed under permutation. When this is not the case, and, as we have argued, this happens in many realistic situations, there is a free lunch for hyper-heuristic techniques provided that, at each level in the search hierarchy, heuristics are evaluated using performance measures that reveal the differences present at the level immediately below.

Naturally, the fact that NFL results may not hold for hyper-heuristic search does not imply that the existing hyper-heuristic methods are good. This may need to be proved by other means. The non-applicability of NFL, however, means that it is worthwhile to try to come up with new and more powerful hyper-heuristic algorithms, including those based on GP.

Having reviewed what this paper has achieved, let us also briefly reconsider what we have left out. Throughout the paper we have made three important assumptions (deriving directly from the original NFL theory). We discuss them below.

Firstly, we used expectations of performance measures $E[P(a)]$ to assess the performance of an algorithm across problems, of a hyper-heuristic across algorithms, of a hyper-hyper-heuristic across hyper-heuristics and so on. While this makes sense in many cases, in principle one could come up with different ways of judging the worth of a searcher over a set of searches. For example, one might be more interested in higher order statistics, tails of distributions or extreme values of performance. In this case, NFL cannot tell much about what happens in search. So, we don't have this nice tool to explore under what circumstances there may be a free lunch for hyper-heuristics. What remains true is that there still is no point in using hyper-heuristics if the aggregate performance measure (whatever it is) computed over a set of problems produces identical results for all search algorithms. This is because, in this case, NFL would apply to the higher levels of the search hierarchy, rendering the search for better hyper-heuristics and higher-order hyper-heuristics futile.

Secondly, we implicitly assumed that problems are drawn from problem sets either by deterministic enumeration or probabilistically by sampling $\mathcal{P}$ with uniform probability. However, there are cases where not all the problems in $\mathcal{P}$ are equally likely (or important). In these cases the expected performance of an algorithm over the problem set $\mathcal{P}$ is given by

$$E[P(a)] = \sum_{f \in \mathcal{G}} p(f) P(f,a) \tag{2}$$

where $p(f)$ is a function that indicates with which probability each problem is presented to a searcher. If $p(f)$ is uniform then, as we have seen in Section 2, NFL holds if and only if $\mathcal{G}$ is closed under permutation. The question is: what happens if $p(f)$ is non-uniform? As proved in [10], NFL may still hold, but rather stringent conditions must be met on the form of $p(f)$. A function and its permutations are characterised by a unique histogram of fitness values. For NFL to hold, $\mathcal{G}$ must be closed under permutation *and* all the functions in $\mathcal{G}$ with a particular fitness histogram must have identical probability of being drawn from $\mathcal{G}$. While it is clearly trivial to extend the theory presented in

this paper to cover the non-uniform case, these additional constraints make it even less likely that NFL will apply to any set of problems presented to a hyper-heuristic in a practical situation. So, hyper-heuristics have an even broader applicability on non-uniform problem domains.

Thirdly, we did not consider re-sampling (some algorithms are worse than others even in the presence of sets of problems that are closed under permutation simply because they waste more time revisiting points). In future research we will explore what implications re-sampling has for hyper-heuristic search.

References

1. Bader-El-Den, M., Poli, R.: Generating SAT local-search heuristics using a GP hyper-heuristic framework. In: Proceedings of Evolution Artificielle (October 2007)
2. Bader-El-Den, M.B., Poli, R.: A GP-based hyper-heuristic framework for evolving 3-SAT heuristics. In: Thierens, D., Beyer, H.-G., Bongard, J., Branke, J., Clark, J.A., Cliff, D., Congdon, C.B., Deb, K., Doerr, B., Kovacs, T., Kumar, S., Miller, J.F., Moore, J., Neumann, F., Pelikan, M., Poli, R., Sastry, K., Stanley, K.O., Stutzle, T., Watson, R.A., Wegener, I. (eds.) GECCO 2007: Proceedings of the 9th annual conference on Genetic and evolutionary computation, vol. 2, p. 1749. ACM Press, New York (2007)
3. Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D.: Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications. Morgan Kaufmann, San Francisco (1998)
4. Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., Schulenburg, S.: Hyper-heuristics: an emerging direction in modern search technology. In: Glover, F.W., Kochenberger, G.A. (eds.) Handbook of metaheuristics, ch. 16, pp. 457–474. Kluwer Academic Publishers, Boston (2003)
5. Burke, E.K., Hyde, M.R., Kendall, G.: Evolving bin packing heuristics with genetic programming. In: Runarsson, T.P., Beyer, H.-G., Burke, E.K., Merelo-Guervós, J.J., Whitley, L.D., Yao, X. (eds.) PPSN 2006. LNCS, vol. 4193, pp. 860–869. Springer, Heidelberg (2006)
6. Burke, E.K., Hyde, M.R., Kendall, G., Woodward, J.: Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one. In: Thierens, D., Beyer, H.-G., Bongard, J., Branke, J., Clark, J.A., Cliff, D., Congdon, C.B., Deb, K., Doerr, B., Kovacs, T., Kumar, S., Miller, J.F., Moore, J., Neumann, F., Pelikan, M., Poli, R., Sastry, K., Stanley, K.O., Stutzle, T., Watson, R.A., Wegener, I. (eds.) GECCO 2007: Proceedings of the 9th annual conference on Genetic and evolutionary computation, vol. 2, pp. 1559–1565. ACM Press, New York (2007)
7. Di Chio, C., Poli, R., Langdon, W.B.: Evolution of force-generating equations for PSO using GP. In: Manzoni, S., Palmonari, M., Sartori, F. (eds.) AI*IA Workshop on Evolutionary Computation, Evoluzionistico GSICE 2005, University of Milan Bicocca, Italy, September 20 (2005)
8. Fukunaga, A.: Automated discovery of composite SAT variable selection heuristics. In: Proceedings of the National Conference on Artificial Intelligence (AAAI), pp. 641–648 (2002)
9. Igel, C., Toussaint, M.: On classes of functions for which no free lunch results hold. Information Processing Letters 86(6), 317–321 (2003)
10. Igel, C., Toussaint, M.: A no-free-lunch theorem for non-uniform distributions of target functions. Journal of Mathematical Modelling and Algorithms 3, 313–322 (2004)
11. Keller, R.E., Poli, R.: Cost-benefit investigation of a genetic-programming hyperheuristic. In: Monmarché, N., Talbi, E.-G., Collet, P., Schoenauer, M., Lutton, E. (eds.) EA 2007. LNCS, vol. 4926, pp. 13–24. Springer, Heidelberg (2008)

