Fitness Probability Distribution of Bit-Flip Mutation

Francisco Chicano chicano@lcc.uma.es
Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, Spain

Andrew M. Sutton andrew.sutton@uni-jena.de
Fakultät für Mathematik und Informatik, Friedrich-Schiller-Universität Jena, Germany

L. Darrell Whitley whitley@cs.colostate.edu
Department of Computer Science Colorado State University, USA

Enrique Alba eat@lcc.uma.es
Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, Spain

doi:10.1162/EVCO_a_00130

Abstract

Bit-flip mutation is a common mutation operator for evolutionary algorithms applied to optimize functions over binary strings. In this paper, we develop results from the theory of landscapes and Krawtchouk polynomials to exactly compute the probability distribution of fitness values of a binary string undergoing uniform bit-flip mutation. We prove that this probability distribution can be expressed as a polynomial in p, the probability of flipping each bit. We analyze these polynomials and provide closed-form expressions for an easy linear problem (Onemax), and an NP-hard problem, MAX-SAT. We also discuss a connection of the results with runtime analysis.

Keywords

Bit-flip mutation, evolutionary algorithms, landscape theory, combinatorial optimization, randomized algorithms.

1 Introduction

Evolutionary algorithms that operate on binary string representations commonly employ the *bit-flip* mutation operator. This operator acts independently on each bit in a solution and changes the value of the bit (from 0 to 1 and vice versa) with probability p, where p is a parameter of the operator. The most commonly recommended value for this parameter is $p = 1/n$ where n is the length of the binary string. For linear functions optimized with a (1 + 1) EA, this rate is provably optimal (Witt, 2013). However, in the general case, very little is currently understood about the mutation operator and its influence on the optimization process.

In this paper we study the operator from the point of view of landscape theory. Using this approach, we provide closed-form formulas for the fitness probability distribution of the solutions obtained after the application of bit-flip mutation to a particular solution. To the best of our knowledge, such general and closed-form formulas have not been presented before. We can find, however, some works in which mathematical expressions are provided for this probability distribution in the case of particular problems like

Manuscript received: March 12, 2013; revised: August 27, 2013 and March 28, 2014; accepted: May 13, 2014.

Onemax (Garnier et al., 1999). In this paper we want to be more general and provide a mathematical expression for the probability distribution that separates two elements: the mathematical entity related to the problem $\boldsymbol{F}$, and another one related to the operator $\boldsymbol{\Lambda}$. This approach yields a general framework that provides an expression that is valid for any problem as far as we can provide the problem-dependent entity $\boldsymbol{F}$.

Sutton et al. (2011b) and Chicano and Alba (2011) used landscape theory to provide a closed-form formula for the expectation after a bit-flip mutation (we repeat this result in Section 3.1). In this work we generalize these results and the one by Sutton et al. (2011a), providing closed-form formulas to compute the fitness probability distribution. The new result provides a deeper understanding of the behavior of the mutation operator. We illustrate the approach by providing concrete expressions for the moments of the probability distribution for two well-known problems: Onemax and MAX-SAT.

Many results coming from the field of fitness landscape analysis provide exact results for the expected fitness of solutions undergoing uniform transformations by evolutionary operators. However, they often cannot say anything about selection operators because the framework does not easily handle the probability of obtaining an improving mutation. This quantity is much harder to derive because it always depends on some instance-dependent structure that is ignored in such analyses. In this paper, we work out how to address the problem of selection by separating the instance-dependent structure from the instance-independent structure. This separation allows us to derive initial results on the probability of producing an improving offspring. As a consequence, we illustrate a way to use the theory of landscapes to derive the expected runtime of a $(1+\lambda)$ EA without crossover on Onemax.

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 landscape analysis of bit-flip mutation and the closed-form formulas for the fitness probability distribution. Section 4 provides particular results for two well-known problems in the domain of combinatorial optimization: Onemax and MAX-SAT. Section 5 proves the connection between the results in this paper and the runtime analysis of a $(1+\lambda)$ EA. Finally, Section 6 presents the conclusions and future work.

2 Background

In this section we present some fundamental results of landscape theory. We will only focus on the relevant information required to understand the rest of the paper. We refer the reader interested in a deeper exposition of this topic to the survey by Reidys and Stadler (2002).

A *landscape* for a combinatorial optimization problem is a triple (X, N, f), where X is a finite or countable solution set, $f : X \to \mathbb{R}$ defines the objective function and N is a *neighborhood function* that maps any solution $x \in X$ to the set $N(x)$ of points reachable from x. If $y \in N(x)$ then y is a neighbor of x. The pair (X, N) is called *configuration space* and can be represented using a graph $G = (X, E)$ in which X is the set of vertices and a directed edge (x, y) exists in E if $y \in N(x)$ (Biyikoglu et al., 2007). We can represent the neighborhood operator by its adjacency matrix

$$\boldsymbol{A}_{x,y} = \begin{cases} 1 & \text{if } y \in N(x), \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Any discrete function, f, defined over the set of candidate solutions can be characterized as a vector in $\mathbb{R}^{|X|}$. Any $|X| \times |X|$ matrix can be interpreted as a linear map that

acts on vectors in $\mathbb{R}^{|X|}$. For example, the adjacency matrix $\boldsymbol{A}$ acts on function f as follows

$$\boldsymbol{A}f = \begin{pmatrix} \sum_{y\in N(x_1)} f(y) \\ \sum_{y\in N(x_2)} f(y) \\ \vdots \\ \sum_{y\in N(x_{|X|})} f(y) \end{pmatrix}. \tag{2}$$

The component x of this matrix-vector product can thus be written as:

$$(\boldsymbol{A}f)(x) = \sum_{y\in N(x)} f(y), \tag{3}$$

which is the sum of the function value of all the neighbors of x. In the case of binary strings, the minimal-change neighborhood at a point x is the set of Hamming neighbors of x. The Hamming neighborhood induces a regular, connected graph $G = (X, E)$, meaning that G is connected and $|N(x)| = d > 0$ for a constant d, for all $x \in X$. When a neighborhood is regular, the so-called *Laplacian matrix* is defined as $\boldsymbol{\Delta} = \boldsymbol{A} - d\boldsymbol{I}$. This corresponds to the Laplacian of the graph G. Stadler (1995) defines the class of *elementary landscapes* where the function f is an eigenvector (or eigenfunction) of the Laplacian up to an additive constant. Formally, we have the following.

DEFINITION 1: *Let (X, N, f) be a landscape and $\boldsymbol{\Delta}$ be the Laplacian matrix of the configuration space graph. The landscape is said to be elementary if there exists a constant b that we call the offset, and an eigenvalue λ of $-\boldsymbol{\Delta}$ such that $(-\boldsymbol{\Delta})(f - b) = \lambda(f - b)$.*

We use eigenvalues of $-\boldsymbol{\Delta}$ instead of $\boldsymbol{\Delta}$ to have positive eigenvalues (Biyikoglu et al., 2007). In connected neighborhoods such as the Hamming neighborhood, the offset b is the average value of the function f evaluated over the entire search space: $b = \bar{f}$. In elementary landscapes, the average value $\bar{f}$ can usually be computed in a very efficient way using the problem data. That is, it is not required to do a complete enumeration over the search space. For a concrete example of the TSP the reader is referred to Whitley et al. (2008).

Suppose (X, N, f) is elementary with eigenvalue λ. For any scalars a and b, define the function $g : X \to \mathbb{R}$ as $g(x) = af(x) + b$. Clearly, (X, N, g) is also elementary with the same eigenvalue λ. Furthermore, in regular neighborhoods, if g is an eigenfunction of $-\boldsymbol{\Delta}$ with eigenvalue λ then g is also an eigenfunction of $\boldsymbol{A}$ (the adjacency matrix of the configuration space graph G) with eigenvalue $d - \lambda$. The average value of the fitness function in the neighborhood of a solution can be computed using the expression:

$$\operatorname*{avg}_{y\in N(x)}\{f(y)\} = \frac{1}{d}(\boldsymbol{A}f)(x). \tag{4}$$

If (X, N, f) is elementary with eigenvalue λ, then the average over the neighborhood is computed as:

$$\begin{aligned} \operatorname*{avg}_{y\in N(x)}\{f(y)\} &= \operatorname*{avg}_{y\in N(x)}\{f(y) - \bar{f}\} + \bar{f} \\ &= \frac{1}{d}(\boldsymbol{A}(f - \bar{f}))(x) + \bar{f} = \frac{d-\lambda}{d}(f(x) - \bar{f}) + \bar{f} \\ &= f(x) + \frac{\lambda}{d}(\bar{f} - f(x)), \end{aligned} \tag{5}$$

which is sometimes referred to as Grover's wave equation (Grover, 1992). In the previous expression we used the fact that $f - \bar{f}$ is an eigenfunction of $\boldsymbol{A}$ with eigenvalue $d - \lambda$.

The wave equation makes it possible to compute the average value of the fitness function f evaluated over all of the neighbors of x using only the value $f(x)$. The previous average can be interpreted as the expected value of the objective function when a random neighbor of x is selected using a uniform distribution. This is exactly the behavior of the so-called *1-bit-flip* mutation (Garnier et al., 1999).

A landscape (X, N, f) is not always elementary, but even in this case it is possible to characterize the function f as the sum of elementary landscapes, called *elementary components* of the landscape. The interested reader can find examples of elementary landscapes in Whitley et al. (2008) and Whitley and Sutton (2009), and can find more on the elementary landscape decomposition in Chicano et al. (2011).

2.1 Binary Hypercube

The previous definitions are general concepts of landscape theory. Let us focus now on the binary configuration spaces with the Hamming neighborhood, the so-called *binary hypercubes*, which are the configuration spaces we need in the analysis of bit-flip mutation. Let us first present the notation. In these spaces the solution set X is the set of all binary strings of size n, formally, $\mathbb{Z}_2^n = \mathbb{B}^n$. The solution set forms an Abelian group with the componentwise 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. 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, $W \wedge u = \{w \wedge u | w \in W\}$. For example, $\mathbb{B}^4 \wedge 0101 = \{0000, 0001, 0100, 0101\}$. We will denote with $\underline{i}$ the binary string with position i set to 1 (starting from the leftmost position) 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 $\underline{3} = 0010$.

It is convenient to characterize the neighborhood by a set of group elements $S = \{s_1, s_2, \ldots, s_d\}$ that generate the entire group. Here S is called a *generating set*. The neighborhood of a solution x is just the set $N(x) = x \oplus S = \{x \oplus s | s \in S\}$. In the binary hypercube, two solutions x and y are neighbors if one can be obtained from the other by flipping a single bit, that is, if the Hamming distance between the solutions, $|x \oplus y|$, is 1. Thus, the generating set is composed of every binary string with a single 1: $S_1 = \{s \in \mathbb{B}^n \mid |s| = 1\}$.

We define the sphere of radius r around a solution x as the set of all solutions at Hamming distance r from x (Sutton et al., 2010). We are also interested in these spheres since the probability of reaching a solution y from a solution x using the bit-flip mutation operator is the same for all the solutions in a sphere around x. Now we can observe that the solutions in a sphere of radius r around x can be thought as the neighborhood N_r of x generated by an appropriate generating set S_r. The generating set is composed of all the solutions having exactly r ones: $S_r = \{s \in \mathbb{B}^n \mid |s| = r\}$. The notation S_1 used before was selected to be a particular case of this more general neighborhood. We will use the notation $N_r(x) = x \oplus S_r$. Another particular case is the one of $S_0 = \{0\}$ that generates the identity neighborhood $N_0(x) = \{x\}$. Each neighborhood has its corresponding adjacency matrix denoted with $\boldsymbol{A}^{(r)}$.

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 one 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:

$$\langle f, g \rangle = \sum_{x \in \mathbb{B}^n} f(x)g(x). \tag{6}$$

Now we introduce a set of functions that will be relevant for our purposes in the next sections: the *Walsh functions* (Walsh, 1923).

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:*

$$\psi_w(x) = \prod_{i=1}^{n} (-1)^{w_i x_i} = (-1)^{\sum_{i=1}^{n} w_i x_i}, \tag{7}$$

where the subindex in w_i and x_i denotes the ith 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. A proof of these properties can be found in Vose (1999).

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

$$\psi_0 = 1, \tag{8}$$

$$\psi_{w \oplus t} = \psi_w \psi_t, \tag{9}$$

$$\psi_w(x \oplus y) = \psi_w(x)\psi_w(y), \tag{10}$$

$$\psi_w(x) = \psi_x(w), \tag{11}$$

$$\psi_w^2 = 1, \tag{12}$$

$$\sum_{x \in \mathbb{B}^n} \psi_w(x) = 2^n \delta_0^{|w|} = \begin{cases} 2^n & \text{if } w = 0, \\ 0 & \text{if } w \neq 0, \end{cases} \tag{13}$$

$$\psi_{\underline{i}}(x) = (-1)^{x_i} = 1 - 2x_i, \tag{14}$$

$$\langle \psi_w, \psi_t \rangle = 2^n \delta_w^t, \tag{15}$$

where δ denotes the Kronecker delta.

