Applying Level Based Analysis to Optimal Mixing Evolutionary Algorithms.

Ivor van der Hoog

June 20, 2016

Abstract

In recent years a new variant of Drift Analysis was introduced in the field of computational complexity called Level Based Analysis. The Level Based Analysis provides a one-size-fits-all framework to analyze Evolutionary Algorithms. This paper analyzes how the Level Based Theorem works and applies it to GOMEA algorithms to get upper bounds on their computational time complexity. The Univariate GOMEA algorithm will be analyzed on the Onemax domain using the Level Based Theorem and more traditional methods. This paper will compare the use and bounds of the Level Based Analysis with these more traditional methods and supply loose upper bounds for more GOMEA algorithms on different domains as benchmarks for future work.

1 Introduction.

One aspect of computer science is finding algorithms to solve combinatorial optimization problems. In most of these problems *exhaustive search* is not feasible and so computer scientists develop smart algorithms to solve these problems within a reasonable time. A potent class of these algorithms are Evolutionary Algorithms and in recent years many EAs have been developed and published. Determining how fast such an EA solves a particular problem on a particular domain can be done in two ways: either through measurements or through calculating theoretical upper bounds. Most publications on the runtime of EAs provide measurements of the runtime of these algorithms on benchmark functions but theories that determine the runtime of these algorithms theoretically are rarely developed and applied [1]. In 2001 a general theory on determining the computational complexity time on EAs called Drift Analysis was presented by Jun He and Xin Yao in [2]. Following this work the Level Based Analysis was introduced by D-C Dang and P.K.Lehre in [4] and later improved in [5] and [6]. The Level Based Analysis claims to provide an easy-to-use framework for determining upper bounds for EAs and even Recombination Algorithms [8].
One class of EAs are the Genetic Optimal Mixing Evolutionary Algorithms introduced in [11]. These algorithms build a model from the population to determine how to execute crossover. This paper will analyze the Level Based Theorem in the context of GOMEA algorithms. It will apply the Level Based Theorem to the Univariate GOMEA algorithm on the Onemax domain to estimate an expected upper bound on the runtime. This paper will also use other methods to determine the upper bounds of the Univariate GOMEA algorithm and compare the use and bounds of the Level Based Analysis with these more traditional methods.
Section 2 will provide background information on EAs and combinatorial optimization problems and introduces the GOMEA algorithms. Section 3 will introduce the predecessors of the Level Based Analysis, Additive and Multiplicative Drift Analysis to introduce the concept of Drift. This section will also elaborate on the proof for both analyses provided in [2]. The first part of Section 4 introduces the Level Based Theorem and its recombination counterpart. This section will paraphrase and elaborate on the latest proofs for these theorems presented in [6]. The second part of section 4 will comment on the viability of the recombination adjustments of the Level Based Theorem and provide adjustments of our own to improve the quality-of-life when applying the theorem to GOMEA algorithms. The last part of section 4 will apply the adjusted theorem to the Univariate GOMEA algorithm on the Onemax domain. This paper also uses two other methods to analyze the Univariate GOMEA algorithm in section 5: Martingale theory, the source of inspiration for drift analysis, and a direct proof.

The functionality of the Level Based Analysis in comparison to these two more traditional methods is discussed in section 6. Loose upper bounds for other variants of the GOMEA framework on other domains are presented in the end as benchmarks for further research on their theoretical upper bounds.

2 Preliminaries.

This section will provide the mathematical definitions that are needed to determine theoretical upper bounds for the types of algorithms that we will analyze. Each subsection will cover a certain category of definitions through first explaining the intuition behind the categories and the definitions in the category and then stating their respective definitions.

2.1 Combinatorial optimization problems.

Intuitively, computational complexity is a theoretical indication for how long an algorithm takes to solve a certain problem of a certain size. If we want to determine a theoretical upper bound for how long an algorithm will take to solve a certain problem of a certain size, we first need a solid definition of what we see as a problem, problem size and what we see as a unit of time.
The introduction mentioned that we determine bounds for so called *combinatorial optimization problems*. These kind of problems are often an abstraction or model for a real life problem. We make a model for the solution of the real life problem, an abstraction of what a solution could look like. Any object that fills the axioms of this model could be the final solution hence we name it a *candidate solution*. Combinatorial Optimization problems create a finite set of these *candidate solutions* called the search space Ω. Algorithms search through these search space and test each candidate solution until they have found the solution to the problem, the optimal candidate solution. This intuition leads to the following definition:

Definition 1. *A **combinatorial optimization problem** is a problem which has a finite set of **candidate solutions** Ω called the **search space**. The problem is to find an optimal element $x^* \in \Omega$.*

There are usually few solutions to a problem so the number of optimal elements x^* is usually quite small. This makes it so that it is harder to find an optimal solution if the search space is larger. For this, we need some sort of indication on how large the solution space is and we call this the problem size $n \in \mathbb{N}$. The problem size is usually derived from the number of *problem variables* and formally defined when the problem itself or the solution space is defined. In all cases the problem size is a number $n \in \mathbb{N}$ *s.t.* $|\Omega| = f(x)$ *for a* $f \in C(\mathbb{R})$. Even for a computer, the search space is usually too large to just try out every existing solution. So algorithms that want to solve these kind of problems within a feasible time must search through the space in a smart way. The type of algorithms that will be discussed in this paper do this using a *fitness function*. A fitness function is a measurement for how 'good' a candidate solution is. A fitness function assigns to each candidate solution a *fitness value* to indicate how close the solution is to the optimal solution, the optimal solution then has a minimal of maximal *fitness value*. Usually a candidate solution has certain aspects it has to have to be the optimal solution. One can construct a fitness function through assigning each aspect a certain weight, the fitness value of candidate solution is then the sum of the weights of each aspect it has. It is clear that in this case an optimal solution has the maximal fitness value. The generic definition for fitness function is as follows:

Definition 2. *A **fitness function** is a function $f :: \Omega \to \mathbb{R}$ satisfying: $f(x^*) = min/max\{f(x) \mid x \in \Omega\}$.*

An example: A traditional example for a combinatorial optimization problem is the *knapsack* problem. The knapsack problem involves a container that can hold a certain amount of weight. Given a set of items, each with a weight and value, find the combination of items that fills the container that has an optimal value but is within the weight limit. One could define a fitness function f that sums the value of the items that are in the knapsack, and that is zero if the knapsack is too full. Knapsack itself involves finding an optimal combination but the domain of combinatorial optimization problems is not limited to only direct combination problems. Often existing problems can be redefined to become a combinatorial optimization problem. Such is the case with *constraint satisfaction problems* which are problems in which you have to comply to a certain set of constraints:

We will present $9x9$-Sudoku as an example. $9x9$-Sudoku becomes a combinatorial optimization problem when we view the conditions (constraints) that create a solution for the 9x9 Sudoku. In Sudoku you get a 9x9 matrix with certain numbers filled in at certain indexes. An (optimal) solution has each of the 81 fields filled with a number ranging from one to nine, no double numbers in each row, column or block of 3 by 3 and contains all the pre-determined numbers at all the predetermined indexes. We can construct a solution space Ω as the set of all the $9x9$ matrices with a number 1 to 9 assigned in each of the 81 fields. We can construct a fitness function f that first checks if all the predetermined field values are still present, if that is the case the fitness value of the candidate solution is the number of doubles in each row, column or block and else the fitness nears infinity. The optimal solution x^* is a solution that has all the fixated values and no doubles and hence it is the solution with the minimal fitness value: 0. We can also provide for a definition of problem size if we take a look at Sudoku in general. Sudokus have to have a dimension of $n^2 \times n^2$ with $n \in \mathbb{N}$ so that we can provide a valid number of blocks. The larger n is, the larger the search space Ω becomes and this makes n a good indicator for the problem size.

