A Study of Some Implications of the No Free Lunch Theorem

Andrea Valsecchi and Leonardo Vanneschi

Department of Informatics, Systems and Communication (D.I.S.Co.)
University of Milano-Bicocca, Milan, Italy
a.valsecchi8@campus.unimib.it
vanneschi@disco.unimib.it

Abstract. We introduce the concept of "minimal" search algorithm for a set of functions to optimize. We investigate the structure of closed under permutation (c.u.p.) sets and we calculate the performance of an algorithm applied to them. We prove that each set of functions based on the distance to a given optimal solution, among which trap functions, onemax or the recently introduced onemix functions, and the NK-landscapes are not c.u.p. and thus the thesis of the sharpened No Free Lunch Theorem does not hold for them. Thus, it makes sense to look for a specific algorithm for those sets. Finally, we propose a method to build a "good" (although not necessarily minimal) search algorithm for a specific given set of problems. The algorithms produced with this technique show better average performance than a genetic algorithm executed on the same set of problems, which was expected given that those algorithms are problem-specific. Nevertheless, in general they cannot be applied for real-life problems, given their high computational complexity that we have been able to estimate.

1 Introduction

The No Free Lunch (NFL) theorem states that all non-repeating search algorithms, if tested on all possible cost functions, have the same average performance [1]. As a consequence, one may informally say that looking for a search algorithm that outperforms all the others on all the possible optimization problems is hopeless. But there are sets of functions for which the thesis of the NFL theorem does not hold, and thus talking about a "good" (or even the "best") algorithm on those sets of functions makes sense. In this paper, we investigate some of those sets of functions, we define for the first time the concept of minimal algorithm and we investigate some of its properties.

The sharpened-NFL [2] states that the thesis of the NFL theorem holds for a set of functions F if and only if F is *closed under permutation* (c.u.p.). For these particular sets of functions calculating the performance of an algorithm is relatively simple. For instance, for c.u.p. sets of functions with a constant number of globally optimal solutions, a method to estimate the average performance of an algorithm has been presented in [3]. In this paper, we try to generalize this result and to give for the first time an equation to calculate the average performance of algorithms for any c.u.p. set of functions.

To prove that a set of cost functions is *not* c.u.p., it is possible to use some properties of c.u.p. sets. For instance, over the functions that belong to c.u.p. sets, it is not

M. Giacobini et al. (Eds.): EvoWorkshops 2008, LNCS 4974, pp. 633–642, 2008.

possible to define a non-trivial neighborhood structure of a specific type, as explained in [4]. Furthermore, a set of functions with description length sufficiently bounded is not c.u.p. [5]. In this paper, we prove that some particular sets of problems, which are typically used as benchmarks for experimental or theoretical optimization studies, are *not* c.u.p. In particular, we focus on problems for which the fitness (or cost) of the solutions is a function of the distance to a given optimum. These problems include, for instance, trap functions [6], onemax and the recently defined onemix [7] functions. We also consider the NK-landscapes. As a consequence of the fact that these sets of functions are not c.u.p., and thus the NFL does not hold for them, we could informally say that it makes sense to look for a "good" (or even the "best") optimization algorithm for them. In this paper, we present a method to build a "good" (although not necessarily minimal) search algorithm for a given set of problems and we apply it to some sets of trap functions and NK-landscapes. Those algorithms are experimentally compared with a standard Genetic Algorithm (GA) on the same sets of functions.

This paper is structured as follows: in Section 2 we briefly recall the NFL theorem. In Section 3 we define the concept of minimal search algorithm and we prove some properties of its performance; furthermore, we give an equation to estimate the performance of an algorithm applied to any set of c.u.p. functions. In section 4 we prove that each set of functions based on the distance to an optimal solution is not c.u.p.; successively, we prove the same property for NK-landscapes. In section 5 we present a method to automatically generate an optimization algorithm specialized for a given set of functions. In section 6, the performance of some of the algorithms generated by our method are compared with the ones of a GA. Finally, in section 7 we offer our conclusions and discuss possible future research activities.

2 No Free Lunch Theorem

Let $\mathcal{X}, \mathcal{Y}$ be two finite sets and let $f : \mathcal{X} \to \mathcal{Y}$. We call *trace* of length m over $\mathcal{X}$ and $\mathcal{Y}$ a sequence of couples: $t = \langle (x_1, y_1), \cdots, (x_m, y_m) \rangle$ such that $x_i \in \mathcal{X}$ and $y_i = f(x_i), \forall i = 1, 2, \cdots, m$. If we interpret $\mathcal{X}$ as the space of all possible solutions of an optimization problem (search space), $f(.)$ as the fitness (or cost) function and $\mathcal{Y}$ as the set of all possible fitness values, a trace t can be interpreted as a sequence of points visited by a search algorithm along with their fitness values.

We call *simple* a trace t such that: $t[i] = t[j] \Rightarrow i = j$, i.e. a trace in which each solution appears only once. Let T be the set of all possible traces over the sets $\mathcal{X}$ and $\mathcal{Y}$. We call *search operator* a function $g : T \to \mathcal{X}$. A search operator can be interpreted as a function that given a trace representing all the solutions visited by a search algorithm until the current instant (along with their fitness values) returns the solution that will be visited at the next step. We say that g is *non-repeating*[1] if $\forall t \in T, g(t) \notin t^{\mathcal{X}}$. We can observe that, if $t \in T$ is simple and g is non-repeating, then also $t' = t \parallel (g(t), f \circ g(t))$ where $\parallel$ is the concatenation operator is a simple trace.

A deterministic search algorithm A_g is an application $A_g : ((\mathcal{X} \to \mathcal{Y}) \times T) \to T$ with $\forall t \in T \quad A_g(f,t) = t \parallel (g(t), f \circ g(t))$ where g is a search operator. We say that A_g is

[1] Given a trace $t = \langle t_1, t_2, ..., t_m \rangle$, with $\forall i = 1, 2, ..., n : t_i = (x_i, y_i)$, we use the notation $t^{\mathcal{X}}$ to indicate the set $\{x_1, x_2, ..., x_m\}$. and the notation $t^{\mathcal{Y}}$ to indicate the set $\{y_1, y_2, ..., y_m\}$.