There exist 2^n Walsh functions in $\mathbb{B}^n$, and according to Equation (15) they are orthogonal, so they form a basis of the set of pseudo-Boolean functions. Any arbitrary pseudo-Boolean function f can be expressed as a weighted sum of Walsh functions. We can represent f in the Walsh basis in the following way:

$$f(x) = \sum_{w \in \mathbb{B}^n} a_w \psi_w(x), \tag{16}$$

where the *Walsh coefficients* a_w are defined as:

$$a_w = \frac{1}{2^n} \langle \psi_w, f \rangle . \tag{17}$$

The previous expression is called the *Walsh expansion* (or decomposition) of f. The interested reader can refer to the text by Terras (1999) for a deeper treatment of Walsh functions and their properties.

The reason why Walsh functions are so important for the mutation analysis is because they are eigenvectors of the adjacency matrices $\boldsymbol{A}^{(r)}$ defined above, as the next proposition proves.

PROPOSITION 2: *In $\mathbb{B}^n$, the Walsh function ψ_w defined in Equation (7) is an eigenvector of the adjacency matrix $\boldsymbol{A}^{(r)}$ based on the generating set S_r (sphere of radius r) with eigenvalue*

$$\sum_{s\in S_r}\psi_w(s) = \mathcal{K}^{(n)}_{r,|w|}, \tag{18}$$

where $\mathcal{K}^{(n)}_{r,j}$ is the (r, j) element of the so-called nth order Krawtchouk matrix $\mathcal{K}^{(n)}$, defined as:

$$\mathcal{K}^{(n)}_{r,j} = \sum_{l=0}^{n}(-1)^l \binom{n-j}{r-l}\binom{j}{l}, \tag{19}$$

for $0 \le r, j \le n$. We assume in the previous expression that $\binom{a}{b} = 0$ if $b > a$ or $b < 0$.

PROOF: The Walsh function ψ_w is an eigenvector of $\boldsymbol{A}^{(r)}$ if $\boldsymbol{A}^{(r)}\psi_w = \lambda\psi_w$ for some constant λ, which is the eigenvalue. Taking into account the definition of neighborhood based on the generating set S_r we can write:

$$\left(\boldsymbol{A}^{(r)}\psi_w\right)(x) = \sum_{s\in S_r}\psi_w(x\oplus s) = \sum_{s\in S_r}\psi_w(x)\psi_w(s) = \left(\sum_{s\in S_r}\psi_w(s)\right)\psi_w(x),$$

where we used the property in Equation (10) and we can identify the eigenvalue with the left-hand side of Equation (18). Let us now prove that this value is exactly $\mathcal{K}^{(n)}_{r,|w|}$. Using the definition of S_r we can write the series as:

$$\sum_{s\in S_r}\psi_w(s) = \sum_{\substack{s\in\mathbb{B}^n\\|s|=r}}\psi_w(s) = \sum_{\substack{s\in\mathbb{B}^n\\|s|=r}}(-1)^{|w\wedge s|}, \tag{20}$$

and we can now change the index of the sum from s to $l = |w \wedge s|$. Written with the new index we only need to count for each l how many binary strings $s \in S_r$ have the property that $|w \wedge s| = l$, that is:

$$\sum_{\substack{s\in\mathbb{B}^n\\|s|=r}}(-1)^{|w\wedge s|} = \sum_{l=0}^{n}(-1)^l\left|\{s\in\mathbb{B}^n||s| = r \text{ and } |w\wedge s| = l\}\right|. \tag{21}$$

Now we can compute the cardinality of the inner set in Equation (21) using counting arguments. We need to count how many ways we can distribute the r 1s in the string s such that they coincide with the 1s of w in exactly l positions. In order to do this, first let us put l 1s in the positions where w has 1. We can do this in $\binom{|w|}{l}$ different ways. Now, let us put the remaining $r - l$ 1s in the positions where w has 0. We can do this in $\binom{n-|w|}{r-l}$ ways. Multiplying both numbers we have the desired cardinality:

$$\left|\{s\in\mathbb{B}^n||s| = r \text{ and } |w\wedge s| = l\}\right| = \binom{|w|}{l}\binom{n-|w|}{r-l}. \tag{22}$$

We should notice here that the cardinality is zero in some cases. This happens when $l > |w|, l > r$ or $r - l > n - |w|$. However, in these cases we defined the binomial

coefficient to be zero and we can keep the previous expression. If we use Equations (22) and (21) and take into account the definition in Equation (19) we get Equation (18). □

In Equation (18) we can observe that the eigenvalue depends only on the order $|w|$ of the Walsh function. This means that there are at most $n+1$ different eigenvalues in the considered adjacency matrices. As a consequence, we can decompose any arbitrary function f as a sum of $n+1$ functions, called *elementary components* of f, where each one is an eigenvector of all the adjacency matrices.

DEFINITION 3: *Let $f : \mathbb{B}^n \to \mathbb{R}$ be a pseudo-Boolean function with Walsh expansion $f = \sum_{w\in\mathbb{B}^n} a_w \psi_w$, we define the order j elementary component of f as:*

$$f_{[j]} = \sum_{\substack{w\in\mathbb{B}^n \\ |w|=j}} a_w \psi_w, \tag{23}$$

for $0 \le j \le n$. As a consequence of the Walsh expansion of f we can write:

$$f = \sum_{j=0}^{n} f_{[j]}. \tag{24}$$

According to Proposition 2 the elementary component $f_{[j]}$ is an eigenvector of $\boldsymbol{A}^{(r)}$ with eigenvalue $\mathcal{K}^{(n)}_{r,j}$.

2.2 Krawtchouk Matrices

Krawtchouk matrices play a relevant role in the mathematical developments of the next sections. For this reason we present here some of their properties. The reader interested in these matrices (also considered polynomials) can read Feinsilver and Kocik (2005). The nth order Krawtchouk matrix is an $(n+1)\times(n+1)$ integer matrix with indices between 0 and n. In Equation (19) we provided an explicit definition of the elements of a Krawtchouk matrix. But these elements can also be implicitly defined with the help of the following generating function:

$$(1+x)^{n-j}(1-x)^j = \sum_{r=0}^{n} x^r \mathcal{K}^{(n)}_{r,j}. \tag{25}$$

From Equation (25) we deduce that $\mathcal{K}^{(n)}_{0,j} = 1$. Observe that $\mathcal{K}^{(n)}_{0,j}$ is the constant coefficient in the polynomial. Other properties of the Krawtchouk matrices are presented in the next proposition.

PROPOSITION 3: *We have the following identities between the elements of the Krawtchouk matrices:*

$$\mathcal{K}^{(n)}_{r,n-j} = (-1)^r \mathcal{K}^{(n)}_{r,j}, \tag{26}$$

$$\mathcal{K}^{(n)}_{n-r,j} = (-1)^j \mathcal{K}^{(n)}_{r,j}. \tag{27}$$

PROOF: With the help of the generating function in Equation (25) we can write:

$$\begin{aligned}\sum_{r=0}^{n}(-x)^r \mathcal{K}^{(n)}_{r,j} &= (1+(-x))^{n-j}(1-(-x))^j \\ &= (1+x)^j(1-x)^{n-j} = \sum_{r=0}^{n} x^r \mathcal{K}^{(n)}_{r,n-j},\end{aligned}$$

and identifying the coefficients of the first and last polynomials we have Equation (26). In order to prove Equation (27) we can write:

$$\begin{aligned}\sum_{r=0}^{n} x^r \mathcal{K}_{r,j}^{(n)} &= (1+x)^{n-j}(1-x)^j = (-1)^j(x+1)^{n-j}(x-1)^j \\ &= (-1)^j x^n (1+1/x)^{n-j}(1-1/x)^j = (-1)^j x^n \sum_{r=0}^{n} (1/x)^r \mathcal{K}_{r,j}^{(n)} \\ &= (-1)^j \sum_{r=0}^{n} x^{n-r} \mathcal{K}_{r,j}^{(n)} = (-1)^j \sum_{r=0}^{n} x^r \mathcal{K}_{n-r,j}^{(n)},\end{aligned}$$

identifying again the coefficients of the first and last polynomials we have (Equation (27)). □

Krawtchouk matrices also appear when we sum Walsh functions. The following proposition provides an important result in this line.

PROPOSITION 4: *Let $t \in \mathbb{B}^n$ be a binary string and $0 \leq r \leq n$. Then the following two identities hold for the sum of Walsh functions:*

$$\sum_{\substack{w \in \mathbb{B}^n \wedge t \\ |w|=r}} \psi_w(x) = \mathcal{K}_{r,|x\wedge t|}^{(|t|)}, \tag{28}$$

$$\sum_{w \in \mathbb{B}^n \wedge t} \psi_w(x) = 2^{|t|} \delta_0^{|x\wedge t|}. \tag{29}$$

PROOF: Given two binary strings $x, t \in \mathbb{B}^n$, let us denote with $x|_t$ the binary string of length $|t|$ composed of all the bits of x in the positions i where $t_i = 1$. The string t acts as a mask for x. This notation allows us to simplify the sums in Equations (28) and (29):

$$\sum_{\substack{w \in \mathbb{B}^n \wedge t \\ |w|=r}} \psi_w(x) = \sum_{\substack{w \in \mathbb{B}^n \wedge t \\ |w|=r}} \psi_{w|_t}(x|_t) = \sum_{\substack{u \in \mathbb{B}^{|t|} \\ |u|=r}} \psi_u(x|_t) = \sum_{u \in S_r} \psi_u(x|_t) = \mathcal{K}_{r,|x\wedge t|}^{(|t|)}$$

by Equation (18), and

$$\sum_{w \in \mathbb{B}^n \wedge t} \psi_w(x) = \sum_{w \in \mathbb{B}^n \wedge t} \psi_{w|_t}(x|_t) = \sum_{u \in \mathbb{B}^{|t|}} \psi_u(x|_t) = \sum_{u \in \mathbb{B}^{|t|}} \psi_{x|_t}(u) = 2^{|t|} \delta_0^{|x\wedge t|}.$$

by Equation (13). □

3 Analysis of the Mutation Operator

The bit-flip mutation operator transforms an arbitrary element $x \in \mathbb{B}^n$ to $y \in \mathbb{B}^n$ by changing the value of each bit of x with probability p. In the literature it is common to use the value $p = 1/n$ that, in expectation, changes one bit in each solution. However, if $0 < p < 1$, the mutation operator can transform x into any element of the search space with positive probability. In the following, we denote with $M_p(x)$ the random variable on $\mathbb{B}^n$ that represents the element in $\mathbb{B}^n$ reached after applying the bit-flip mutation operator with probability p to solution x.

LEMMA 1: *Given two solutions $x, y \in \mathbb{B}^n$, the probability of obtaining y after a bit-flip mutation over x is*

$$Pr\{M_p(x) = y\} = p^{|x\oplus y|}(1-p)^{n-|x\oplus y|}. \tag{30}$$

PROOF: The solution y can only be obtained if all the bits that differ from the solution x are mutated and the other ones are kept unchanged. Since the number of differing bits is $|x \oplus y|$ and each bit is individually changed with probability p we obtain the claimed result. □

We are interested in $f(M_p(x))$, the objective function value after the mutation of a solution. This value is also a random variable and we want to analyze its probability distribution. Given a particular search space, directly enumerating this distribution by evaluating every solution is not tractable. However, the theory of landscapes provides tools for extracting information from this probability distribution in an efficient way. This information arises from the moments of the probability distribution. In the following sections we analyze these moments.

3.1 Expectation

Let us start by computing the expected value of $f(M_p(x))$. The expected value is easy to compute in the case of the elementary components of a function f. The result of the next theorem was previously published by Chicano and Alba (2011). Sutton et al. (2011b) also studied the expected value after mutation and found that it must be a polynomial in p. We, however, present here the result and its proof because the notation is slightly different from the one used in the previous works.