The last definition we need for determining the upper bound on the runtime of an algorithm is a definition of time. The faster an algorithm solves a problem, the better we consider the algorithm to be. One could try and determine upper bounds for the actual time in seconds. The time in seconds however is very dependent on what kind of hardware the algorithm is run on and how the tasks are scheduled. One could calculate that in, but it would give a very complex calculation. Other possible definitions for time such as number of clockticks or processor instructions are in unfeasible for the same reasons. Almost every attempt to solve a computational optimization problem makes use of a fitness function and generally speaking it takes quite some time to calculate the fitness value of a candidate solution. That is why we define the time an algorithm takes to run when talking about computational optimization problems as the number of fitness evaluation calls. The number τ of function evaluations required to find the optimal solution will be called a *Stopping time*. In this paper we will try to give a tight estimation for an upper bound of the stopping time for GOMEA algorithms.

2.2 Evolutionary Algorithms.

The GOMEA algorithms are a subset of Evolutionary algorithms. Evolutionary algorithms are stochastic algorithms for solving the earlier mentioned combinatorial optimization problems. Evolutionary algorithms are inspired by biological evolution and imitate the successful optimization strategy of combining and manipulating individuals in a population that we see in nature. There are many implementations of Evolutionary Algorithms but we will define a basic framework. Note that there can exist evolutionary algorithms that do not follow this framework but most of them do. Evolutionary Algorithms (EAs) work with a subset of candidate solutions called the population which we will denote as P. The standard framework has three possible tools: recombination, mutation and selection. The last one is noteworthy, if an EA makes use of selection we can speak of *generations*. Intuitively the selection operator takes a large set of candidate solutions and chooses a subset of candidate solutions to be the new population. After selection has happened we say we have a new *generation*. The size of the population at generation t (after applying the selection operator t times) will be denoted as λ_t, or λ if the size is constant for every generation. The notion of generations will suffice to define the operators:

- *Recombination.* The recombination step f_{recomb} selects a subset of an arbitrary size k of candidate solutions from the population as mates $f_{mat} :: P -> \Omega^k$ and then applies a crossover operator on those mates to generate a new candidate solution $f_{xor} :: \Omega^k \to \Omega$. $f_{recomb} := f_{xor} \circ f_{mat}$.
- *Mutation.* The mutation step applies a mutation operator to candidate solutions in the population $f_{mut} :: \Omega \to \Omega$ which transforms the candidate solutions into new candidate solutions.
- *Selection.* The selection step takes the new transformed population P'_t and selects candidate solutions to form the new generation P_{t+1} with a selection operator $f_{sel} :: \Omega^{|P'_t|} \to \Omega^{\lambda_{t+1}}$. We call the algorithm *elitist* if the selection operator does not select candidate solutions with a lower fitness value than the previous generation for the next generation.

Evolutionary algorithms are stochastic, meaning that there is a certain amount of chance involved. Because the process of an evolutionary algorithm is a stochastic process, we can solidify the earlier definition of *stopping time* τ with the definition of stopping time for Markov Chains in [9].

Definition 3. *Given a stochastic process* $X := \{X_n \mid n \in \mathbb{N}\}$. *If we were to observe the values* $X_0, X_1 \ldots$ *We define the* ***total information up to*** τ *to be all the information contained in* $\{X_0 \ldots X_\tau\}$.

Definition 4. *Let* $X := \{X_n \mid n \in \mathbb{N}\}$ *be a stochastic process. A* ***stopping time with respect to*** $\boldsymbol{X}$ *is a random time* τ *such that for each* $n \in \mathbb{N}$ *the event* $\tau = n$ *is completely determined by the* ***total information up to*** $\boldsymbol{n}$.

We will give examples of stopping times using the following case: assume you and an opponent both have 5 dollars. You flip a coin and if it's head you get a dollar from your opponent, else vice versa.

- Playing until five games are played will introduce a stopping time.
- Playing until one player has reached 3 dollar is a stopping time.
- Playing until you have the largest lead you'll ever have over the other is not a stopping time since it requires you to use information from the future.

Definition 5. *Given two stopping times* τ_1 *and* τ_2, ***combined stopping time*** *denoted as* $\tau_1 \vee \tau_2$ *is the event that either* τ_1 *occurs or* τ_2.

Corollary 1. Combined stopping time *is also a stopping time.*

Earlier we stated the number τ of function evaluations required before the algorithm is done will be called a *stopping time*. We can show that τ satisfies the formal definition of a *stopping time*. This is because the algorithm terminates when an optimal solution is found hence if $\tau = n$ then $X_n = \{x^* \in P_n\} \cup \{x^* \notin P_t \mid 0 \leq t < n\}$, making the state only dependent on the previous states.

The definition of stopping time for an evolutionary algorithm gives rise for a definition the runtime of that EA. We calculate the runtime of an upper bound of an algorithm by giving an upper bound for the expected stopping time of an algorithm. This expected upper bound $E[T]$ is an upper bound for the expected number of function calls that have to be done before one first encounters the optimum. It's interesting to note that there are two ways of estimating an upper bound for an Evolutionary Algorithm. One can analyze the exact effect of the algorithm on the specified domain to give an estimation for how long the algorithm would take. This however, can never be a generic approach because you look at the specific actions of the algorithm in the search space. The other way is to utilize that Evolutionary Algorithms are stochastic processes and make use of that process to provide upper bounds in a more generic way. Traditionally this is done by creating a Markov chain model [10], but because absorbing Markov chains can be a daunting task this is not very desirable. The Drift Analysis and the Level Based Theorem that is new in this category tries to model the stochastic process in a way that has its origins in *Martingale Theory*.

2.3 GOMEA Algorithms.

This paper will analyze the runtime of Gene-pool Optimal Mixing Evolutionary Algorithms introduced in [11] using the Level Based Theorem and more. GOMEA algorithms make use of a *FOS structure*, where FOS stands for Family Of Subsets. Evolutionary algorithms traditionally have a separate recombination, mutation and selection operator (each of those can be the identity function). GOMEA algorithms work different from Evolutionary algorithms because they combine the recombination and mutation phase. GOMEA algorithms are a subset of what we call Model-Based Evolutionary Algorithms. These algorithms create an effect similar to recombination and mutation through learning a model of the population from the mutual information between the candidate solutions in the population. This model is then used to combine elements to generate new offspring. The model usually takes the form of a family of subsets of the problem variables. The key to constructing such a model lies in identifying groups of problem variables that together make a grand contribution to the quality of the solutions.

Definition 6. *Let there be* n *problem variables and* S *be the set of the indexes of those variables:* $\{0, 1, ..., n-1\}$. *We define a* ***FOS-structure*** $\mathcal{F} \subset \mathcal{P}(S)$ *as a subset of the power set of* S *with the restriction that* $\forall x \in S, \exists F \in \mathcal{F} \quad s.t. \quad x \in F$.

In GOMEA algorithms the FOS-structure specifies the linked variables of the solution. A subset of the FOS is used as a traditional crossover mask. In this case, the crossover is *greedy*, accepting only improvements or solutions that are equally good or better. The algorithm tries to generate new better solutions by selecting a parent solution p and traversing the FOS structure and recombining accordingly in a greedy way. For every subset $F \in \mathcal{F}$, we select a new donor d from the population and we copy all the values of the indexes in F from d to p.

