Theoretical Computer Science 545 (2014) 76–93

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Exact computation of the expectation surfaces for uniform crossover along with bit-flip mutation

Francisco Chicano [a,*], Darrell Whitley [b], Enrique Alba [a]

[a] Dept. of Lenguajes y Ciencias de la Computación, University of Málaga, 29071 Málaga, Spain
[b] Computer Science Dept., Colorado State University, Fort Collins, CO 80523, USA

ARTICLE INFO

Keywords:
Uniform crossover
Bit-flip mutation
Walsh decomposition
Landscape theory
Fitness landscapes

ABSTRACT

Uniform crossover and bit-flip mutation are two popular operators used in genetic algorithms to generate new solutions in an iteration of the algorithm when the solutions are represented by binary strings. We use the Walsh decomposition of pseudo-Boolean functions and properties of Krawtchouk matrices to exactly compute the expected value for the fitness of a child generated by uniform crossover followed by bit-flip mutation from two parent solutions. We prove that this expectation is a polynomial in ρ, the probability of selecting the best-parent bit in the crossover, and μ, the probability of flipping a bit in the mutation. We provide efficient algorithms to compute this polynomial for Onemax and MAX-SAT problems, but the results also hold for other problems such as NK-Landscapes. We also analyze the features of the expectation surfaces.

1. Introduction

In evolutionary computation, the classical variation operators are crossover and mutation. Crossover takes two solutions as input and generates one or more solutions combining the original ones. Mutation takes one solution and introduces a small change on it. These operators are usually run in sequence: crossover comes first and then mutation is applied to the output solution of crossover. The operators act on the genotype, that is, on the representation of the solutions. Thus, they are linked to the solution representation. Many different crossover and mutation operators have been defined for different representations in the literature. In the case of the binary strings some examples are 1-point crossover, 2-point crossover, uniform crossover, bit-flip mutation and 1-bit-flip mutation [1].

In particular, Uniform Crossover (UX) builds a new solution by randomly selecting each "allele" from one of the parent solutions. The "allele" in the best of the two parents is selected with probability ρ, which is called the *bias*. A common value for this bias is $\rho = 0.5$, where each parent has the same probability of providing its "allele" to the offspring. The so-called bit-flip mutation (BF) flips each individual bit of the binary string with a probability μ. In the literature it is common to use $\mu = 1/n$, which flips one bit per solution on average. From the point of view of runtime analysis, it has been proven that the value $\mu = c/n$ where c is a constant is optimal if we want to minimize the number of iterations to reach the global optimum on a linear pseudo-Boolean function using a $(1+1)$ Evolutionary Algorithm (EA) [2].

In this work we extend the results in [3] by providing a closed-form expression for the expected value of the objective function evaluated in a solution that is the result of applying UX and BF to two parent solutions. This expected value is a function of ρ and μ and, for this reason, we also call it *expectation surface*. The previous work [3] only considered UX. With the results presented in this paper we combine UX and BF and also prove in a different form the results in [4,5] for

* Corresponding author.
E-mail addresses: chicano@lcc.uma.es (F. Chicano), whitley@cs.colostate.edu (D. Whitley), eat@lcc.uma.es (E. Alba).

0304-3975/$ – see front matter
http://dx.doi.org/10.1016/j.tcs.2014.01.002

the expectation after BF. We also extend the formula to compute higher order moments, and not only the expected fitness value.

From a theoretical point of view, the closed-form formulas could help to understand the behavior of the two operators acting together. From a practical point of view, they could be used to develop new search strategies or operators. In particular, using for example the expected value and the standard deviation we can compute some bounds for the fitness values that the algorithm can find with a high probability after applying the operators [6]. Using this approach it would be possible to try different values for ρ and μ before applying the operators in order to improve the chances of finding good solutions.