THEOREM 1: *Let $x \in \mathbb{B}^n$ be a binary string, $f : \mathbb{B}^n \to \mathbb{R}$ a function, $f_{[j]}$ its order j elementary component and let us denote with $M_p(x)$ the random variable that represents the element in $\mathbb{B}^n$ reached after applying the bit-flip mutation operator with probability p to solution x. The expected value of the random variable $f_{[j]}(M_p(x))$ is*

$$E\{f_{[j]}(M_p(x))\} = (1-2p)^j f_{[j]}(x) \tag{31}$$

PROOF:

$$\begin{aligned} E\{f_{[j]}(M_p(x))\} &= \sum_{y \in \mathbb{B}^n} f_{[j]}(y) \Pr\{M_p(x) = y\} \\ &= \sum_{y \in \mathbb{B}^n} f_{[j]}(y) p^{|x \oplus y|} (1-p)^{n-|x \oplus y|} \end{aligned}$$

by Lemma 1

$$E\{f_{[j]}(M_p(x))\} = \sum_{r=0}^{n} \sum_{y \in N_r(x)} f_{[j]}(y) p^{|x \oplus y|} (1-p)^{n-|x \oplus y|}$$

dividing the search space

$$E\{f_{[j]}(M_p(x))\} = \sum_{r=0}^{n} \left(p^r (1-p)^{n-r} \sum_{y \in N_r(x)} f_{[j]}(y) \right)$$

by definition of N_r

$$E\{f_{[j]}(M_p(x))\} = \sum_{r=0}^{n} p^r (1-p)^{n-r} \left(\boldsymbol{A}^{(r)} f_{[j]} \right)(x)$$

by Proposition 2

$$E\{f_{[j]}(M_p(x))\} = \left(\sum_{r=0}^{n} p^r (1-p)^{n-r} \mathcal{K}_{r,j}^{(n)} \right) f_{[j]}(x). \tag{32}$$

Using the generating function for Krawtchouk matrices as shown in Equation (25), we can simplify the term within the parentheses in the following way:

$$\begin{aligned}\sum_{r=0}^{n} p^r (1-p)^{n-r} \mathcal{K}_{r,j}^{(n)} &= (1-p)^n \sum_{r=0}^{n} \left(\frac{p}{1-p}\right)^r \mathcal{K}_{r,j}^{(n)} \\ &= (1-p)^n \left(1+\frac{p}{1-p}\right)^{n-j} \left(1-\frac{p}{1-p}\right)^j \\ &= (1-p)^n \left(\frac{1}{1-p}\right)^{n-j} \left(\frac{1-2p}{1-p}\right)^j \\ &= (1-2p)^j. \end{aligned} \tag{33}$$

The previous development is valid if $p < 1$. In the case where $p = 1$ we cannot divide by $1 - p$, but even in this case the final result holds. To prove this we just have to consider that the term $(1 - p)^{n-r}$ is zero except for $r = n$ and p^r is always 1. Then we can write

$$\sum_{r=0}^{n} p^r (1-p)^{n-r} \mathcal{K}_{r,j}^{(n)} = \mathcal{K}_{n,j}^{(n)} = (-1)^j \mathcal{K}_{0,j}^{(n)} = (1-2p)^j, \tag{34}$$

where we used Equation (27) and the fact that $\mathcal{K}_{0,j}^{(n)} = 1$. We finally obtain the claimed result for all the possible values of p. □

As a direct consequence of the previous theorem we can compute the expected value of $f(M_p(x))$ for an arbitrary function with the help of the decomposition of the function into elementary components.

COROLLARY 1: *Let $x \in \mathbb{B}^n$ be a binary string, $f : \mathbb{B}^n \to \mathbb{R}$ a function and $M_p(x)$ the solution reached after applying the bit-flip mutation operator with probability p to solution x. The expected value of the random variable $f(M_p(x))$ is*

$$E\{f(M_p(x))\} = \sum_{j=0}^{n} (1-2p)^j f_{[j]}(x), \tag{35}$$

where $f_{[j]}$ is the order j elementary component of f.

PROOF: We can write f as the sum of its elementary components as $f = \sum_{j=0}^{n} f_{[j]}$. Then, we can compute the expected value as:

$$E\{f(M_p(x))\} = \sum_{j=0}^{n} E\{f_{[j]}(M_p(x))\} = \sum_{j=0}^{n} (1-2p)^j f_{[j]}(x), \tag{36}$$

where we used the result of Theorem 1. □

3.2 Higher-Order Moments

Equation (35) can be used to compute the expected value of $f(M_p(x))$. We may also use it to extend to higher-order moments, as in the following theorem.

THEOREM 2: *Let $x \in \mathbb{B}^n$ be a binary string, $f : \mathbb{B}^n \to \mathbb{R}$ a function and $M_p(x)$ the solution reached after applying the bit-flip mutation operator with probability p to solution x. The mth moment of the random variable $f(M_p(x))$ is*

$$\mu_m\{f(M_p(x))\} = \sum_{j=0}^{n} (1-2p)^j f_{[j]}^m(x), \tag{37}$$

where $f^m_{[j]}$ is the order j elementary component of f^m.[1]

PROOF: By definition, $\mu_m\{f(M_p(x))\}$ can be expressed as the expectation of the random variable $f^m(M_p(x))$. Then, using Equation (35) we can write:

$$\mu_m\{f(M_p(x))\} = E\{f^m(M_p(x))\} = \sum_{j=0}^{n}(1-2p)^j f^m_{[j]}(x). \qquad \square$$

We define the 0th moment $\mu_0\{f(M_p(x))\} = 1$. We can observe from Equation (37) that all the higher-order moments are polynomials in p, just like the expectation (first-order moment).

Let us now introduce some new notation. Let us denote with $\boldsymbol{\mu}\{f(M_p(x))\}$ the vector of moments, that is, the mth component of this vector is the mth moment. We do not limit the number of components of this vector; we can consider it as an infinite-dimensional vector. Later we will see that only a finite number of elements of this vector would be required for our purposes. We define the matrix function $\boldsymbol{F}(x)$ as $\boldsymbol{F}_{m,j}(x) = f^m_{[j]}(x)$ where $0 \leq j \leq n$ and $m \geq 0$. Let us also define the vector $\boldsymbol{\Lambda}(p)$ as $\boldsymbol{\Lambda}_j(p) = (1-2p)^j$ for $0 \leq j \leq n$.

Using the new notation we can write Equation (37) in vector form as:

$$\begin{pmatrix} \mu_0\{f(M_p(x))\} \\ \mu_1\{f(M_p(x))\} \\ \vdots \\ \mu_m\{f(M_p(x))\} \\ \vdots \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ f_{[0]}(x) & f_{[1]}(x) & \dots & f_{[n]}(x) \\ \vdots & \vdots & \ddots & \vdots \\ f^m_{[0]}(x) & f^m_{[1]}(x) & \dots & f^m_{[n]}(x) \\ \vdots & \vdots & \ddots & \vdots \end{pmatrix} \begin{pmatrix} 1 \\ 1-2p \\ \vdots \\ (1-2p)^n \end{pmatrix},$$

or in more compact form:

$$\boldsymbol{\mu}\{f(M_p(x))\} = \boldsymbol{F}(x)\boldsymbol{\Lambda}(p), \tag{38}$$

where $\boldsymbol{F}(x)$ and $\boldsymbol{\Lambda}(p)$ are multiplied using the matrix product. This new form of writing Equation (37) has the property of expressing the vector of moments of $f(M_p(x))$ as the product of a matrix that depends on the objective function (and solution x) and a vector that depends on the mutation operator and its parameter p. In some sense, we can claim that Equation (38) decomposes the moments into a problem-dependent part, $\boldsymbol{F}(x)$, and an operator-dependent part, $\boldsymbol{\Lambda}(p)$. This is the kind of equation we are looking for, since it can be applied to different problems provided that the problem-dependent part for each one is computed and we do not need to recompute the operator-dependent part. In the same way, it can also be applied to any parameter of the operator (value of p) without recomputing the problem-dependent part.

We should note here that the first column of matrix $\boldsymbol{F}(x)$ provides the statistical moments of the fitness distribution in the whole search space considering a uniform random distribution. Thus, $\boldsymbol{F}_{1,0}(x) = f_{[0]}(x)$ is the average value of the evaluation function in the search space, $\boldsymbol{F}_{2,0}(x) = f^2_{[0]}(x)$ is the second-order moment, and so on. We can prove this by setting $p = 1/2$ in Equation (38) because a probability of $p = 1/2$ for bit-flip mutation is equivalent to a uniform random selection of a solution in the search space. All the elements but the first in $\boldsymbol{\Lambda}(p)$ vanish and we get the claimed result.

[1] We use this notation instead of $(f^m)_{[j]}$ to simplify the expressions, but $f^m_{[j]}$ should not be confused with $f_{[j]}$ to the power of m.

3.3 Computing the Matrix Function $\boldsymbol{F}$

The computation of the matrix function $\boldsymbol{F}$ is not efficient in general. Sutton et al. (2012) provide an algorithm to compute the Walsh decomposition of f^m. Using this Walsh decomposition it is possible to obtain the elementary components of f^m, as required for the computation of $\boldsymbol{F}(x)$. If the Walsh decomposition of f is:

$$f = \sum_{w \in \mathbb{B}^n} a_w \psi_w,$$

then the Walsh decomposition of the mth power f^m is:

$$f^m = \left(\sum_{w \in \mathbb{B}^n} a_w \psi_w \right)^m = \sum_{\sum_{w \in \mathbb{B}^n} i_w = m} \binom{m}{i_{00\ldots0}, i_{00\ldots1}, \ldots, i_{11\ldots1}} \prod_{w \in \mathbb{B}^n} a_w^{i_w} \psi_w^{i_w}$$

This procedure has the advantage that it is general and can be used with any function defined over bitstrings. The drawback, however, is its inefficiency when m is high. Thus, for each particular problem, we should analyze the objective function in order to find an efficient way of evaluating the matrix function $\boldsymbol{F}$ in an arbitrary solution x. In Section 4 we analyze two problems and provide an efficient computation of this matrix function for these problems.

In some cases, the efficient (polynomial time) evaluation of $\boldsymbol{F}(x)$ can only be possible if NP = P. This happens for example in the SAT problem as the following proposition states.

PROPOSITION 5: *Let us consider the SAT problem and an evaluation function $f(x)$ that takes value 1 if $x \in \mathbb{B}^n$ satisfies the propositional formula and 0 otherwise. If there exists a polynomial time algorithm for computing $f_{[0]}$ then NP = P.*

PROOF: The value $f_{[0]}$ is the average value of the objective function in the whole search space. Since f can only take values 0 and 1, if $f_{[0]} > 0$ then the formula is satisfiable. Thus, if we find a polynomial time algorithm to evaluate $f_{[0]}$ we can solve the decision problem in polynomial time. But, as SAT is NP-complete, then NP = P. □

As a consequence of the previous proposition we cannot ensure that an efficient evaluation of the matrix function $\boldsymbol{F}(x)$ exists in general. The complexity of computing $\boldsymbol{F}(x)$ depends on the problem.

3.4 Fitness Probability Distribution

With the help of the moments vector $\boldsymbol{\mu}\{f(M_p(x))\}$ we can compute the probability distribution of the values of f in a mutated solution. In order to do this we proceed in the same way as Sutton et al. (2011a).

Let us call $\xi_0 < \xi_1 < \cdots < \xi_{q\text{-}1}$ to the q possible values that the function f can take in the search space. Since we are dealing with a finite search space, q is a finite number (perhaps very large). We are interested in computing $\Pr\{f(M_p(x)) = \xi_i\}$ for $0 \leq i < q$. In order to simplify the notation in the following we define the vector of probabilities $\boldsymbol{\pi}(f(M_p(x)))$ as $\boldsymbol{\pi}_i(f(M_p(x))) = \Pr\{f(M_p(x)) = \xi_i\}$.

THEOREM 3: *Let us consider the binary hypercube and let us denote with ξ_i the possible values that the objective function f can take in the search space, where $\xi_i < \xi_{i+1}$ for $0 \leq i < q - 1$. Then, the vector of probabilities $\boldsymbol{\pi}(f(M_p(x)))$ can be computed as:*

$$\boldsymbol{\pi}(f(M_p(x))) = \underbrace{\left(\boldsymbol{V}^T\right)^{-1} \boldsymbol{F}(x)}_{\textit{problem-dependent}} \boldsymbol{\Lambda}(p), \tag{39}$$

where the matrix function $\boldsymbol{F}(x)$ is limited to the first q rows and $\boldsymbol{V}$ denotes the Vandermonde matrix for the ξ_i values, that is, $\boldsymbol{V}_{i,j} = \xi_i^j$ for $0 \leq i, j < q$.

PROOF: We can compute the mth moment $\boldsymbol{\mu}_m\{f(M_p(x))\}$ using the following expression:

$$\boldsymbol{\mu}_m\{f(M_p(x))\} = \sum_{i=0}^{q-1} \xi_i^m \Pr\{f(M_p(x)) = \xi_i\}. \tag{40}$$

We can write this in vector form as:

$$\begin{aligned}\boldsymbol{\mu}\{f(M_p(x))\} &= \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \xi_0^1 & \xi_1^1 & \cdots & \xi_{q-1}^1 \\ \vdots & \vdots & \ddots & \vdots \\ \xi_0^{q-1} & \xi_1^{q-1} & \cdots & \xi_{q-1}^{q-1} \end{pmatrix} \begin{pmatrix} \Pr\{f(M_p(x)) = \xi_0\} \\ \Pr\{f(M_p(x)) = \xi_1\} \\ \vdots \\ \Pr\{f(M_p(x)) = \xi_{q-1}\} \end{pmatrix} \\ &= \boldsymbol{V}^T \boldsymbol{\pi}(f(M_p(x))). \end{aligned} \tag{41}$$

Using Equation (38) we can write:

$$\boldsymbol{\mu}\{f(M_p(x))\} = \boldsymbol{V}^T \boldsymbol{\pi}(f(M_p(x))) = \boldsymbol{F}(x)\boldsymbol{\Lambda}(p), \tag{42}$$

and solving $\boldsymbol{\pi}(f(M_p(x)))$ we finally get Equation (39). The determinant of the Vandermonde matrix is $\prod_{0 \leq i < j < q} (\xi_i - \xi_j)$ (Mirsky, 1955, pp. 17–18) and the matrix is nonsingular if and only if all the ξ_i values are different. This is our case, so the Vandermonde matrices we use are invertible. □

Again we can observe that $\boldsymbol{\pi}(f(M_p(x)))$ is the product of a term that is problem-dependent and a vector that depends on the parameter of the mutation p. From Equation (39) it is clear that each particular probability $\boldsymbol{\pi}_i(f(M_p(x)))$ is a polynomial in p.