Algorithm 1 GOMEA

```
1: procedure GOMEA
2:     population ← initializePop()
3:     while NotTerminated(population) do
4:         FOS ← BuildFOS(population)
5:         for all p ∈ population do
6:             for all subset ∈ FOS do
7:                 donor ← Selection(population)
8:                 p ← GreedyComb(p, donor, subset)
       return GetSolution(population)
```

Algorithm 2 GreedyComb

```
1: procedure GreedyComb(parent, donor, subset)
2:     newSol ← replaceSubset(parent, donor, subset)
3:     if isImprovement(newSol, parent) then
4:         parent ← newSol
       return parent
```

2.3.1 FOS-structures used.

We will introduce three FOS-structures of which we will use two in the analysis of the algorithms.

Definition 7. *The **Univariate structure** on an index set S is defined as:* $\mathcal{F} := \{\{i\} \mid i \in S\}$.

It is the set of singleton indexes, making every variable independent of the other variables. GOMEA with this structure strongly resembles doing $(\lambda + \lambda)$-EA.

Definition 8. *The **Marginal Product structure** on an index set S is defined as:*
$\mathcal{F} \subset \mathcal{P}(S)$ *where* $\forall i \in S, \quad \exists F \in \mathcal{F}$ *s.t.* $i \in F$ *and* $\forall F_1, F_2 \in \mathcal{F}, \quad F_1 \cap F_2 = \emptyset$

In the Marginal Product structure every group of variables is independent of all the other groups. We will not give bounds using this structure. The last structure is what we call a Linkage Tree structure:

Definition 9. *The **Linkage Tree structure** on an index set S is defined as:*
$\mathcal{F} \subset \mathcal{P}(S)$ *where* $\forall F \in \mathcal{F}$ *with* $|F| > 1, \quad \exists F_1, F_2 \in \mathcal{F}$ *with:*

- $F_1 \neq \emptyset$
- $F_2 \neq \emptyset$
- $F_1 \cap F_2 = \emptyset$
- $F_1 \cup F_2 = F$

The idea of a Linkage Tree is that within every subset the problem variables are linked but that they have 'child' subsets where the problem variables become unlinked. For a problem with length n the Linkage Tree has n leaf nodes consisting of singleton indexes making the univariate structure a subset of the Linkage Tree structure. When using a Linkage Tree structure, we traverse the tree *top down*.

2.3.2 The domain.

The last thing left to define is the domain that we will analyze the Univariate GOMEA on. We will use the Univariate algorithm to optimize bitstrings of length n. For a given individual x and index i with $0 \leq i < n$ we denote $x[i]$ as the value found in x at the index i. We initiate an individual by traversing all indexes i and making $x[i]$ a one with chance $\frac{1}{c}$ and a zero with chance $1 - \frac{1}{c}$. We call a string optimal if the string consists out of all ones. In general, a fitness function determines the 'shape' of the search space. The fitness function must indicate how close a candidate solution is to the optimum and how well a fitness function can reflect that distance steers how well an algorithm can find a solution. A fitness function is an integral part of the domain that the algorithm works on. The domain of optimizing strings of length n is often taken as a benchmark for algorithms and there are a few known benchmark fitness functions on this domain [13]:

Definition 10. *The **Onemax** fitness function simply counts how many ones are in the string. It is defined as* $f_{Onemax}(x) := \sum_{i=0}^{n} x[i]$.

Definition 11. *The **Leading Ones** fitness function (often shortened as LO) counts how many consecutive ones there are starting from index zero. It is defined as* $f_{LO}(x) := \sum_{i=0}^{n} \prod_{j=0}^{i} x[j]$.

Definition 12. *The **k-plateau**. Given an integer k smaller than n and a fitness function f_{base}, k-plateau changes the fitness function for a fitness interval I_k of size k, in that interval the fitness value is constant. The value of $f_{plateau}(x)$ is based on the value of $f_{base}(x)$:*

- $f_{base}(x) < L \rightarrow f_{base}(x)$
- $L \leq f_{base}(x) < L + k \rightarrow L$
- $f_{base}(x) \geq L + K \rightarrow f_{base}(x)$

The domains in this paper. This paper will focus on determining upper bounds using the simplest FOS structure to analyze, the univariate structure on the Onemax domain. Loose upper bounds for the Linkage Tree Genetic Algorithm will be provided near the end of this paper and also bounds on different domains. We think it would be valuable future research to determine tight theoretical upper bounds for those variants.

3 Drift analysis.

The Level Based Theorem is part of Drift Analysis. In probability theory, stochastic drift is the change of the average value of a stochastic process. Drift analysis took the core concept of stochastic drift and the theorems that came with it and transformed it into a framework for computational fitness. We first note that Drift Analysis makes two assumptions:

- The optimization problem is a minimization problem:
  a solution x^* is optimal if $f(x^*) = min\{x \mid \Omega\}$.
- The optimal solution has fitness zero.

If the optimization problem is not of this form, we can transform the fitness function so that it does become of this form. For instance, if the problem is a maximization problem we can define a new fitness function f_{new} to be $f_{new}(x) = MAX - f(x)$ to satisfy the axioms. In Drift analysis we determine the *fitness of the population* and we define drift of generation t as the difference in fitness between generation t and generation $t - 1$.

Definition 13. *The **fitness of a population** P, given a minimizing fitness function f is defined as* $f(P) := min\{f(x) \mid x \in p\}$.

Definition 14. *The **drift** of a population at generation t denoted as Δ_t or $\Delta(P_t)$ is the difference in fitness between P_t and P_{t-1}, $\Delta_t := f(P_{t-1}) - f(P_t)$. Drift can either be towards the goal if $\Delta_t \geq 0$ or away from the goal. In the latter case the drift has a negative impact on the solution and hence we call it **negative drift**.*

The first theorem in the Drift analysis was later called *Additive Drift* and presented in [7]. This theorem denotes the expected number of function evaluations that the algorithm will do before finding a global optimum at E[T]. It states that if you model the progress of an EA as a stochastic process, you can get an upper bound for E[T] the following way:

Theorem 1. Additive drift.
If you know an upper bound for the expected starting fitness of the population s_{max} and if you know a lower bound for the expected positive drift $\forall t$, $\Delta_t \geq z^*$. Then the expected number of timesteps before you have found an optimal solution is upper bound by $E[T] \leq \frac{s_{max}}{z^*}$.

This theorem uses a concept of basic physics: if you have a known maximal distance and a known minimal speed, you can compute a known maximal time by dividing the former by the latter. The additive theorem is however too unwieldy to be applied directly. Often the lower bound for finding an improvement is much lower than the actual odds of finding an improvement when running the algorithm. Using a too low lower bound for z^* will result in a too large upper bound for the runtime of the algorithm.

An improvement of the additive drift theorem is the Multiplicative Drift Theorem presented in [3]. The Multiplicative Drift tries to overcome the weak spot of the Additive Drift, picking a too low bound for improvement and thus not getting tight bounds. Many Evolutionary Algorithms struggle to find improvements in the beginning but as they progress they get a higher chance to make an improvement. Other algorithms (including the Univariate GOMEA) work exactly the other way around with a higher chance to find improvements early and later experiencing a rough end. Additive drift cannot accurately model this convergence behavior as it uses the lowest bound for improvement. Multiplicative drift is designed for these kind of algorithms. Multiplicative drift states that if the lower bound for improvement is a multiple of the current fitness, better bounds can be found.