12. Keller, R.E., Poli, R.: Linear genetic programming of metaheuristics. In: Thierens, D., Beyer, H.-G., Bongard, J., Branke, J., Clark, J.A., Cliff, D., Congdon, C.B., Deb, K., Doerr, B., Kovacs, T., Kumar, S., Miller, J.F., Moore, J., Neumann, F., Pelikan, M., Poli, R., Sastry, K., Stanley, K.O., Stutzle, T., Watson, R.A., Wegener, I. (eds.) GECCO 2007: Proceedings of the 9th annual conference on Genetic and evolutionary computation, p. 1753. ACM Press, New York (2007)
13. Keller, R.E., Poli, R.: Linear genetic programming of parsimonious metaheuristics. In: Proceedings of IEEE Congress on Evolutionary Computation (CEC) (September 2007)
14. Kibria, R.H., Li, Y.: Optimizing the initialization of dynamic decision heuristics in DPLL SAT solvers using genetic programming. In: Collet, P., Tomassini, M., Ebner, M., Gustafson, S., Ekárt, A. (eds.) EuroGP 2006. LNCS, vol. 3905, pp. 331–340. Springer, Heidelberg (2006)
15. Koza, J.R.: Genetic Programming: On the Programming of Computers by Natural Selection. MIT Press, Cambridge (1992)
16. Oltean, M.: Evolving evolutionary algorithms for function optimization. In: Chen, K. (ed.) The 7th Joint Conference on Information Sciences, September 2003, vol. 1, pp. 295–298. Association for Intelligent Machinery (2003)
17. Oltean, M.: Evolving evolutionary algorithms using linear genetic programming. Evolutionary Computation 13(3), 387–410 (Fall 2005)
18. Oltean, M., Dumitrescu, D.: Evolving TSP heuristics using multi expression programming. In: Bubak, M., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2004. LNCS, vol. 3037, pp. 670–673. Springer, Heidelberg (2004)
19. Poli, R., Di Chio, C., Langdon, W.B.: Exploring extended particle swarms: a genetic programming approach. In: Beyer, H.-G., O'Reilly, U.-M., Arnold, D.V., Banzhaf, W., Blum, C., Bonabeau, E.W., Cantu-Paz, E., Dasgupta, D., Deb, K., Foster, J.A., de Jong, E.D., Lipson, H., Llora, X., Mancoridis, S., Pelikan, M., Raidl, G.R., Soule, T., Tyrrell, A.M., Watson, J.-P., Zitzler, E. (eds.) GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation, vol. 1, pp. 169–176. ACM Press, New York (2005)
20. Poli, R., Langdon, W.B., Holland, O.: Extending particle swarm optimisation via genetic programming. In: Keijzer, M., Tettamanzi, A.G.B., Collet, P., van Hemert, J., Tomassini, M. (eds.) EuroGP 2005. LNCS, vol. 3447, pp. 291–300. Springer, Heidelberg (2005)
21. Poli, R., Langdon, W.B., McPhee, N.F.: A field guide to genetic programming (2008), `http://lulu.com` `http://www.gp-field-guide.org.uk` (With contributions by Koza, J.R.)
22. Poli, R., Woodward, J., Burke, E.K.: A histogram-matching approach to the evolution of bin-packing strategies. In: Proceedings of the IEEE Congress on Evolutionary Computation, Singapore (2007) (accepted)
23. Schumacher, C., Vose, M.D., Whitley, L.D.: The no free lunch and problem description length. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pp. 565–570. Morgan Kaufmann, San Francisco (2001)
24. Valsecchi, A., Vanneschi, L.: A study of some implications of the no free lunch theorem. In: Giacobini, M., Brabazon, A., Cagnoni, S., Di Caro, G.A., Drechsler, R., Ekárt, A., Esparcia-Alcázar, A.I., Farooq, M., Fink, A., McCormack, J., O'Neill, M., Romero, J., Rothlauf, F., Squillero, G., Uyar, A.Ş., Yang, S. (eds.) EvoWorkshops 2008. LNCS, vol. 4974, pp. 633–642. Springer, Heidelberg (2008)
25. Whitley, D., Watson, J.P.: Complexity theory and the no free lunch theorem. In: Burke, E.K., Kendall, G. (eds.) Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, ch. 11, pp. 317–339. Springer, US (2005)
26. Wolpert, D., Macready, W.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation 1(1), 67–82 (1997)