We can also compute the cumulative density function $\boldsymbol{\Pi}(f(M_p(x)))$ defined by:

$$\boldsymbol{\Pi}_i(f(M_p(x))) = \Pr\{f(M_p(x)) \leq \xi_i\} = \sum_{j=0}^{i} \boldsymbol{\pi}_j(f(M_p(x))). \tag{43}$$

We can write the previous equation in vector form as:

$$\boldsymbol{\Pi}(f(M_p(x))) = \boldsymbol{L}\, \boldsymbol{\pi}(f(M_p(x))). \tag{44}$$

where $\boldsymbol{L}$ is the lower triangular matrix defined by

$$\boldsymbol{L}_{i,j} = \begin{cases} 1 & \text{if } i \geq j, \\ 0 & \text{otherwise.} \end{cases}$$

We can note again that each element of $\boldsymbol{\Pi}(f(M_p(x)))$ is a polynomial in p. The component $\boldsymbol{\Pi}_i(f(M_p(x)))$ is the probability of reaching a solution y with function value $f(y) \leq \xi_i$ after the mutation with parameter p. If x has function value $f(x) = \xi_i$, then $1 - \boldsymbol{\Pi}_i(f(M_p(x)))$ is the probability of improving the function value of solution x in one application of bit-flip mutation. For problems in which the matrix $\boldsymbol{F}$ can be efficiently computed, the expression $1 - \boldsymbol{\Pi}_i(f(M_p(x)))$ could be used as the base for a new mutation operator that tries to maximize the probability of an improving move.

4 Case Studies

In this section we present the elementary landscape decomposition of two well-known problems and their powers (the $\boldsymbol{F}(x)$ matrix). With this decomposition we can compute the probability distribution of any solution after mutation. We start by analyzing a Onemax problem. In Section 4.2 we analyze MAX-SAT.

4.1 Onemax

Onemax is a linear pseudo-Boolean fitness function that is often used in the analysis of evolutionary algorithms. In our case, we consider the sum of all order-1 Walsh functions, which is related to Onemax by a simple linear transformation. That is:

$$f(x) = \sum_{i=1}^{n} \psi_{\underline{i}}(x) = n - 2\sum_{i=1}^{n} x_i = n - 2|x|. \tag{45}$$

The objective function in Onemax is $|x|$ (the number of ones in $x \in \mathbb{B}^n$). Maximizing the number of ones in x (original Onemax problem) is equivalent to minimizing $f(x)$. We should note here that $f(x)$ can take values in the range $[-n, n]$ by steps of 2. That is, the range of f is the set $\{n - 2j | j \in \mathbb{N}, 0 \leq j \leq n\}$. Although we study here the function $f(x)$ defined in Equation (45) for the sake of simplicity, we will see at the end of this section that the probability distribution after mutation of the regular Onemax function is the same as $f(x)$.

The following lemma provides intermediate results that will be useful in the search for an expression for $\boldsymbol{F}(x)$.

LEMMA 2: *The sum of all the Walsh functions with the same order is related to the Krawtchouk matrices by means of the following identity:*

$$\sum_{\substack{w \in \mathbb{B}^n \\ |w|=p}} \psi_w(x) = \mathcal{K}^{(n)}_{p,|x|}. \tag{46}$$

PROOF: The claim follows immediately from Equation (28) when $t = 11 \ldots 1$. □

THEOREM 4: *The matrix function $\boldsymbol{F}(x)$ for the objective function $f(x)$ defined in Equation (45) depends only on $|x|$ and its elements satisfy the following identity:*

$$\boldsymbol{F}_{m,j}(x) = \boldsymbol{\Xi}^{(n)}_{m,j} \mathcal{K}^{(n)}_{j,|x|}, \tag{47}$$

where $\mathcal{K}^{(n)}$ is the nth Krawtchouk matrix and $\boldsymbol{\Xi}^{(n)}$ is the matrix defined as:

$$\boldsymbol{\Xi}^{(n)}_{m,j} = \frac{1}{2^n} \sum_{k=0}^{n} (n-2k)^m \mathcal{K}^{(n)}_{k,j}. \tag{48}$$

PROOF: Let us write the Walsh decomposition of f^m. Given a binary string $w \in \mathbb{B}^n$, the Walsh coefficient $a^{(m)}_w$ of f^m is

$$a^{(m)}_w = \frac{1}{2^n} \sum_{x \in \mathbb{B}^n} \psi_w(x) f^m(x) = \frac{1}{2^n} \sum_{x \in \mathbb{B}^n} \psi_w(x) (n - 2|x|)^m$$

dividing the search space yields

$$a^{(m)}_w = \frac{1}{2^n} \sum_{k=0}^{n} (n-2k)^m \sum_{\substack{x \in \mathbb{B}^n \\ |x|=k}} \psi_w(x) = \frac{1}{2^n} \sum_{k=0}^{n} (n-2k)^m \, \mathcal{K}^{(n)}_{k,|w|} = \boldsymbol{\Xi}^{(n)}_{m,|w|}, \tag{49}$$

where we used the result of Lemma 2 and introduced the matrix $\Xi^{(n)}$ to simplify the notation. Now we can sum together all the Walsh functions of the same order j to find the elementary component $f^m_{[j]}$:

$$\boldsymbol{F}_{m,j}(x) = f^m_{[j]}(x) = \sum_{\substack{w\in\mathbb{B}^n \\ |w|=j}} a^{(m)}_w \psi_w(x) = \Xi^{(n)}_{m,j} \sum_{\substack{w\in\mathbb{B}^n \\ |w|=j}} \psi_w(x) = \Xi^{(n)}_{m,j} \mathcal{K}^{(n)}_{j,|x|}, \tag{50}$$

where we used Lemma 2 in the last step. □

In the following proposition we provide a property of the $\Xi^{(n)}$ matrix that is useful to simplify the computation of the matrix.

PROPOSITION 6: *All the elements $\Xi^{(n)}_{m,j}$ in which $m + j$ is odd are zero.*

PROOF: We can develop Equation (48) to write:

$$2\Xi^{(n)}_{m,j} = \frac{1}{2^n} \sum_{k=0}^{n} \left((n-2k)^m \mathcal{K}^{(n)}_{k,j} + (n-2k)^m \mathcal{K}^{(n)}_{k,j} \right)$$

changing k by $n - k$ yields

$$\begin{aligned} 2\Xi^{(n)}_{m,j} &= \frac{1}{2^n} \sum_{k=0}^{n} \left((n-2k)^m \mathcal{K}^{(n)}_{k,j} + (2k-n)^m \mathcal{K}^{(n)}_{n-k,j} \right) \\ &= \frac{1}{2^n} \sum_{k=0}^{n} (n-2k)^m \left(\mathcal{K}^{(n)}_{k,j} + (-1)^m \mathcal{K}^{(n)}_{n-k,j} \right) \end{aligned}$$

by Proposition 3 and Equation (27),

$$\begin{aligned} 2\Xi^{(n)}_{m,j} &= \frac{1}{2^n} \sum_{k=0}^{n} (n-2k)^m \left(\mathcal{K}^{(n)}_{k,j} + (-1)^{m+j} \mathcal{K}^{(n)}_{k,j} \right) \\ &= \frac{1}{2^n} \sum_{k=0}^{n} (n-2k)^m \mathcal{K}^{(n)}_{k,j} \left(1 + (-1)^{m+j} \right). \end{aligned}$$

If $m + j$ is odd, all the terms in the sum are zero and, thus, $\Xi^{(n)}_{m,j} = 0$. □

Theorem 4 claims that $\boldsymbol{F}$ depends only on $|x|$ and not on the solution itself. As a consequence, the vector of probabilities $\boldsymbol{\pi}(f(M_p(x)))$ depends only on $|x|$. But, according to Equation (45), $|x|$ is related to the fitness value of a solution by $f(x) = n - 2|x|$, and the vector of probabilities $\boldsymbol{\pi}$ depends only on the fitness level of the solution we are evaluating. We can then build a matrix, denoted $\boldsymbol{\varpi}$, where element $\boldsymbol{\varpi}_{i,j}$ is the probability of generating a solution with fitness ξ_j using bit-flip mutation from a solution with fitness ξ_i. This matrix depends on p (probability of mutation), but we omit p in the notation to make it simpler. The expression for $\boldsymbol{\varpi}_{i,j}$ can be obtained using simple counting arguments, without the need of the mathematical framework developed in Section 3. However, in the next theorem we provide an expression for this matrix using our mathematical framework. The purpose of this result is twofold: it proves that $\boldsymbol{\varpi}_{i,j}$ can be computed using our framework and, to the best of our knowledge, it provides a previously unknown expression for $\boldsymbol{\varpi}_{i,j}$ involving Krawtchouk matrices.

THEOREM 5: *Given the objective function defined in Equation (45) over $\mathbb{B}^n$, the probability of reaching a solution with fitness $\xi_j = 2j - n$ when bit-flip mutation with probability p is applied*

to a solution with fitness $\xi_i = 2i - n$ is given by:

$$\boldsymbol{\varpi}_{i,j} = \sum_{l=0}^{n} \mathcal{K}_{j,l}^{(n)} (1-2p)^l \mathcal{K}_{l,i}^{(n)}, \tag{51}$$

where $0 \leq i, j \leq n$.

PROOF: First, we will express the matrix $\boldsymbol{\Xi}^{(n)}$ as a product of two other matrices.

$$\begin{aligned}
\boldsymbol{\Xi}_{c,j}^{(n)} &= \frac{1}{2^n} \sum_{k=0}^{n} (n-2k)^c \mathcal{K}_{k,j}^{(n)} \quad \text{replacing } k \text{ by } n-k \\
&= \frac{1}{2^n} \sum_{k=0}^{n} (2k-n)^c \mathcal{K}_{n-k,j}^{(n)} = \frac{1}{2^n} \sum_{k=0}^{n} \xi_k^c \mathcal{K}_{n-k,j}^{(n)} = \frac{1}{2^n} \sum_{k=0}^{n} \boldsymbol{V}_{k,c} \tilde{\mathcal{K}}_{k,j}^{(n)} \\
&= \frac{1}{2^n} \left(\boldsymbol{V}^T \tilde{\mathcal{K}}^{(n)} \right)_{c,j},
\end{aligned} \tag{52}$$

where we used the Vandermonde matrix and we introduced a new matrix $\tilde{\mathcal{K}}^{(n)}$. This is the Krawtchouk matrix of order n in which the rows are reversed. Then, we have $\boldsymbol{\Xi}^{(n)} = 2^{-n} \boldsymbol{V}^T \tilde{\mathcal{K}}^{(n)}$. Let us now define the vector $\mathcal{K}_{*,j}^{(n)}$ as the jth column of the nth order Krawtchouk matrix. We can write the matrix function $\boldsymbol{F}(x)$ defined in Equation (50) in a compact way as:

$$\boldsymbol{F}(x) = \boldsymbol{\Xi}^{(n)} \operatorname{diag}\left(\mathcal{K}_{*,|x|}^{(n)}\right), \tag{53}$$

where the function diag () maps a vector into a matrix having the vector in the diagonal. If we introduce this compact expression of $\boldsymbol{F}(x)$ in Equation (39) we obtain:

$$\begin{aligned}
\boldsymbol{\pi}(f(M_p(x))) &= \left(\boldsymbol{V}^T\right)^{-1} \boldsymbol{F}(x) \boldsymbol{\Lambda}(p) = \left(\boldsymbol{V}^T\right)^{-1} \boldsymbol{\Xi}^{(n)} \operatorname{diag}\left(\mathcal{K}_{*,|x|}^{(n)}\right) \boldsymbol{\Lambda}(p) \\
&= \frac{1}{2^n} \left(\boldsymbol{V}^T\right)^{-1} \boldsymbol{V}^T \tilde{\mathcal{K}}^{(n)} \operatorname{diag}\left(\mathcal{K}_{*,|x|}^{(n)}\right) \boldsymbol{\Lambda}(p) = \frac{1}{2^n} \tilde{\mathcal{K}}^{(n)} \operatorname{diag}\left(\mathcal{K}_{*,|x|}^{(n)}\right) \boldsymbol{\Lambda}(p) \\
&= \frac{1}{2^n} \tilde{\mathcal{K}}^{(n)} \left(\boldsymbol{\Lambda}(p) \circ \mathcal{K}_{*,|x|}^{(n)}\right),
\end{aligned} \tag{54}$$

where the symbol $\circ$ denotes the Hadamard product[2] of matrices and we used the fact that $\operatorname{diag}(\boldsymbol{A})\, \boldsymbol{B} = \boldsymbol{A} \circ \boldsymbol{B}$.