Theorem 2. Multiplicative Drift.
Denote the stopping time as T. Denote j_{min} to be $min\{f(P_t) \mid t < T\}$ (this is possible since a finite set always has a minimum). Denote $J := \{j \in \mathbb{R} \mid Pr(f(P)) = j > 0\}$ as the set of all distances we could ever obtain. If for all generations, for all of those values j the expected drift is higher than delta times j:

$$E[\Delta(P_t) \mid f(P_t) = j] > \delta \cdot j$$

If we start at a distance $f(P_0)$ then the expected stopping time is bounded by:

$$E[T] \leq \frac{1 + \ln(\frac{f(P_0)}{j_{min}})}{\delta}$$

The idea behind the proof presented in [3] is that you introduce an extra concave function g called the *potential function*. This function transforms the values of the fitness function f, $g :: f(\Omega) \to \mathbb{R}$. This function will also have a minimum at the optimum but its extra properties allow for tighter bounds when using the additive drift theorem. We will provide a paraphrased version of the proof presented in [3], shortening the algebra and adding commentary on why certain decisions are made. This will be done to make the use of a potential function intuitive because the Level Based Theorem in this paper makes extensive use of that concept:

3.1 The proof

Denote the search space Ω. If we know the fitness function $f :: \Omega \to \mathbb{R}^+$, we make a function $g :: f(\Omega) \to \mathbb{R}$ which sends the distances to another number via: $g(j) = 1 + ln(\frac{j}{j_{min}})$. Now we denote:

$$Z^{(t)} = \begin{cases} 0 & if \quad f(P_t) = 0 \\ g(f(P_t)) & else \end{cases}$$

Basically, Z is the same function as g but now has a minimum at 0 instead of a negative minimum so that the additive drift theorem can be applied. We now derive a lower bound for the *drift* of Z for each generation t:

$$Z^{(t-1)} - Z^{(t)} \geq \frac{\Delta(P_t)}{f(P_{t-1})} \tag{1}$$

In order to see that this is valid, we view two cases:
First we have the case that t is the last step: $t = T$. Then the optimum is found in t so Z^t is zero, making the left side equal to $1 + ln(\frac{f(P_{t-1})}{s_{min}})$. For the right hand side we note that we are at the last step $t = T$ so the drift has to cross the remaining distance, $\Delta(P_t) = f(P_{t-1})$. This makes the right hand side equal to 1. 1 is lesser or equal to 1 with a positive log.
The second case is that t is not the last step. Then $Z^{t-1} - Z^t = ln(\frac{f(P_{t-1})}{f(P_t)})$. The theorem requires a positive lower bound for the drift so we know that $f(P_{t-1}) > f(P_t)$ for all t and we use that in the following trick:

$$\begin{aligned} 1 > \frac{f(P_t)}{f(P_{t+1})} \Rightarrow \\ \frac{f(P_t)}{f(P_{t+1})} = 1 + \frac{f(P_t) - f(P_{t-1})}{f(P_{t-1})} \leq EXP\left[\frac{f(P_t) - f(P_{t-1}))}{f(P_{t-1})}\right] \Rightarrow \\ \ln\left(\frac{f(P_t)}{f(P_{t-1})}\right) \leq \frac{f(P_t) - f(P_{t-1}))}{f(P_{t-1})} \Rightarrow \\ Z^{t-1} - Z^t = \ln(\frac{f(P_{t-1})}{f(P_t)}) \geq \frac{f(P_{t-1}) - f(P_t)}{f(P_{t-1})} = \frac{\Delta(P_t)}{f(P_{t-1})} \end{aligned}$$

We now know that (1) holds. Now note that $E\left[\Delta(P_t) \mid f(P_t) = j\right] > \delta \cdot j$, substituting with 1 gives a lower bound on the drift $E\left[Z^{t-1} - Z^t\right] > \delta$:

$$E\left[\Delta(P_t)\right] > \delta \cdot f(P_t) \Rightarrow E\left[\frac{\Delta(P_t)}{f(p_t)}\right] > \delta \Rightarrow E\left[Z^{t-1} - Z^t\right] > \delta$$

Having a minimal value for the drift, the additive drift theorem now states that:

$$E\left[T \mid f(P_0) = j\right] \leq \frac{g(j)}{\delta} = \frac{1 + \ln(\frac{j}{j_{min}})}{\delta} \quad \square$$

Applying the Multiplicative drift theorem. The multiplicative drift theorem is not directly applicable to our problem, solving a Onemax bit problem with the Univariate GOMEA algorithm. This is because it is hard to derive a multiplicative lower bound for the odds of improving. Assuming a parent p has fitness j, then the parent can find an improvement if for any index i where $p[i] = 0$ it matches with a donor d with $d[i] = 1$. The odds of that happening for a specific index i are lower bound by $\frac{1}{c}$. We will not find an improvement if we select a donor which has a zero for each of the zero indexes making $z_j = (1 - \frac{1}{c})^{n-j}$. One can see that the odds of finding an improvement decrease exponentially and not by a multiplicative factor. That makes multiplicative drift not suitable for analyzing GOMEA algorithms and thus we must resort to the more advanced form, Level Based Analysis.

4 Level Based Analysis

4.1 The Level Based Theorem

The main theorem of this paper is the *Level Based Analysis* theorem. The Level Based Analysis comes forth from Drift Analysis and was first published in [4]. The idea of the level based analysis is that we divide the possible fitness values into *levels* and that we divide the search space into sets A_j according to those levels. We say that an entire population is at a certain level j if at least γ_0 of the population has a fitness that belongs to level A_j or higher. The analysis looks at two aspects of the Evolutionary Algorithm it analyzes: The first aspect is described in the first axiom $G1$ where we determine a lower bound z_j for the odds of generating offspring in a level higher than the current level j. The second aspect is modeled by $G2$ where we create a lower bound $\gamma(1 + \delta)$ for the odds of generating offspring in a level higher than the current level given that a percentage γ of the population is already in a higher level than j. These two aspects are the two aspects of an EA, exploration versus exploitation and the weight of both aspects can be tuned in the variables γ_0 and the later presented δ. The proof uses these lower bounds and just like the multiplicative drift theorem a potential function g. The theorem also requires the population to have a certain size in order for the algebraic tricks in the proof to work.

A quick note: There is one last requirement for the Level Based Theorem that is not mentioned in the definition of the the theorem but is vital for the proof: the odds of sampling a new individual with a certain level form a binomial distribution, dependent on the level of the current population. We denote the distribution from which we sample new individuals based on the current population P as $D(P)$.

Theorem 3. Level Based Theorem.
Given a problem with problem size n, a partition $\{A_0, A_1, ..., A_m\}$ of the search space Ω and $A_j^+ := \{A_k \mid k > j\}$, and given an algorithm with population size λ and a population per generation P_t, define the stopping time $T := min\{t\lambda \mid |P_t \cap A_m| > 0\}$ to be the first time (with time in the number of function calls) that an optimal element in appears in P_t.
If there exist $z_0, ...z_m, \delta \in (0, 1]$ and $\gamma_0 \in (0, 1)$ such that for any population P and all $\gamma \in (0, \gamma_0)$ holds:

G1) $\forall j \in [n], \forall t$ *if* $|P_t \cap A_{j-1}^+| \geq \gamma_0 \lambda$ *then* $Pr(y \in A_j^+ \mid y \in D(P_t)) \geq z_j$
This axiom describes a lower bound on the chance z_j to sample an individual in a level higher than the current level j.

G2) $\forall j \in [n-1], \forall t$ *if* $|P_t \cap A_{j-1}^+| \geq \gamma_0 \lambda$ *and* $|P_t \cap A_j^+| \geq \gamma \lambda$ *then* $Pr(y \in A_j^+ \mid y \in D(P_t)) \geq (1+\delta)\gamma$
This axiom models the influence of the selection operator with a parameter δ. It provides a lower bound on the chance to sample an individual in a level greater than the current level, given that γ of the population is already in a level higher than j.