The remainder of the paper is organized as follows. In the next section the mathematical tools required to understand the rest of the paper are presented. In Section 3 we present our main contribution of this work: the expected fitness value of the solution generated by UX and BF. We discuss in that section how the formula can be used to compute higher order moments (other than the expectation). Section 4 provides closed-form formulas for the expression of the expected fitness value in the case of the Onemax and MAX-SAT problems. In Section 5 we analyze the features of the expectation surface and provide a procedure to build an instance of weighted MAX-SAT having the desired surface. Finally, Section 6 presents the conclusions and future work.

2. Background

Let us first clarify the notation used for binary strings. We write here $\mathbb{B}^n$ as a synonym of $\mathbb{Z}_2^n$: $\mathbb{Z}_2^n = \mathbb{B}^n$. This set forms an Abelian group with the component-wise sum in $\mathbb{Z}_2$ (exclusive OR), denoted with $\oplus$. Given an element $z \in \mathbb{B}^n$, we will denote with $|z|$ the number of ones of z and with $\bar{z}$ the complement of the string (all bits inverted). Given a set of binary strings W and a binary string u we denote with $W \wedge u$ the set of binary strings that can be computed as the bitwise AND of a string in W and u, that is, $W \wedge u = \{w \wedge u \mid w \in W\}$. For example, $\mathbb{B}^4 \wedge 0101 = \{0000, 0001, 0100, 0101\}$. Given a string $w \in \mathbb{B}^n$, the set $\mathbb{B}^n \wedge w$ defines a hyperplane in $\mathbb{B}^n$ [7]. We will denote with $\underline{i}$ the binary string with position i set to 1 and the rest set to 0. We omit the length of the string n in the notation, but it will be clear from the context. For example, if we are considering binary strings in $\mathbb{B}^4$ we have $\underline{1} = 1000$ and $\underline{3} = 0010$. We will denote with $\vee$ the bitwise OR operator between two binary strings. We will assume that the $\wedge$ operator has precedence over $\vee$. However, we will use parentheses to clarify the precedence.

Definition 1. We define a *pseudo-Boolean* function f as a map between $\mathbb{B}^n$, the set of binary strings of length n, and $\mathbb{R}$, the set of real numbers.

Let us consider the set of all the pseudo-Boolean functions defined over $\mathbb{B}^n$, $\mathbb{R}^{\mathbb{B}^n}$. We can think of a pseudo-Boolean function as an array of 2^n real numbers, each one being the function evaluation of a particular binary string of $\mathbb{B}^n$. Each pseudo-Boolean function is, thus, a particular vector in a vector space with 2^n dimensions. Let us define the dot-product between two pseudo-Boolean functions as:

$$\langle f, g \rangle = \sum_{x \in \mathbb{B}^n} f(x) g(x).$$

Now we introduce a set of functions that will be relevant for our purposes in the next sections: the *Walsh functions* [8].

Definition 2. The (non-normalized) Walsh function with parameter $w \in \mathbb{B}^n$ is a pseudo-Boolean function defined over $\mathbb{B}^n$ as:

$$\psi_w(x) = \prod_{i=1}^{n} (-1)^{w_i x_i} = (-1)^{|w \wedge x|} = (-1)^{\sum_{i=1}^{n} w_i x_i}, \tag{1}$$

where the subindex in w_i and x_i denotes the i-th component of the binary strings w and x, respectively.

We can observe that the Walsh functions map $\mathbb{B}^n$ to the set $\{-1, 1\}$. We define the *order* of a Walsh function ψ_w as the value $|w|$. Some properties of the Walsh functions are given in the following proposition. The proof can be found in Vose [9].

Proposition 1. *Let us consider the Walsh functions defined over $\mathbb{B}^n$. The following identities hold*:

$$\psi_0 = 1, \tag{2}$$

$$\psi_{w \oplus t} = \psi_w \psi_t, \tag{3}$$

$$\psi_w(x \oplus y) = \psi_w(x) \psi_w(y), \tag{4}$$

$$\psi_w(x) = \psi_x(w), \tag{5}$$