According to the definition of $\boldsymbol{\varpi}$, it must be related to $\boldsymbol{\pi}$ by the following equation:

$$\boldsymbol{\varpi}_{i,j} = \boldsymbol{\pi}_j(f(M_p(x))) \quad \text{for any } x \text{ such that } f(x) = \xi_i. \tag{55}$$

Since $\xi_i = 2i - n$, $f(x) = n - 2|x| = \xi_i$ if and only if $|x| = n - i$, and using Equation (54) we have:

$$\begin{aligned}
\boldsymbol{\varpi}_{i,j} &= \boldsymbol{\pi}_j(f(M_p(x))) \quad \text{for any } x \text{ with } |x| = n - i \\
&= \frac{1}{2^n} \left(\tilde{\mathcal{K}}^{(n)} (\boldsymbol{\Lambda}(p) \circ \mathcal{K}_{*,n-i}^{(n)})\right)_j = \frac{1}{2^n} \sum_{l=0}^{n} \tilde{\mathcal{K}}_{j,l}^{(n)} \left(\boldsymbol{\Lambda}(p) \circ \mathcal{K}_{*,n-i}^{(n)}\right)_l \\
&= \frac{1}{2^n} \sum_{l=0}^{n} \tilde{\mathcal{K}}_{j,l}^{(n)} \boldsymbol{\Lambda}_l(p) \mathcal{K}_{l,n-i}^{(n)} = \frac{1}{2^n} \sum_{l=0}^{n} \mathcal{K}_{n-j,l}^{(n)} \boldsymbol{\Lambda}_l(p) \mathcal{K}_{l,n-i}^{(n)}
\end{aligned}$$

[2]The Hadamard product of two matrices with the same dimension is the element wise product of the matrices.

by Proposition 3

$$\boldsymbol{\varpi}_{i,j} = \frac{1}{2^n}\sum_{l=0}^{n} \mathcal{K}^{(n)}_{j,l}(-1)^l \boldsymbol{\Lambda}_l(p)\mathcal{K}^{(n)}_{l,n-i} = \frac{1}{2^n}\sum_{l=0}^{n} \mathcal{K}^{(n)}_{j,l}\boldsymbol{\Lambda}_l(p)\mathcal{K}^{(n)}_{l,i}, \tag{56}$$

and we get Equation (51) just considering that $\boldsymbol{\Lambda}_l(p) = (1-2p)^l$. □

In the following proposition we provide two properties of the $\boldsymbol{\varpi}$ matrix that are useful to reduce the computational complexity.

PROPOSITION 7: *The matrix $\boldsymbol{\varpi}$ has the following properties:*

$$\binom{n}{i}\boldsymbol{\varpi}_{i,j} = \binom{n}{j}\boldsymbol{\varpi}_{j,i}, \tag{57}$$

$$\boldsymbol{\varpi}_{n-i,n-j} = \boldsymbol{\varpi}_{i,j}, \tag{58}$$

where $0 \leq i, j \leq n$.

PROOF: The first property is a consequence of an analogous property of the Krawtchouk matrices: $\binom{n}{j}\mathcal{K}^{(n)}_{i,j} = \binom{n}{i}\mathcal{K}^{(n)}_{j,i}$ (Terras, 1999, p. 179). We can write:

$$\begin{aligned}\binom{n}{i}\boldsymbol{\varpi}_{i,j} &= \frac{1}{2^n}\sum_{l=0}\mathcal{K}^{(n)}_{j,l}\boldsymbol{\Lambda}_l(p)\binom{n}{i}\mathcal{K}^{(n)}_{l,i} = \frac{1}{2^n}\sum_{l=0}\binom{n}{l}\mathcal{K}^{(n)}_{j,l}\boldsymbol{\Lambda}_l(p)\mathcal{K}^{(n)}_{i,l}\\ &= \frac{1}{2^n}\sum_{l=0}\binom{n}{j}\mathcal{K}^{(n)}_{l,j}\boldsymbol{\Lambda}_l(p)\mathcal{K}^{(n)}_{i,l} = \binom{n}{j}\boldsymbol{\varpi}_{j,i}.\end{aligned}$$

The second property is a consequence of Proposition 3:

$$\begin{aligned}\boldsymbol{\varpi}_{n-i,n-j} &= \frac{1}{2^n}\sum_{l=0}\mathcal{K}^{(n)}_{n-j,l}\boldsymbol{\Lambda}_l(p)\mathcal{K}^{(n)}_{l,n-i} = \frac{1}{2^n}\sum_{l=0}(-1)^l\mathcal{K}^{(n)}_{j,l}\boldsymbol{\Lambda}_l(p)(-1)^l\mathcal{K}^{(n)}_{l,i}\\ &= \frac{1}{2^n}\sum_{l=0}\mathcal{K}^{(n)}_{j,l}\boldsymbol{\Lambda}_l(p)\mathcal{K}^{(n)}_{l,i} = \boldsymbol{\varpi}_{i,j}.\end{aligned}$$

□

At this point we can discuss the utility of the $\boldsymbol{\varpi}$ matrix. We can see $\boldsymbol{\varpi}$ as a practical substitute for all the probability vectors $\boldsymbol{\pi}(M_p(f(x)))$ in the case of the objective function $f(x)$ defined in Equation (45). In general, the components of the previous vector depend on the solution x. However, in the particular case of the Onemax-related function in Equation (45), the components depend only on the fitness level ξ_i the solution has. This way, we can forget the concrete solution x and focus only on the fitness levels ξ_i. Furthermore, the number of fitness levels is $n+1$ and the complexity of computing any element of $\boldsymbol{\varpi}$ using Equation (51) is $O(n)$, where we assume that the Krawtchouk matrix $\mathcal{K}^{(n)}$ is precomputed.[3] This means that we can compute the probabilities of reaching any fitness level from any other one after bit-flip mutation in $O(n^3)$. That is, we obtain in polynomial time a practical piece of information that summarizes the behavior of bit-flip mutation in this problem. We will see in Section 5 how this information can be used.

[3]Krawtchouk matrix $\mathcal{K}^{(n)}$ can be precomputed in $O(n^3)$ using Proposition 2.1 of Feinsilver and Kocik (2005).

We derived the ϖ matrix for only one objective function. Now we wonder whether similar ϖ matrices can be derived for other objective functions. The answer to this question is not easy in general, but the next results give a first answer along this line. Let us first formally define the property that allows one to compute a matrix such as ϖ.

DEFINITION 4: *Let $f(x)$ be an objective function and let us call $\xi_0 < \xi_1 < \cdots < \xi_{q-1}$ to the different values it can take. We say that the function f has a fitness-dependent distribution for a unary operator if the probability distribution of the objective value after applying the operator to any solution does only depend on the objective value of the initial solution. In formal terms, if $U(x) \in \mathbb{B}^n$ is a random variable that represents the application of the unary U operator to x, we have*

$$\forall x, y \in \mathbb{B}^n,\ f(x) = f(y) \Rightarrow Pr\{f(U(x)) = \xi_j\} = Pr\{f(U(y)) = \xi_j\}, \tag{59}$$

for all the possible ξ_j values. If this happens, then we can define a matrix ϖ whose elements are:

$$\varpi_{i,j} = Pr\{f(U(x)) = \xi_j | f(x) = \xi_i\}. \tag{60}$$

There is a trivial family of functions having fitness-dependent distributions for any unary operator. It is the family of injective functions. In these functions each particular solution has a unique image and the fitness-dependency condition trivially holds. However, the probability matrix in this case has size $2^n \times 2^n$ (the size of the search space squared), which makes this treatment impractical. Even simple linear pseudo-Boolean functions (such as BINVAL) can have this property. The next theorem claims that the property of having a fitness-dependent distribution can be kept even after some simple manipulations of the fitness function.

THEOREM 6: *Let $g(x)$ be an objective function having a fitness-dependent distribution for the unary operator U and let us call $\varpi^{(g)}$ the associated probability matrix, where we used the name of the function as superindex. Then, the function $f(x)$, which is a composition of $g(x)$ with another function, also has a fitness-dependent distribution for U under the following conditions:*

- *When $f(x) = h(g(x))$ for h, a strictly increasing function. The probability matrix does not change: $\varpi^{(f)} = \varpi^{(g)}$.*
- *When $f(x) = h(g(x))$ for h, a strictly decreasing function. The probability matrix flips its rows and columns: $\varpi^{(f)}_{i,j} = \varpi^{(g)}_{(q-1)-i,(q-1)-j}$.*
- *When $f(x) = g(x \oplus u)$ for $u \in \mathbb{B}^n$ and U commute with the $\oplus$ operator: $Pr\{U(x \oplus u) = y\} = Pr\{U(x) \oplus u = y\}\ \forall x, u \in \mathbb{B}^n$. The probability matrix does not change: $\varpi^{(f)} = \varpi^{(g)}$.*

PROOF: First, we can observe that in the three cases the number of values that f can take is the same as the number of values that g can take, $|f(\mathbb{B}^n)| = |g(\mathbb{B}^n)| = q$. Then, let us denote with $\xi_0^{(g)} < \xi_1^{(g)} < \cdots < \xi_{q-1}^{(g)}$ these values for the g function. We will use the notation $\xi_i^{(f)}$ to refer to the corresponding values of f.

Let us start with the first case: $f(x) = h(g(x))$ and h strictly increasing. In this case $\xi_i^{(f)} = h(\xi_i^{(g)})$ for all $0 \le i < q$. And the property of having a fitness-dependent distribution trivially holds for f since if $f(x) = f(y)$ then $g(x) = g(y)$ and consequently

$\Pr\{g(U(x)) = \xi_j^{(g)}\} = \Pr\{g(U(y)) = \xi_j^{(g)}\}$, which implies $\Pr\{f(U(x)) = \xi_j^{(f)}\} = \Pr\{f(U(y)) = \xi_j^{(f)}\}$. Regarding the probability matrix we have:

$$\varpi_{i,j}^{(f)} = \Pr\left\{ f(U(x)) = \xi_j^{(f)} | f(x) = \xi_i^{(f)} \right\}$$

by definition of f

$$\varpi_{i,j}^{(f)} = \Pr\left\{ h(g(U(x))) = h\left(\xi_j^{(g)}\right) | h(g(x)) = h\left(\xi_i^{(g)}\right) \right\}$$

since h is strictly monotonic

$$\varpi_{i,j}^{(f)} = \Pr\left\{ g(U(x)) = \xi_j^{(g)} | g(x) = \xi_i^{(g)} \right\} = \varpi_{i,j}^{(g)}. \tag{61}$$

In the second case, in which $f(x) = h(g(x))$ for h, a strictly decreasing function, we can prove that f has a fitness-dependent distribution with an argument similar to the first case. However, the probability matrix is different due to the change in the order of the values $\xi_i^{(f)}$. Since h is strictly decreasing, we have $\xi_i^{(f)} = h(\xi_{(q-1)-i}^{(g)})$. Thus, the elements of the probability matrix are given by:

$$\varpi_{i,j}^{(f)} = \Pr\left\{ f(U(x)) = \xi_j^{(f)} | f(x) = \xi_i^{(f)} \right\}$$

by definition of f

$$\varpi_{i,j}^{(f)} = \Pr\left\{ h(g(U(x))) = h\left(\xi_{(q-1)-j}^{(g)}\right) | h(g(x)) = h\left(\xi_{(q-1)-i}^{(g)}\right) \right\}$$

h strictly monotonic

$$\varpi_{i,j}^{(f)} = \Pr\{g(U(x)) = \xi_{(q-1)-j}^{(g)} | g(x) = \xi_{(q-1)-i}^{(g)}\} = \varpi_{(q-1)-i,(q-1)-j}^{(g)}. \tag{62}$$

Finally, let us prove the last case. The values that the function takes do not change, that is: $\xi_i^{(f)} = \xi_i^{(g)}$. If $f(x) = f(y)$ then $g(x \oplus u) = g(y \oplus u)$, which implies $\Pr\{g(U(x \oplus u)) = \xi_j^{(g)}\} = \Pr\{g(U(y \oplus u)) = \xi_j^{(g)}\}$ by hypothesis. But if U commutes with $\oplus$ then we have:

$$\Pr\left\{ g(U(x \oplus u)) = \xi_j^{(g)} \right\} = \Pr\left\{ g(U(x) \oplus u) = \xi_j^{(g)} \right\} = \Pr\left\{ f(U(x)) = \xi_j^{(f)} \right\}, \tag{63}$$

where we used the definition $f(x) = g(x \oplus u)$ in the last step and the fact that $\xi_i^{(f)} = \xi_i^{(g)}$.

As a consequence we have $\Pr\{f(U(x)) = \xi_j^{(f)}\} = \Pr\{f(U(y)) = \xi_j^{(f)}\}$ and f has a fitness-dependent distribution for U. The elements of the probability matrix are:

$$\varpi_{i,j}^{(f)} = \Pr\left\{ f(U(x)) = \xi_j^{(f)} | f(x) = \xi_i^{(f)} \right\}$$

by definition of f

$$\varpi_{i,j}^{(f)} = \Pr\left\{ g(U(x) \oplus u) = \xi_j^{(g)} | g(x \oplus u) = \xi_i^{(g)} \right\}$$

by commutation of U and $\oplus$

$$\varpi_{i,j}^{(f)} = \Pr\left\{ g(U(x \oplus u)) = \xi_j^{(g)} | g(x \oplus u) = \xi_i^{(g)} \right\} = \varpi_{i,j}^{(g)}. \tag{64}$$

□

The only condition imposed on the unary operator in the previous theorem is the commutation with $\oplus$. Fortunately, the bit-flip mutation operator commutes with $\oplus$. Furthermore, we provide in the next proposition a result that generalizes that of the mutation operator.