G3) *For* $z_* := min\{z_j \mid j \in [m]\}$ *the population size λ satisfies:* $\lambda \geq \left(\frac{8}{\gamma_0 \delta^2}\right) \ln\left(\frac{128n}{\gamma_0 \delta^3 z_*}\right)$
This axiom notes that given the parameters γ_0, z_j and δ, we have a lower bound on the size of the population λ so that we can assume in the proof that we never drop a level.

then $E[T] \leq \left(\frac{454}{\delta^{\frac{9}{2}}}\right)\left(n\lambda(1 + \ln(1 + \frac{\delta^{7/2}\lambda}{203})) + \sum_{j \in [n]} \frac{1}{z_j}\right)$

We mention a quick lemma that is needed for the proof and immediately follows out of the definition of the theorem. This lemma states that if we are at level j, the odds of sampling an individual at level j or higher are greater or equal than $(1+\delta)\gamma_0$.

Lemma 1. $|P_t \cap A_{j-1}^+| \geq \gamma_0 \lambda \Rightarrow Pr(y \in A_{j-1}^+ \mid y \in D(P_t)) \geq (1+\delta)\gamma_0$

Proof: We simply note that $G2$ applies: if we are at level j then by definition γ_0 of the population is at level j or higher and filling in $\gamma = \gamma_0$ we get these lower bounds.

4.2 The proof

The proof is a long proof and so we will first provide a sketch of the proof. Just as with multiplicative drift, the proof defines a potential function g. After proving some attributes of the potential function we try to calculate the expected drift. We do this by splitting the drift into drift in the positive direction and drift in the negative direction and use the needed size of the population to show that the drift in the negative direction is negligible. Because the negative drift is negligible, we only have to derive a lower bound for the positive drift to be able to apply the Additive Drift Theorem and get to our bounds.

4.2.1 The potential function.

Definition 15. *For any integers $\lambda, m > 0$, a function $g :: [\lambda] \times [m+1] \to \mathbb{R}$ is called a* ***level function*** *if:*

1) $\forall (x,y) \in [\lambda] \times [m], \quad g(x,y) \geq g(x,y+1)$

2) $\forall (x,y) \in [\lambda - 1] \times [m+1], \quad g(x,y) \geq g(x+1,y)$

3) $\forall y \in [m], \quad g(\lambda, y) \geq g(0, y+1)$

These definitions ensure that g decreases monotonously with y. The axioms also result in two lemmas we are going to be using:

Lemma 2. *The sum of two level functions is also a level function.*

The proof of this lemma is trivial.

Lemma 3. $\forall y_1, y_2 \in [m+1]$ *with* $y_2 \geq y_1, \quad \forall x_1, x_2 \in [\lambda], \quad g(x_2, y_2) \leq g(x_1, y_1)$

Proof: If $y_2 = y_1$ the proof is trivial, if $y_2 > y_1$ the proof is a concatenation of axioms: first we note that $g(x_2, y_2) \leq g(0, y_2)$ because of (2). Then we note that $y_2 \leq y_1 + 1$ (because $y_2 > y_1$ implies that the difference between them is at least 1 since they are both integers). From this it follows that $g(0, y_2) \leq g(0, y_1 + 1)$ and (3) states that $g(0, y_1 + 1) \leq g(\lambda, y_1)$. Then we note that $x_1 \leq \lambda$ and thus according to (2) $g(\lambda, y_1) \leq g(x_1, y_1) \quad \square$

We denote for all generations t: $X_t^j := |P_t \cap A_{j-1}^+|$ as the number of individuals in the level j or higher. Note that we sample each individual per generation $t+1$ independently from the distribution $D(P_t)$. Let $p_{t+1}^j := Pr(y \in A_{j-1}^+ \mid D(P_t))$ be the probability of sampling an individual in level j or higher for generation $t+1$. It follows that X_{t+1}^j is binomially distributed with chance p_{t+1}^j.

$$X_{t+1}^j \sim Bin(\lambda, p_{t+1}^j)$$

Given the level constant γ_0, we define Y_t as:

$$Y_t := min\{j \in [m] \mid X_t^{j+1} < \gamma_0 \lambda\}$$

If we are at level j and $X_t^{j+1} = 0$ then *(G1)* states that $p_{t+1}^{j+1} \geq z_j$. Moreover, if for a $\gamma \in (0, \gamma_0)$ we have $X_t^{j+1} \geq \gamma\lambda$ then *(G2)* states that p_{t+1}^{j+1} is also greater or equal to $\gamma(1+\delta)$.
We define the distance function as: $g(X_t^{Y_t+1}, Y_t) := g_1(X_t^{Y_t+1}, Y_t) + g_2(X_t^{Y_t+1}, Y_t)$ with:

$$g_1(k,j) := (m-j)\ln(1 + \frac{\delta^{\frac{7}{2}}\lambda}{144\sqrt{2}}) - \ln(1 + \frac{\delta^{\frac{7}{2}}k}{144\sqrt{2}})$$

$$g_2(k,j) := \frac{1}{1-(1-z_j)^\lambda}\left(1 - \frac{\delta^{\frac{9}{2}}\lambda}{355}\right)^k + \sum_{i=j+1}^{m} \frac{1}{1-(1-z_j)^\lambda}$$

g_1 will be used to provide a lower bound for the expected drift when the number of individuals in the population with a higher level than the current level Y_t is greater than zero, else we use g_2.

Lemma 4. *For any constant $c > 0$ and any integers λ and m, and for $x \in [\lambda]$ and $y \in [m]$, the function $g_1'(x,y) := (m-y)\ln(1+c\lambda) - \ln(1+cx)$ is a level function.*

Proof: We will prove the lemma by proving all the axioms:

1. $$\forall (x,y) \in [\lambda] \times [m], \quad g_1'(x,y+1) - g_1'(x,y) =$$
$$(m-y-1)\ln(1+c\lambda) - \ln(1+cx) - (m-y)\ln(1+c\lambda) + \ln(1+cx) = -\ln(1+c\lambda) < 0$$

2. $$\forall (x,y) \in [\lambda - 1] \times [m+1], \quad g_1'(x+1,y) - g_1'(x,y) =$$
$$(m-y)\ln(1+c\lambda) - \ln(1+c(x+1)) - (m-y)\ln(1+c\lambda) + \ln(1+cx) = \ln(\frac{1+cx}{1+cx+c}) < 0$$

3. $\forall y \in [m], \quad g_1'(\lambda, y) = (m-y)\ln(1+c\lambda) - \ln(1+c\lambda) = (m-(y+1))\ln(1+c\lambda) = g_1'(0,y+1)$ $\square$

Lemma 5. *For any number $c > 0$ and any integers λ and m, and for $x \in [\lambda]$, $y \in [m]$ and $k \in (0,1)$ and any $q_j \in (0,1]$, the function $g_2'(x,y) := \frac{(1-k)^x}{q_y} + \sum_{i=y+1}^{m} \frac{1}{q_i}$ is a level function.*

Proof: The proof will be done in the same way:

1. $\forall (x,y) \in [\lambda] \times [m], \quad g_2'(x,y+1) - g_2'(x,y) =$
$$\frac{(1-k)^x}{q_{y+1}} + \sum_{i=y+2}^{m} \frac{1}{q_i} - \frac{(1-k)^x}{q_y} - \sum_{i=y+1}^{m} \frac{1}{q_i} =$$
$$\frac{(1-k)^x}{q_{y+1}} - \frac{(1-k)^x}{q_y} - \frac{1}{q_{y+1}} < 0$$

2. $\forall (x,y) \in [\lambda] \times [m], \quad g_2'(x+1,y) - g_2'(x,y) =$
$$\frac{(1-k)^{x+1}}{q_{y+1}} + \sum_{i=y+1}^{m} \frac{1}{q_i} - \frac{(1-k)^x}{q_y} - \sum_{i=y+1}^{m} \frac{1}{q_i} =$$
$$\frac{(1-k)^{x+1}}{q_{y+1}} - \frac{(1-k)^x}{q_{y+1}} < 0$$

3. $$\forall y \in [m], \quad g_1'(\lambda, y) = \frac{(1-k)^\lambda}{q_y} + \sum_{i=y+1}^{m} \frac{1}{q_i} > \sum_{i=y+1}^{m} \frac{1}{q_i}$$
$$= \frac{(1-k)^0}{q_{y+1}} + \sum_{i=y+2}^{m} \frac{1}{q_i} = g_2'(0,y+1) \quad \square$$

It follows from a combination of lemma 2, 4 and 5 that our potential function g is a level function.

4.2.2 Expected drift.

We denote the *drift* per timestep as $\Delta_{t+1} := g(X_t^{Y_t+1}, Y_t) - g(X_t^{Y_{t+1}+1}, Y_{t+1})$. The level of the population can either drop a level, remain the same or gain a level. We calculate the expected drift as the sum of the expected drift when we drop a level and the expected drift when we do not:

$$E_t[\Delta_{t+1}] = (1 - Pr(Y_{t+1} < Y_t)) \cdot E_t[\Delta_{t+1} \mid Y_{t+1} \geq Y_t] + Pr(Y_{t+1} < Y_t) \cdot E[\Delta_{t+1} \mid Y_{t+1} < Y_t]$$

4.2.3 Negative Drift.

Note that $Y_{t+1} < Y_t$ means that we have dropped a level, going in the wrong direction is called*negative drift*. We drop a level if and only if less than γ_0 of the population is at level Y_t or higher meaning: $X_{t+1}^{Y_t} < \gamma_0 \lambda$. We will try and estimate these bounds. Note that lemma 1 states that if we are at level Y_t, the expected average sampled individuals in level Y_t is lower bound by $(1+\delta)\gamma_0\lambda$. Noting that the number of individuals per generation are sampled from a binomial distribution, we can use multiplicative Chernoff bounds to calculate the odds of sampling less than a factor $\frac{\delta}{\delta+1}$ away from the average. If it is nearly impossible to drop such a small factor, it will be nearly impossible to drop a level.

$$Pr(Y_{t+1} < Y_t) = Pr(X_{t+1}^{Y_t} < \gamma_0\lambda) \leq Pr\left(X_{t+1}^{Y_t} < (1 - \frac{\delta}{\delta+1})(1+\delta)\gamma_0\lambda\right)$$
$$\leq EXP[\frac{-\delta^2(1+\delta)\gamma_0\lambda}{2(1+\delta)^2}] \leq EXP[\frac{-\delta^2\gamma_0\lambda}{4}]$$

This is where we need condition $(G3)$, substituting the lower bound for the population size into the equation grants:

$$EXP[\frac{-\delta^2\gamma_0\lambda}{4}] \leq EXP[2 \cdot \ln\left(\frac{128m}{\gamma_0\delta^3 z_*}\right)] = \left(\frac{\delta^3\gamma_0 z_*}{128m}\right)^2 < \left(\frac{\delta^{9/2}}{335}\right)\left(\frac{z_*}{46m}\right) \tag{2}$$

We also provide a bound using the bound $e^{-x} < \frac{1}{x}$ instead.

$$EXP[\frac{-\delta^2\gamma_0\lambda}{4}] = EXP[\frac{-\delta^2\gamma_0\lambda}{8}] \cdot EXP[\frac{-\delta^2\gamma_0\lambda}{8}] < \left(\frac{\delta^3\gamma_0 z_*}{128m}\right) \cdot \left(\frac{8}{\delta^2\gamma_0\lambda}\right) \leq \left(\frac{\delta}{16m\lambda}\right) \tag{3}$$

Both bounds give a very low probability of the event $Y_{t+1} < Y_t$ and thus we can choose a very pessimistic value for the expected value of the drift since it won't have a major effect on the overall expected drift. We now note that because g is a level function and the lowest level is 1, g is always bounded from above by $g(0,1)$. Moreover filling in these values and noting that $1-(1-z_j)^\lambda \geq 1 - e^{\lambda z_j} \geq \lambda z_j/(1+\lambda z_j)$ gives an upper bound for $g(0,1)$:

$$g(0,1) \leq m \ln\left(1 + \frac{\delta^{\frac{7}{2}}\lambda}{144\sqrt{2}}\right) + \sum_{j=1}^{m} \frac{1}{1-(1-z_j)^\lambda} \tag{4}$$
$$\leq m \ln\left(1 + \frac{\delta^{\frac{7}{2}}\lambda}{144\sqrt{2}}\right) + m + \sum_{j=1}^{m} \frac{1}{\lambda z_j} \tag{5}$$
$$\leq m\left(\ln\left(\frac{\delta^{\frac{7}{2}}\lambda}{144\sqrt{2}} + 1\right) + 1 + \frac{1}{\lambda z_j}\right) \tag{6}$$
$$\leq m\left(\frac{\delta^{\frac{7}{2}}\lambda}{144\sqrt{2}} + \frac{2}{z_*}\right) \tag{7}$$

We now use the bound on $g(0,1)$ and the bounds found in 2 and 3 to produce a lower bound on the negative drift:

$$Pr(Y_{t+1} < Y_t)E\left[\Delta_{t+1} \mid Y_{t+1} < Y_t\right] \geq -g(0,1)Pr(Y_{t+1} < Y_t) \tag{8}$$
$$\geq -\left(Pr(Y_{t+1} < Y_t)\left(\frac{2m}{z_*}\right) + Pr(Y_{t+1} < Y_t)\left(\frac{\delta^{\frac{7}{2}}\lambda m}{144\sqrt{2}}\right)\right) \tag{9}$$
$$\geq -\left(\left(\frac{\delta^{9/2}}{335}\right)\left(\frac{z_*}{46m}\right)\left(\frac{2m}{z_*}\right) + \left(\frac{\delta}{16m\lambda}\right)\left(\frac{\delta^{\frac{7}{2}}\lambda m}{144\sqrt{2}}\right)\right) \tag{10}$$
$$\geq -\left(\frac{2}{46}\right)\left(\frac{\delta^{\frac{9}{2}}}{355}\right) - \left(\frac{\delta^{\frac{9}{2}}}{16 \cdot 144\sqrt{2}}\right) \tag{11}$$

4.2.4 Positive drift.

We now look at the expected value gained when not dropping a level, so called *positive drift*. In this case $Y_{t+1} \geq Y_t$ and from lemma 3 it follows that for any $X \in [\lambda]$:

$$g(X_{t+1}^{Y_{t+1}+1}, Y_{t+1}) \leq g(X, Y_t) = g(X_t^{Y_t+1}, Y_t)$$

We can substitute this inequality into the expected drift:

$$E\left[\Delta_t \mid Y_{t+1} \geq Y_t\right] = E\left[g(X_t^{Y_t+1}, Y_t) - g(X_{t+1}^{Y_{t+1}+1}, Y_{t+1}) \mid Y_{t+1} \geq Y_t\right]$$
$$\geq E\left[g(X_t^{Y_t+1}, Y_t) - g(X_{t+1}^{Y_{t+1}+1}, Y_t) \mid Y_{t+1}\right]$$

We again distinguish two cases. In the first case the number of individuals in a lever higher than the current level is zero, $X_t^{Y_t+1} = 0$ and the case where that is not the case.