PROPOSITION 8: *If a unary operator U has the property $Pr\{U(x) = y\} = f(x \oplus y)$ for a real function f then it commutes with the $\oplus$ operation.*

PROOF: For any $x, u, y \in \mathbb{B}^n$ we can write:

$$\Pr\{U(x \oplus u) = y\} = f(x \oplus u \oplus y) = \Pr\{U(x) = u \oplus y\} = \Pr\{U(x) \oplus u = y\}, \tag{65}$$

and we have the commutation property. □

The bit-flip mutation satisfies the hypothesis of the previous proposition, as Lemma 1 states. Now, we can combine the results of Theorem 6 and Proposition 8 to provide a concrete result for the Onemax-related functions.

PROPOSITION 9: *All the objective functions of the form*

$$g(x) = h(|x \oplus u|), \tag{66}$$

where h is a strictly monotonic function have a fitness-dependent distribution for the bit-flip mutation operator and the probability matrix is the one defined in Equation (51).

PROOF: First, we observe that in the case of the sum of order-1 Walsh functions shown in Equation (45) the probability matrix $\boldsymbol{\varpi}^{(f)}$ does not change even in the case in which we compose the functions with a strictly decreasing function, since $q = n + 1$ and $\boldsymbol{\varpi}_{n-i,n-j} = \boldsymbol{\varpi}_{i,j}$ according to Proposition 7. Then, based on the results of Theorem 6 and Proposition 8 we only need to express $g(x)$ as a strictly monotonic function of the objective function f defined in Equation (45). This expression is:

$$g(x) = h(|x \oplus u|) = h\left(\frac{n - f(x \oplus u)}{2}\right). \tag{67}$$

□

A direct consequence of the previous result is that even though we focused in this section on the objective function defined in Equation (45) instead of the Onemax objective function, the probability matrix $\boldsymbol{\varpi}$ is valid also for the original Onemax function. Furthermore, it is also valid for any strictly monotonic function composed of the Onemax function.

4.2 MAX-SAT

The MAX-SAT problem is a well-known NP-hard problem related to the satisfiability of Boolean formulas. An instance of this problem is composed of a set of clauses C. A clause is a disjunction of literals, each one being a decision variable x_i or a negated decision variable $\overline{x_i}$. The MAX-SAT problem consists of finding an assignment of Boolean values to the literals in such a way that the number of satisfied clauses is a maximum. Let us assume that there exist n Boolean decision variables. For each clause $c \in C$ we define the vectors $v(c) \in \mathbb{B}^n$ and $u(c) \in \mathbb{B}^n$ as follows (Sutton et al., 2009):

$$v_i(c) = \begin{cases} 1 & \text{if } x_i \text{ appears (negated or not) in } c, \\ 0 & \text{otherwise,} \end{cases} \tag{68}$$

$$u_i(c) = \begin{cases} 1 & \text{if } x_i \text{ appears negated in } c, \\ 0 & \text{otherwise.} \end{cases} \tag{69}$$

We will omit the argument of the vectors (the clause) when there is no confusion. According to this definition $u \wedge v = u$. We should note here that the previous notation

allows us to express the empty clause, $\square$, with $v = u = 0$. But it is not possible to express the top clause $\top$. We will need a special treatment of the top clause in the following.

The objective function of MAX-SAT is defined as

$$f(x) = \sum_{c \in C} f_c(x); \quad \text{where}$$
$$f_c(x) = \begin{cases} 1 & \text{if } c \text{ is satisfied with assignment } x, \\ 0 & \text{otherwise.} \end{cases} \tag{70}$$

A clause c is satisfied with x if at least one of the literals is true (we assume the usual identity true $= 1$ and false $= 0$). Using the vectors $v(c)$ and $u(c)$ we can say that c is satisfied by x if $(\overline{x} \wedge u) \vee (x \wedge v \wedge \overline{u}) \neq 0$.

Sutton et al. (2009) provide the Walsh decomposition for the MAX-SAT problem. Let the function f_c evaluate one clause $c \in C$. The Walsh coefficients for f_c are:

$$a_w = \begin{cases} 0 & \text{if } w \wedge \bar{v} \neq 0, \\ 1 - \frac{1}{2^{|v|}} & \text{if } w = 0, \\ \frac{-1}{2^{|v|}} \psi_w(u) & \text{otherwise.} \end{cases} \tag{71}$$

If the clause c is $\top$ then the only nonzero Walsh coefficient is $a_0 = 1$.

For the sake of simplicity in the mathematical development, instead of using f_c in the following, it is better to use $g_c(x) = 1 - f_c(x)$. The Walsh coefficients for g_c are:

$$a_w = \begin{cases} 0 & \text{if } c = \top \text{ or } w \wedge \bar{v} \neq 0, \\ \frac{1}{2^{|v|}} \psi_w(u) & \text{otherwise.} \end{cases} \tag{72}$$

We will also focus on the fitness function $g(x)$ defined as:

$$g(x) = \sum_{c \in C} g_c(x) = m - f(x). \tag{73}$$

Maximizing $f(x)$ is equivalent to minimizing $g(x)$.

The following lemma provides the elementary landscape decomposition of g_c.

LEMMA 3: *The jth elementary component of g_c is*

$$g_{c,[j]}(x) = \left(1 - \delta_c^{\top}\right) \frac{1}{2^{|v(c)|}} \mathcal{K}^{(|v(c)|)}_{j,|(x \oplus u(c)) \wedge v(c)|}. \tag{74}$$

PROOF: With the help of Equation (72) we can write

$$g_{c,[j]}(x) = \sum_{\substack{w \in \mathbb{B}^n \\ |w| = j}} a_w \psi_w(x) = \left(1 - \delta_c^{\top}\right) \sum_{\substack{w \in \mathbb{B}^n \wedge v(c) \\ |w| = j}} \frac{1}{2^{|v(c)|}} \psi_w(u(c)) \psi_w(x)$$

by Equation (9)

$$g_{c,[j]}(x) = \left(1 - \delta_c^{\top}\right) \sum_{\substack{w \in \mathbb{B}^n \wedge v(c) \\ |w| = j}} \frac{1}{2^{|v(c)|}} \psi_w(x \oplus u(c))$$

by Equation (28)

$$g_{c,[j]}(x) = \left(1 - \delta_c^{\top}\right) \frac{1}{2^{|v(c)|}} \mathcal{K}^{(|v(c)|)}_{j,|(x \oplus u(c)) \wedge v(c)|}. \tag{75}$$

$\square$

The next two lemmas provide intermediate results related to the g_c functions that are required in the proof of the main theorem in this section.

LEMMA 4: *The mth power of g_c for $m > 0$ is:*

$$g_c^m(x) = g_c(x). \tag{76}$$

PROOF: The function g_c takes only values 0 and 1. If $m > 0$ we have $g_c^m(x) = g_c(x)$. □

LEMMA 5: *Given a family of clauses $c \in C$, the product of functions g_c is:*

$$\prod_{c \in C} g_c(x) = g_{\bigvee C}(x), \tag{77}$$

where $\bigvee C$ is the disjunction of the family of clauses. For the previous expression to be true even in the case in which the family of clauses is empty we define $g_\square(x) = 1$.

PROOF: The function g_c is 0 when the clause c is satisfied. Thus, a product of g_c functions will be 0 when any of the clauses is satisfied and 1 if none of the clauses is. This behavior is the same as the function g associated with the disjunction of the clauses. This disjunction is also another clause. □

The following theorem provides the expression for the matrix function $\boldsymbol{F}(x)$ for $g(x)$ defined in Equation (73).

THEOREM 7: *The matrix function $\boldsymbol{F}(x)$ for the objective function $g(x)$ defined in Equation (73) is:*

$$\boldsymbol{F}_{m,j}(x) = \sum_{\substack{W \subseteq C \\ \bigvee W \neq \top}} \frac{1}{2^{|v(\bigvee W)|}} \boldsymbol{\Upsilon}_{m,|W|} \mathcal{K}^{(|v(\bigvee W)|)}_{j,|(x \oplus u(\bigvee W)) \wedge v(\bigvee W)|}, \tag{78}$$

where the $\boldsymbol{\Upsilon}$ matrix is defined by the following recurrence equations:

$$\boldsymbol{\Upsilon}_{m,0} = 0, \tag{79}$$

$$\boldsymbol{\Upsilon}_{m,k} = k^m - \sum_{l=0}^{k-1} \binom{k}{l} \boldsymbol{\Upsilon}_{m,l}. \tag{80}$$

PROOF: Let us number the clauses in C from 1 to h and let us denote with c_j the jth clause. We can write $g^m(x)$ as

$$g^m(x) = \left(\sum_{j=1}^{h} g_{c_j}(x) \right)^m = \sum_{i_1 + i_2 + \cdots + i_h = m} \binom{m}{i_1, i_2, \ldots, i_h} \prod_{j=1}^{h} g_{c_j}^{i_j}(x)$$

by Lemma 4

$$= \sum_{i_1 + i_2 + \cdots + i_h = m} \binom{m}{i_1, i_2, \cdots, i_h} \prod_{\substack{j=1 \\ i_j > 0}}^{h} g_{c_j}(x)$$

removing the indices that are zero we can write the sum in an alternative way

$$= \sum_{W \subseteq C} \sum_{\substack{i_1 + i_2 + \cdots i_{|W|} = m \\ i_1, i_2, \cdots i_{|W|} > 0}} \binom{m}{i_1, i_2, \ldots, i_{|W|}} \prod_{c \in W} g_c(x)$$

$$= \sum_{W \subseteq C} \boldsymbol{\Upsilon}_{m,|W|} \prod_{c \in W} g_c(x)$$

by Lemma 5

$$= \sum_{W \subseteq C} \boldsymbol{\Upsilon}_{m,|W|} g_{\bigvee W}(x), \tag{81}$$

where we defined $\boldsymbol{\Upsilon}_{m,k}$ as

$$\boldsymbol{\Upsilon}_{m,k} = \sum_{\substack{i_1+i_2+\cdots+i_k=m \\ i_1,i_2,\ldots,i_k>0}} \binom{m}{i_1, i_2, \ldots, i_k}. \tag{82}$$

Using Equations (81) and (74) we obtain Equation (78).

In order to complete the proof we only need to justify Equations (79) and (80) based on the definition in Equation (82). In the following we will use the notation $[k]$ to denote the set of numbers from 1 to k. When $m, k > 0$ the sum of the multinomial coefficients is:

$$k^m = \sum_{i_1+i_2+\cdots+i_k=m} \binom{m}{i_1, i_2, \ldots, i_k}$$

which we can reorganize in the following way

$$\begin{aligned}
&= \sum_{\substack{J \subseteq [k] \\ J \neq \emptyset}} \sum_{\substack{i_1+i_2+\cdots+i_{|J|}=m \\ i_1,i_2,\ldots,i_{|J|}>0}} \binom{m}{i_1, i_2, \ldots, i_{|J|}} \\
&= \sum_{l=1}^{k} \sum_{\substack{J \subseteq [k] \\ |J|=l}} \sum_{\substack{i_1+i_2+\cdots+i_{|J|}=m \\ i_1,i_2,\ldots,i_{|J|}>0}} \binom{m}{i_1, i_2, \cdots, i_{|J|}} \\
&= \sum_{l=1}^{k} \sum_{\substack{J \subseteq [k] \\ |J|=l}} \sum_{\substack{i_1+i_2+\cdots+i_l=m \\ i_1,i_2,\ldots,i_l>0}} \binom{m}{i_1, i_2, \cdots, i_l} = \sum_{l=1}^{k} \sum_{\substack{J \subseteq [k] \\ |J|=l}} \boldsymbol{\Upsilon}_{m,l} \\
&= \sum_{l=1}^{k} \binom{k}{l} \boldsymbol{\Upsilon}_{m,l}.
\end{aligned} \tag{83}$$

In order to extend the previous sum to the value $l = 0$ we can just define $\boldsymbol{\Upsilon}_{m,0} = 0$ as in Equation (79) and we obtain the recurrence Equation (80). Now we can observe that Equation (80) is valid even in the case in which $k = 0$. □

By the definition of $\boldsymbol{\Upsilon}_{m,k}$ in Equation (82) it is clear that $\boldsymbol{\Upsilon}_{m,k} = 0$ if $k > m$. As a consequence, the sum in Equation (78) must consider only the subsets W of at most m elements if we are interested in the elementary landscape decomposition of the mth power of $g(x)$. The computation of the v and u vectors in Equation (78) can be done in $O(nm)$ at most, since we have to explore up to n bits of up to m clauses (it can be much less in practice if the number of literals per clause is low). Thus, the complexity of computing the elementary components of $g^m(x)$ is $O(nm|C|^m)$, where we assume that the matrices $\boldsymbol{\Upsilon}$ and $\mathcal{K}^{(n)}$ are precomputed.