$X_t^{Y_t+1} = 0$: In this case we prove that g_1 has a drift greater than zero and provide a lower bound for the drift using g_2. The number of individuals in a level higher than Y_t can not drop below zero so we know that $X_t^{Y_t+1} \leq X_{t+1}^{Y_t+1}$. We also know that the first part of the potential function g_1 is a level function and so it follows from the first axiom that:

$$E\left[g_1(X_t^{Y_t+1}, Y_t) - g_1(X_{t+1}^{Y_t+1}, Y_t) \mid Y_{t+1} \geq Y_t\right] \geq 0$$

If we gain no individuals in the next level we have an expected drift of zero so we can derive an expected bound for g_2 by calculating the odds that the number of individuals in the next level or higher will be greater or equal than zero. We note that because we know that $X_{t+1}^{Y_t+1}$ is binomially distributed it follows that:

$$Pr(X_{t+1}^{Y_t+1} \geq 1) = 1 - (1 - p_{t+1}^{Y_t+1})^\lambda \geq 1 - (1 - z_{Y_t})^\lambda$$

Now using the fact that g_2 is also a level function we get:

$$E\left[g_2(X_t^{Y_t+1}, Y_t) - g_2(X_{t+1}^{Y_t+1}, Y_t) \mid Y_{t+1} \geq Y_t\right] \geq Pr(X_{t+1}^{Y_t+1} \geq 1)(g_2(0, Y_t) - g_2(1, Y_t)) \geq \frac{\delta^{\frac{9}{2}}}{355}$$

$X_{t+1}^{Y_t+1} > 0$: In this case we give a lower bound for g_1 using lemma 26 in [8]:

Lemma 6. (Lemma 26 in [8])
If for some $\delta \in (0, 1]$ and $i > 0$, $X \sim Bin(\lambda, p)$ with $p \geq (i/\lambda)(1 + \delta)$,
then:

$$E\left[\ln\left(\frac{1 + \frac{\delta^{\frac{9}{2}} X}{144\sqrt{(2)}}}{1 + \frac{\delta^{\frac{9}{2}\, i}}{144\sqrt{(2)}}}\right)\right] \geq \frac{\delta^{\frac{9}{2}}}{355}$$

Using this lemma gives:

$$E\left[g_1(X_t^{Y_t+1}, Y_t) - g_1(X_{t+1}^{Y_t+1}, Y_t)\right] = E\left[\ln\left(1 + \frac{\delta^{\frac{7}{2}}}{144\sqrt{2}} X_{t+1}^{Y_t}\right) - \ln\left(1 + \frac{\delta^{\frac{7}{2}}}{144\sqrt{2}} X_t^{Y_t}\right)\right]$$
$$= E\left[\ln\left(\frac{1 + \frac{\delta^{\frac{7}{2}}}{144\sqrt{2}} X_{t+1}^{Y_t}}{1 + \frac{\delta^{\frac{7}{2}}}{144\sqrt{2}} X_t^{Y_t}}\right)\right] \geq \frac{\delta^{\frac{9}{2}}}{355}$$

We now show that the drift of g_2 is greater or equal to zero. Note that $X_t^{Y_t+1}$ is known and constant.

$$E\left[g_2(X_t^{Y_t+1}, Y_t) - g_2(X_{t+1}^{Y_t+t}, Y_t)\right] = \frac{1}{1 - (1 - z_j)^\lambda} E\left[\left(1 - \frac{\delta^{\frac{9}{2}}\lambda}{355}\right)^{X_t^{Y_t+1}} - \left(1 - \frac{\delta^{\frac{9}{2}}\lambda}{355}\right)^{X_{t+1}^{Y_t+1}}\right]$$
$$= \frac{1}{1 - (1 - z_j)^\lambda}\left(\left(1 - \frac{\delta^{\frac{9}{2}}\lambda}{355}\right)^{X_t^{Y_t+1}} - E\left[\left(1 - \frac{\delta^{\frac{9}{2}}\lambda}{355}\right)^{X_{t+1}^{Y_t+1}}\right]\right) \geq 0$$

So for all cases we have a lower bound for the positive drift:

$$(1 - Pr(Y_{t+1} < Y_t))E[\Delta_{t+1} \mid Y_{t+1} \geq Y_t] \geq \left(1 - \frac{1}{16}\right)\left(\frac{\delta^{\frac{9}{2}}}{355}\right) \tag{12}$$

4.2.5 Using the lower bounds.

We can now compute a lower bound for the expected drift using the lower bounds for the negative drift and the positive drift found in (8) and (12).

$$\begin{aligned}
E[\Delta_{t+1}] &= Pr(Y_{t+1} < Y_t)E[\Delta_{t+1} \mid Y_{t+1} < Y_t] + (1 - Pr(Y_{t+1} < Y_t))E_t[\Delta_{t+1} \mid Y_{t+1} \geq Y_t] \\
&\geq \left(1 - \frac{1}{16}\right)\left(\frac{\delta^{\frac{9}{2}}}{355}\right) - \left(\frac{2}{46}\right)\left(\frac{\delta^{\frac{9}{2}}}{355}\right) - \left(\frac{\delta^{\frac{9}{2}}}{16 \cdot 144\sqrt{2}}\right) \\
&\geq \left(1 - \frac{1}{16}\frac{1}{23} - \frac{1}{9}\right)\left(\frac{\delta^{\frac{9}{2}}}{355}\right) > \frac{\delta^{\frac{9}{2}}}{454}
\end{aligned}$$

We know that the potential function is upper bound by $g(0,1)$ in (6). We also have a lower bound for the expected drift. This allows us to use the additive drift theorem to create an upper bound for the expected stopping generation. The level based analysis assumes that one makes λ function calls per generation and thus we multiply by λ to get the expected stopping time $E[T]$:

$$E[T] \leq \frac{\lambda g(0,1)}{\Delta_{min}} \leq \left(\frac{454}{\delta^{\frac{9}{2}}}\right)\left(m\lambda\left(\ln\left(1 + \frac{\delta^{\frac{7}{2}}\lambda}{144\sqrt{2}}\right) + 1\right) + \sum_{j=1}^{m}\frac{1}{z_j}\right) \quad \square$$

4.3 A recombination version of the Level Based Theorem.

[5] and [8] also present a version of the Level Based Theorem that applies to algorithms using recombination (the papers call it crossover). Technically GOMEA algorithms do not make use of mutation, only of recombination and thus we look at the possibility of using the recombination version of the theorem. This version uses five axioms: C1 to C5.

- C1 describes a lower bound s_j of the chance of mutating an individual from level j into a level higher than j.
- C2 describes a lower bound P_0 of the chance to mutate an individual in level j into a level that is not worse than j.
- C3 a lower bound ϵ on the chance of creating an individual in a level higher than j when crossing an individual in j with an individual in a level higher than j.
- C4 a lower bound on the chance of selecting the fittest part of the population.
- C5 follows from C1-C4 and describes a lower bound on the population size λ.