5 Connection to Runtime Analysis

The results we present in this section represent, to the best of our knowledge, the first connection between landscape theory and runtime analysis, and this is, in fact, the reason why we think they are relevant. They are an application of the results of Sections 3 and 4 to the computation of the first hitting time of a $(1 + \lambda)$ EA. The results

Algorithm 1 Pseudocode of a $(1 + \lambda)$ EA

```
1: x ← RandomSolution();
2: while x is not a global optimum do
3:    for i = 1 to λ do
4:       y ← M_p(x);
5:       if f(y) ≥ f(x) then
6:          x ← y;
7:       end if
8:    end for
9: end while
```

themselves are not new or significant for the runtime community. We use the Markov chain framework by He and Yao (2003), which has important limitations when the goal is to find asymptotic bounds for runtime. With this framework we are able to compute exact expressions for the expected runtime as a function of p, the probability of flipping a bit in the mutation, but we are not able to find asymptotic expressions or make conclusions about the runtime when n is large, which is the main goal of the runtime analysis community.

When one is interested in computing bounds for the runtime required by an evolutionary algorithm to solve an optimization problem, it is quite common to analyze the probability of improving a solution in one iteration of the algorithm. This is the way, for example, in which an upper bound of $O(n \log n)$ is derived for the Onemax problem solved with a $(1 + 1)$ EA using bit-flip with probability $p = 1/n$ (Neumann and Witt, 2010, p. 39). These probabilities of improvement are not usually exactly computed, but an asymptotic lower bound of the probability is used instead. Thus, an upper bound of the expected runtime is derived instead of a precise expression.

In Section 3 we showed how we can compute the probability distribution of the objective values after mutation. In Section 4 we found that for a family of functions that includes Onemax, the probability distribution only depends on the value of the fitness function in the current solution and, thus, we could define a matrix $\boldsymbol{\varpi}$, which summarizes the behavior of the algorithm in one step. The question we want to answer in this section is, can we use the $\boldsymbol{\varpi}$ matrix to provide an expression of the expected runtime of an evolutionary algorithm? In the next sections we will show how a precise expression for the expected runtime of a $(1 + \lambda)$ EA can be derived using the $\boldsymbol{\varpi}$ matrix. In Algorithm 1 we show the pseudocode of a $(1 + \lambda)$ EA, where (abusing notation slightly) we use $M_p(x)$ to denote a random solution that is the result of applying the bit-flip mutation operator to x.

5.1 Runtime Analysis of $(1 + \lambda)$-EAs

Based on the probability matrix $\boldsymbol{\varpi}$, that only assumes one single application of the mutation operator, we can define a new probability matrix $\boldsymbol{\varpi}^{(\lambda)}$ related to the generation of λ offspring using bit-flip mutation and selecting the best one. This new probability matrix must reduce to $\boldsymbol{\varpi}$ when $\lambda = 1$. The element $\boldsymbol{\varpi}^{(\lambda)}_{i,j}$ is the probability of obtaining a solution with fitness value ξ_j after applying bit-flip mutation λ times with probability p to a solution with fitness value ξ_i and taking the offspring with the highest fitness. The following proposition provides an expression for $\boldsymbol{\varpi}^{(\lambda)}_{i,j}$.

PROPOSITION 10: *The probability matrix* $\boldsymbol{\varpi}^{(\lambda)}$ *is defined as*

$$\varpi_{i,j}^{(\lambda)} = \left(\sum_{l=0}^{j} \varpi_{i,l}\right)^{\lambda} - \left(\sum_{l=0}^{j-1} \varpi_{i,l}\right)^{\lambda}. \tag{84}$$

PROOF: The element $\varpi_{i,j}^{(\lambda)}$ is exactly the probability of obtaining at least one solution with fitness value ξ_j and no solution with a higher fitness value in the λ trials. This is exactly the probability of obtaining the λ solutions with fitness value lower than or equal to ξ_j (first term) minus the probability of obtaining the solutions with fitness value lower than ξ_j (second term). □

We can observe in Equation (84) that $\boldsymbol{\varpi}^{(1)} = \boldsymbol{\varpi}$. We must recall here that $\boldsymbol{\varpi}^{(\lambda)}$ is a polynomial in p because $\boldsymbol{\varpi}$ is also a polynomial in p. Now we analyze the runtime of the $(1 + \lambda)$ EA with the help of the Markov chain framework presented by He and Yao (2003). The first step is to present the transition matrix (we assume maximization):

$$\boldsymbol{P}_{i,j}^{(\lambda)} = \begin{cases} \varpi_{i,j}^{(\lambda)} & \text{if } j > i, \\ \sum_{l=0}^{j} \varpi_{i,l}^{(\lambda)} & \text{if } i = j, \\ 0 & \text{if } j < i, \end{cases} \tag{85}$$

where $0 \leq i, j < q$. If the probability p of flipping a bit is $0 < p < 1$, then the previous transition matrix will have only one absorbing state that corresponds to the solutions with the highest fitness value $\xi_{q\text{-}1}$.

Now we can use some results from the Markov chain theory (Iosifescu, 1980) to compute the expected runtime of the $(1 + \lambda)$ EA. The $\boldsymbol{P}$ matrix can be written in the form:

$$\begin{pmatrix} \boldsymbol{T} & \boldsymbol{R} \\ 0 & 1 \end{pmatrix}, \tag{86}$$

where $\boldsymbol{R}$ is a column vector and $\boldsymbol{T}$ is a $(q-1) \times (q-1)$ submatrix with the transition probabilities of the transient states in the Markov chain. The fundamental matrix is $\boldsymbol{N} = (\boldsymbol{I} - \boldsymbol{T})^{-1}$, and the expected runtime (number of iterations) of the $(1 + \lambda)$ EA starting in a solution with fitness value ξ_i and $0 \leq i < q - 1$ is given by the ith component of the vector of mean absorption times $\boldsymbol{t}_i$. This vector is computed as $\boldsymbol{t} = \boldsymbol{N}1$. From a computational point of view, the vector can be efficiently computed by solving the following linear equation system:

$$(\boldsymbol{I} - \boldsymbol{T})\boldsymbol{t} = 1, \tag{87}$$

since $\boldsymbol{I} - \boldsymbol{T}$ is an upper triangular matrix, the system can be solved in $O(q^2)$. The components of $\boldsymbol{t}$ will be, in general, fractions of polynomials in p. The vector $\boldsymbol{t}$ has only $q - 1$ components indexed by number from 0 to $q - 2$, but for the sake of completeness we can extend it with an additional component $\boldsymbol{t}_{q\text{-}1} = 0$, which is the expected runtime of the algorithm when the initial solution is the global optimum.

Assuming that the algorithm starts from a random solution, the expected runtime is given by

$$E\{\tau\} = \sum_{l=0}^{q-1} \frac{|X_l|}{|X|} \boldsymbol{t}_l, \tag{88}$$

where X is the set of solutions and X_l is the set of solutions with fitness $f(x) = \xi_l$.

The expected runtime in Equation (88) is not an approximation or bound, it is the exact expression of the expected runtime as a function of p, the probability of flipping a

bit. However, this expression will only be practical if: (1) we can define the $\boldsymbol{\varpi}$ matrix for the problem we are interested in, and (2) the evaluations of this matrix can be efficiently done on a computer. These conditions limit the number of problems whose runtime can be analyzed using this approach. However, we found in Section 4.1 that for any monotonic function of Onemax we can efficiently construct and evaluate the $\boldsymbol{\varpi}$ matrix. In the next section we focus on Onemax.

5.2 Runtime of $(1+\lambda)$ EA for Onemax

The Onemax problem has been studied in the literature on runtime analysis many times. Garnier et al. (1999) derived an expression for the transition probability matrix of the $(1+1)$ EA for the Onemax function that was later reported by He and Yao (2003). Their expression is the same as Equation (85) for $\lambda = 1$. Tight upper and lower bounds have been derived for the $(1+1)$ EA using different mutation rates. Recently, Witt (2013) proved that the $(1+1)$ EA optimizes all linear functions (including Onemax) in expected time $en\ln n + O(n)$, and the expected optimization time is polynomial as long as $p = O((\log n)/n)$ and $p = \Omega(1/\text{poly}(n))$. Jansen et al. (2005) proved that using a $(1+\lambda)$ EA the expected number of iterations after reaching the global optimum is $O(n\log n/\lambda + n)$. Doerr and Künnemann (2013) extended this result to linear functions. In summary, the Onemax problem is well known and the content of this section adds not too much to the current knowledge on this problem. The goal of this section is, thus, to obtain the same results from a different perspective, that of landscape analysis. The advantage of this approach is that with an exact expression of the expected runtime we can find a precise answer to some concrete questions for some particular instances. The disadvantage is that the expression is quite complex to analyze and we need to use numerical methods, so it is not easy to generalize the answers obtained.

Let us first start by studying the $(1+1)$ EA. Taking into account the $\boldsymbol{\varpi}$ matrix defined in Equation (51) for Onemax, the expected number of iterations can be exactly computed as a function of p, the probability of flipping a bit. Just for illustration purposes, we present the expressions of such expectation for $n \leq 3$:

$$E\{\tau\} = \frac{1}{2p} \qquad \text{for } n = 1, \tag{89}$$

$$E\{\tau\} = \frac{7-5p}{4(p-2)(p-1)p} \qquad \text{for } n = 2, \tag{90}$$

$$E\{\tau\} = \frac{26p^4 - 115p^3 + 202p^2 - 163p + 56}{8(p-1)^2 p\left(p^2-3p+3\right)\left(2p^2-3p+2\right)} \qquad \text{for } n = 3. \tag{91}$$

We can observe how the expressions grow very fast as n increases. The factor $p(p-1)$ is always present in the denominator for $n > 2$, what means that when p takes extreme values, $p = 0$ or $p = 1$, it is not possible to reach the global optimum from any solution, since the algorithm will keep the same solution if $p = 0$ or will alternate between two solutions if $p = 1$. However, when $n = 1$ the probability $p = 1$ is valid, and furthermore, it is optimal, because if the global solution is not present at the beginning we can reach it by alternating the only bit we have. In Figure 1 we show the expected runtime as a function of the probability of flipping a bit for $n = 1$ to 7. We can observe how the optimal probability (the one obtaining the minimum expected runtime) decreases as n increases.

Having the exact expressions we can compute the optimal mutation probability for each n by using classical optimization methods in one variable. In particular, for $n = 1$

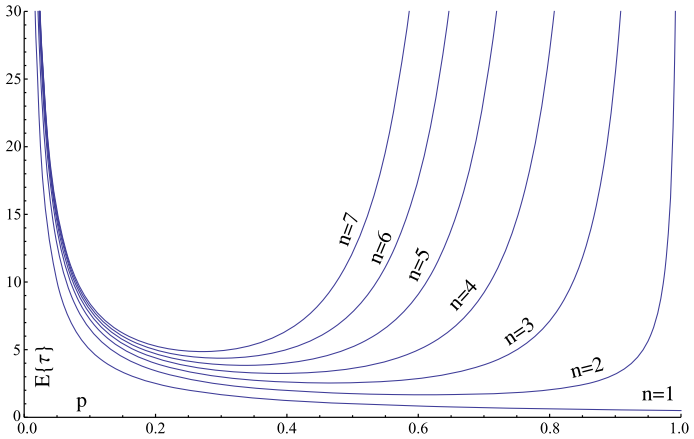


Figure 1: Expected runtime of the $(1+1)$ EA for Onemax as a function of the probability of flipping a bit. Each line corresponds to a different value of n from 1 to 7.

the optimal value is $p = 1$ as we previously saw and for $n = 2$ we have to solve a cubic polynomial in order to obtain the exact expression. The result is:

$$p_2^* = \frac{1}{5}\left(6 - \sqrt[3]{\frac{2}{23-5\sqrt{21}}} - \sqrt[3]{\frac{23-5\sqrt{21}}{2}}\right) \approx 0.561215, \tag{92}$$

which is slightly higher than the recommended value $p = 1/n$. Observe, however, that this result does not contradict the ones in Witt (2013), since Witt's work provides asymptotic expressions and discards low-order terms, while we are working here with expressions for low n values. In accordance with Witt's work, we would expect the optimal probability of mutation p_n^* to approximate the $1/n$ value, and this is what happens. As we increase n, analytical responses for the optimal probability are not possible and we have to apply numerical methods. In our case we used the Newton method in order to find a root of the equation $\frac{dE\{\tau\}}{dp} = 0$. Some results up to $n = 100$ can be found in Table 1. A fast observation of the results reveals that the optimal probability is always a little bit higher than the recommended $p = 1/n$.

Before we go further, we could question the use of an approximate method (the Newton method) over an exact expression to find the optimal probability for mutation. In particular, as we get approximate values anyway, why not directly run the $(1+\lambda)$ EA enough times to get an accurate enough approximate value for the optimal probability? Although in both cases we end with an approximated value, the approximation is done at a different level and using the Newton method, we get higher accuracy in less time. Let us explain this in detail. First, we have to say that given a probability of bit-flip p the computation of the first-hitting time using Equation (88) is very fast and the result we obtain is exact up to machine precision. If we want to obtain the expected first-hitting time running the algorithm we need to run it several thousand times to get a good confidence interval and the final approximation will be coarser than using the exact formulas. Just as an illustration we ran the algorithm 1,000 times for $n = 100$, $\lambda = 1$, and $p = 0.01$, and it took 193 s to find an expected runtime of 1058.60 ± 21.73 with 95% confidence. On the other hand the evaluation of the exact expression was done in 0.837 s and an expected runtime of 1069.54 with 11 decimal

Table 1: Optimal probability values for an $(1+1)$ EA solving Onemax.

n	p_n^*	$E\{\tau\}$	n	p_n^*	$E\{\tau\}$
1	1.00000	0.500	20	0.06133	127.453
2	0.56122	2.959	30	0.04046	222.079
3	0.38585	6.488	40	0.03009	325.900
4	0.29700	10.808	50	0.02391	436.580
5	0.24147	15.758	60	0.01981	552.734
6	0.20323	21.222	70	0.01690	673.445
7	0.17526	27.120	80	0.01473	798.059
8	0.15391	33.391	90	0.01304	926.088
9	0.13710	39.990	100	0.01170	1057.151
10	0.12352	46.882			

precision was found.[4] Second, if we want to obtain the optimal probability for mutation we have to apply a numerical method. Using the exact formulas, we applied the Newton method and after several steps we obtained an approximated optimal probability p^*. In order to get an optimal mutation probability using the completely empirical approach we need to again apply a numerical method such as the false position method (the Newton method cannot be applied now because we cannot compute derivatives). In order to evaluate each probability p we need to run the algorithm thousands of times and, in the end we can only obtain an approximated value for the expected runtime. As an illustration we can say that the execution of the Newton method for $n = 100$ stopped after 7.7 s, which is the time required to run the $(1+1)$ EA algorithm around 40 times in our machine. However, with 40 independent runs, the precision of the expected runtime is very low. In summary, the completely empirical method requires much more time than the Newton method applied to the exact formulas for the same precision.

From previous work we know that the optimal probability is in the form c/n for a constant c. We can use the results obtained by numerical analysis to find the value of c and check the dependency with n. That is, using the optimal probability p_n^* shown in Table 1 we can compute $c_n = p_n^* n$ in order to see what the value of c_n is. In Figure 2 we plot c_n as a function of n. We can observe that the optimal probability is not $p = c/n$ for a fixed c. The value of the constant c_n is higher than 1 and depends on n. This is not a surprise, since the optimality of $p = c/n$ has been proven in the asymptotic sense and we are analyzing low values for n. However, we can observe a clear trend $c_n \to 1$ as n tends to ∞. The maximum value for c_n is reached in $n = 11$ and the value is $c_{11} = 1.23559$.

It is also well known that for this optimal probability the expected runtime is $\Theta(n \log n)$. We can also check this using numerical analysis. We used the optimal expected runtime for this computation and found the best fit model including n and $n \log n$ terms. The result is:

$$E\{\tau\} \approx -1.51165n + 2.62161n \log n, \tag{93}$$

where we can observe how the factor in front of the $n \log n$ term is near e, which is the theoretically predicted factor for $c = 1$ and large values of n (Witt, 2013).

[4]The experiments were done in a MacBook Pro with an Intel Core i7 processor running at 2.8 GHz and 4 GB of DDR3 RAM memory.

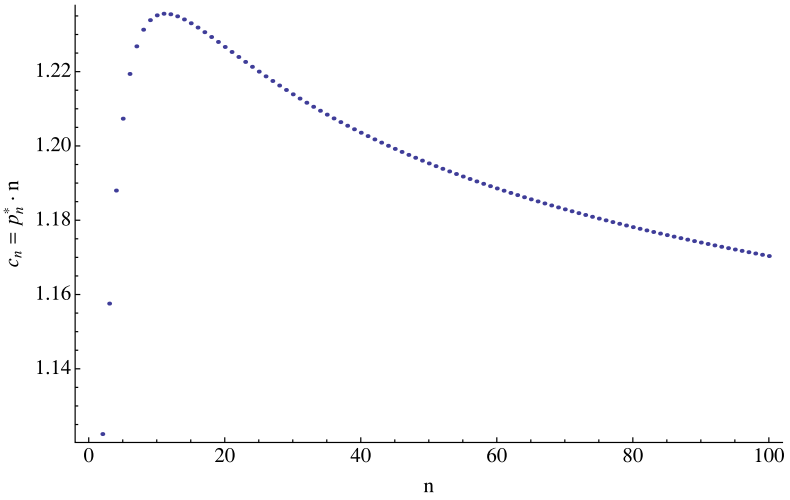
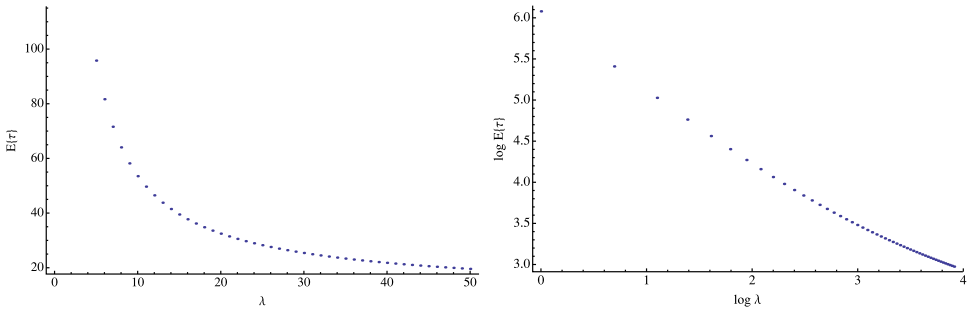


Figure 2: The value of the constant c in the optimal probability $p = c/n$ for Onemax instances from 2 to 100.



(a) Expected runtime as a function of λ (b) Log-log plot of the expected runtime against λ

Figure 3: The expected runtime of a $(1+1)$ EA with $p = 1/n$ and $n = 50$ for $\lambda = 1$ to 50.

Let us now study the expected runtime for different values of λ. In this case we fix the size of the problem to $n = 50$ and we analyze the expected runtime using $p = 1/n$ for $\lambda = 1$ to 50. The results are shown in Figure 3 in both natural scale and log-log scale.

The log-log plot is almost a straight line, which suggests that we can express the expected runtime as a potential function of λ. The best linear regression model for the log-log plot is:

$$\log E\{\tau\} \approx 5.78452 - 0.746412 \log \lambda, \tag{94}$$

or equivalently:

$$E\{\tau\} \approx \frac{325.226}{\lambda^{0.746412}}. \tag{95}$$

However, we cannot compare this model with the result of Jansen et al. (2005) of $O(n \log n/\lambda + n)$. Thus, we found the best fit model in the form $E\{\tau\} = A + \frac{B}{\lambda}$ and we got:

$$E\{\tau\} \approx 11.1306 + \frac{424.99}{\lambda}. \tag{96}$$

The value of the constant A is small enough to say that the expected runtime is approximately divided by λ when we generate λ offspring.

6 Conclusions

We analyzed the bit-flip mutation operator from the point of view of landscape theory. In particular, we derived closed-form formulas for all the statistical moments of the fitness distribution of a mutated solution. These moments can be expressed as a polynomial in p, the probability of flipping a bit. Using the moments we derived an expression for the probability mass function of the fitness value after applying bit-flip mutation to a given solution. The expression takes an elegant matrix form in which we can distinguish a problem-dependent part and an operator-dependent part. The problem-dependent part can be obtained using the elementary landscape decomposition of the objective function of the problem and their powers. The operator-dependent part depends only on the probability p.

We also derived the problem-dependent part for two well-known problems: Onemax and MAX-SAT. In the first case, the problem-dependent part is especially simple and efficient to compute. This allowed us to derive the exact expression for the runtime of an $(1 + \lambda)$ EA for solving Onemax, finding a connection between landscape theory and runtime analysis. Using this expression we obtained the optimal probability for bit-flip mutation as a function of n, the number of bits.

It is possible to analyze other operators in the same way we did with bit-flip mutation. Thus, we think that an interesting future line of research could be the application of similar ideas to find the probability mass function of the distribution after the application of several chained operators. In particular, recent developments in landscape theory suggest that it is possible to analyze the fitness distribution of the offspring of two parent solutions when uniform crossover is applied (Chicano et al., 2012), followed by the bit-flip mutation operator (Chicano et al., 2014). These results together with the connection between landscape theory and runtime analysis shown in this paper could provide a natural way of introducing crossover in the runtime results.

Acknowledgments

This work has been partially funded by the Spanish Ministry of Economy and Competitiveness and FEDER under contract TIN2011-28194 (the roadME project), VSB-Technical University of Ostrava under contract OTRI 8.06/5.47.4142, and by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number FA9550-08-1-0422. The U.S. government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation therein.

The authors would also like to thank the organizers and participants of the seminars on theory of evolutionary algorithms (10361 and 13271) at Schloß Dagstuhl-Leibniz-Zentrum für Informatik.

References

Biyikoglu, T., Leyold, J., and Stadler, P. F. (2007). *Laplacian Eigenvectors of graphs, Perroa-Frobenivs and Faber-Krahn type theorem*. *Lecture notes in mathematics*, Vol. 1915. Berlin: Springer-Verlag.

Chicano, F., and Alba, E. (2011). Exact computation of the expectation curves of the bit-flip mutation using landscapes theory. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, pp. 2027–2034.

Chicano, F., Whitley, D., and Alba, E. (2012). Exact computation of the expectation curves for uniform crossover. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, pp. 1301–1308.

Chicano, F., Whitley, D., and Alba, E. (2014). Exact computation of the expectation surfaces for uniform crossover along with bit-flip mutation. *Theoretical Computer Science*, 545:76–93, doi: 10.1016/j.tcs.2014.01.002

Chicano, F., Whitley, L. D., and Alba, E. (2011). A methodology to find the elementary landscape decomposition of combinatorial optimization problems. *Evolutionary Computation*, 19(4): 597–637.

Doerr, B., and Künnemann, M. (2013). How the (1 + λ) evolutionary algorithm optimizes linear functions. In *Proceeding of the 15th Annual Conference on Genetic and Evolutionary Computation Conference*, pp. 1589–1596.

Feinsilver, P., and Kocik, J. (2005). Krawtchouk polynomials and Krawtchouk matrices. In R. Baeza-Yates, J. Glaz, H. Gzyl, J. Hüsler, and J. Palacios (Eds.), *Recent advances in applied probability*, pp. 115–141. Berlin: Springer-Verlag.

Garnier, J., Kallel, L., and Schoenauer, M. (1999). Rigorous hitting times for binary mutations. *Evolutionary Computation*, 7(2):173–203.

Grover, L. K. (1992). Local search and the local structure of NP-complete problems. *Operations Research Letters*, 12(9):235–243.

He, J., and Yao, X. (2003). Towards an analytic framework for analysing the computation time of evolutionary algorithms. *Artificial Intelligence*, 145(1–2):59–97.

Iosifescu, M. (1980). *Finite Markov processes and their applications*. New York: John Wiley & Sons.

Jansen, T., De Jong, K. A., and Wegener, I. (2005). On the choice of the offspring population size in evolutionary algorithms. *Evolutionary Computation*, 13(4):413–440.

Mirsky, L. (1955). *An introduction to linear algebra*. Oxford, UK: Clarendon Press.

Neumann, F., and Witt, C. (2010). *Bioinspired computation in combinatorial optimization*. Berlin: Springer-Verlag.

Reidys, C. M., and Stadler, P. F. (2002). Combinatorial landscapes. *SIAM Review*, 44(1):3–54.

Stadler, P. F. (1995). Toward a theory of landscapes. In R. López-Peña, R. Capovilla, R. García-Pelayo, H. Waelbroeck, and F. Zertruche (Eds.), *Complex systems and binary networks* (pp. 77–163). Berlin: Springer-Verlag.

Sutton, A. M., Howe, A. E., and Whitley, L. D. (2010). Directed plateau search for MAX-k-SAT. In *Proceedings of the 3rd Annual Symposium on Combinatorial Search*, pp. 90–97.

Sutton, A. M., Whitley, D., and Howe, A. E. (2011a). Approximating the distribution of fitness over Hamming regions. In *Proceedings of the 11th Workshop Proceedings on Foundations of Genetic Algorithms*, pp. 93–104.

Sutton, A. M., Whitley, D., and Howe, A. E. (2011b). Mutation rates of the (1 + 1)-EA on pseudo-Boolean functions of bounded epistasis. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, pp. 973–980.

Sutton, A. M., Whitley, L. D., and Howe, A. E. (2009). A polynomial time computation of the exact correlation structure of k-satisfiability landscapes. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, pp. 365–372.

Sutton, A. M., Whitley, L. D., and Howe, A. E. (2012). Computing the moments of k-bounded pseudo-Boolean functions over Hamming spheres of arbitrary radius in polynomial time. *Theoretical Computer Science*, 425:58–74.

Terras, A. (1999). *Fourier analysis on finite groups and applications*. Cambridge, UK: Cambridge University Press.

Vose, M. D. (1999). *The simple genetic algorithm: Foundations and theory*. Cambridge, MA: MIT Press.

Walsh, J. L. (1923). A closed set of normal orthogonal functions. *American Journal of Mathematics*, 45(1):5–24.

Whitley, D., Sutton, A. M., and Howe, A. E. (2008). Understanding elementary landscapes. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, pp. 585–592.

Whitley, L. D., and Sutton, A. M. (2009). Partial neighborhoods of elementary landscapes. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, pp. 381–388.

Witt, C. (2013). Tight bounds on the optimization time of a randomized search heuristic on linear functions. *Combinatorics, Probability and Computing*, 22(2):298–314.