These axioms are used to produce a lower bound for the expected runtime $E[T]$ which is almost similar to the one provided by the level based theorem. The adjusted version of the Level Based Theorem is proved in both [5] and [8] by transforming $C1-C5$ into $G1-G3$. This version is however, not applicable. We will provide a small sketch of the proof to explain why:

- G1 is constructed in the following way: we want a lower bound z_j on the chance that we sample an individual in a level higher than the current level j. The proof states that this can happen in two ways: we either select two parents with one in a level higher than j, and get an individual in a level higher than j with chance ϵ (C3), if that happens the mutation operator must simply not ruin the new offspring with chance P_0 (C2). If we select two parents in level j, we get offspring in level j with chance ϵ (C3) and then mutate into a level higher with a chance lower bound by c_j (C1). These two chances can form a new version of $G1$.
- G2 is constructed by looking at the chance of not destroying a good solution during crossover presented in (C3) and at the selection pressure described in (C4).
- The bounds in C5 just imply the bounds in G3

The proof shows that the recombination version of the Level Based Theorem cannot accurately model the behavior of recombination algorithms. The proof only looks at the chance that the crossover does not ruin the solution: G1 is constructed by assuming that we already have a good individual and looking at the odds of keeping it, or by looking at the chance that the crossover does not ruin any individual and that mutation then makes progress. This analysis never looks at the chance that the crossover operator generates a better solution on its own. It is not surprising that this analysis provides similar bounds to the Level Based Theorem since it practically models the same behavior: progress through mutation and convergence to that progress. This approach will never be able to correctly model the behavior of crossover algorithms. Crossover algorithms are used on domains that contain local optima. Mutation operators have a hard time breaking through local optima and once the algorithm has reached a local optimum the chance that the mutation operator breaks through might even be zero and the only hope for progress in these cases is the crossover operator. If your domain contains such local optima, this version of the Level Based Theorem will provide an upper bound of infinite time because it neglects the chance that the crossover operator makes progress on its own. That makes this version unsuitable for use, therefore we choose to make adjustments to the original Level Based Theorem when analyzing GOMEA algorithms. The reason that this version cannot model the behavior of recombination algorithms accurately is because the Level Based Theorem itself has problems modeling the behavior of two separate operators, more on this will be in section 6.

4.4 Adjustments to the theorem.

Because we cannot make use of the recombination adjustments, this paper will introduce its own adjustments to the Level Based Theorem that improve the quality-of-life when applying the Level Based Theorem. The Level Based Theorem has three axioms G1, G2, G3. Two axioms have parameters which you can tweak and the third axiom G3 follows from the values of the other two. Calculating G1 and G2 and the following G3 for every single GOMEA algorithm on every domain is a large task but applying Level Analysis on only GOMEA algorithms allows us to make a few adjustments to the Level Based Theorem to improve the quality of life when applying it. Each of the axioms in the Level Based Analysis tries to model a certain aspect of an EA. G1 tries to model mutation and looks for the minimal chance of having a successful mutation z_*. G2 models the effects of the selection operator with a parameter δ and G3 just ensures that the population is large enough for the proof to work. GOMEA algorithms do not have a specific mutation/selection phase only accept improvements. Because the algorithm only makes use of one operator and not two or three, having two separate tweak variables z_* and δ seems redundant. This paper presents adjustments to the Level Based Theorem that reduce the amount of tweaking parameters. We will keep G1 intact and rename it axiom I and alter the second and third axiom using the properties of the GOMEA algorithms to create an adjusted version of theorem 3:

4.4.1 The second axiom.

The second axiom, G2, requires that if γ_0 of the population is at level j or higher and γ of the population is above level j, then we create an individual in a level higher than j with a chance lower bound by $(1+\delta)\gamma$. The elitism of the algorithm assures that if we pick an individual with a level higher than j, we are guaranteed to get an individual in a level higher than j. This will happen $\gamma\lambda$ times per generation. In all the other cases ($(1-\gamma)$ of the time) we have a lower bound chance of z_* to find an improvement. This leads to a chance of: $\gamma(1+\frac{1-\gamma}{\gamma})z_*$ to get an improvement. Now note that γ is upper bound by γ_0 and that gives us a lower bound for $G2$:

$$\forall j \in [m-1], \forall t \text{ if } |P_t \cap A^+_{j-1}| \geq \gamma_0\lambda \text{ and } |P_t \cap A^+_j| \geq \gamma\lambda \text{ then}$$
$$Pr(y \in A^+_j \mid y \in D(P_t)) \geq (1 + \frac{(1-\gamma_0)}{\gamma_0}z_*)\gamma \Rightarrow \delta = \frac{1-\gamma_0}{\gamma_0}z_*$$

4.4.2 The third axiom.

The third axiom, $G3$, requires that for $z_* := min\{z_j \mid j \in [m]\}$ the population size λ satisfies: $\lambda \geq \left(\frac{8}{\gamma_0 \delta^2}\right) \ln\left(\frac{128n}{\gamma_0 \delta^3 z_*}\right)$. The first intuition is to use the δ that we have just set for $G2$ in this axiom, but that leads to a problem: filling in our value for δ ensures that the population size λ must be greater or equal to: $\left(\frac{8}{c_1 \cdot z_*^2}\right) \ln\left(\frac{128n}{c_2 \cdot z_*^4}\right)$ for some constants c_1, c_2. This makes the population size almost always too large if the smallest chance for sampling a good individual is even moderately low. Assume that we have an elitist algorithm that has a chance of $1/n$ to gain a level each generation. If we have n levels, we assume to be done in roughly $\mathcal{O}(n^2)$ generations. But if $z_* = 1/n$, the population size would have to be of $\mathcal{O}(n^2 \ln(n))$. When you note that per generation GOMEA algorithms tend to do n function evaluations for every individual in the population, we would have order $n^3 \ln(n)$ function calls in one generation alone, making this population size not feasible. The elitism of the EA algorithms allows us to make a different adjustment to G3:
Intuitively, G3 ensures that the population must be large enough to ensure that the odds of dropping a level are near zero. But the elitism of the algorithm already ensures that we will never drop a level. The GOMEA algorithms do require a certain minimal population size. The GOMEA algorithms have no actual form of mutation so if there exists an index i for which all the individuals of the population have no 1, the algorithm can never terminate. The population size λ must be large enough to ensure that for every string index i, the odds of having a zero at i for every individual in the starting population is near zero. Usually when a genetic algorithm takes too long we terminate and start over. We do not want this to happen too often but in practice we'll gladly accept a chance of 1 in n^k per index to have to start over. Assume we initialize the population with a random chance $\frac{1}{c}$ to have a 1 at any index. We now look at the odds that every individual in the population receives a zero bit:

$$Pr(\textit{all zero bits}) = (1 - \frac{1}{c})^\lambda = \frac{1}{n^k} \Rightarrow \lambda = \frac{k \ln(1/n)}{\ln(1 - \frac{1}{c})}$$

From this it follows that if we pick λ to be greater or equal to $\mathcal{O}(\ln(n))$ the odds of never finding an improvement and having to restart are small enough.

4.4.3 The Adjusted Level Based Analysis.

The last adjustment we must make is that the GOMEA algorithm does not do λ function calls per generation. Per individual, GOMEA algorithms traverse the entire FOS structure $\mathcal{F}$ and can do a function evaluation per item in the FOS structure. Therefore we mustn't multiply the expected number of generations by λ but rather by $\lambda \cdot |\mathcal{F}|$. This all leads to an adjusted version of the Level Based Theorem that we'll use for the analysis of GOMEA algorithms:

Theorem 4. Adjusted Level Based Analysis.
Given a problem with problem size n, a partition $\{A_0, A_1, ..., A_m\}$ of the search space Ω an algorithm with population size λ and a population per generation P_t, define the stopping time $T := \{t\lambda \mid |P_t \cap A_m| > 0\}$ to be the first time (with time in the number of function calls) that an optimal element in appears in P_t. If there exist $z_0, ...z_m \in (0, 1]$ and $\gamma_0 \in (0, 1)$ and such that for any population P:

I) $\forall j \in [m], \forall t$ *if* $|P_t \cap A_{j-1}^+| \geq \gamma_0 \lambda$ *then* $Pr(y \in A_j^+ \mid y \in D(P_t)) \geq z_j$

II) *The population size λ satisfies:* $\lambda \geq \mathcal{O}(\ln(n))$

$$\textit{Then with high probability } E[T] \leq |\mathcal{F}| \left(\frac{454}{\frac{9}{z_*^2}}\right) \left(m\lambda(1 + \ln(1 + \frac{z_*^{7/2}\lambda}{203})) + \sum_{j \in [m]} \frac{1}{z_j}\right)$$

4.5 Applying the Level Based Theorem on Onemax.

We will now calculate an expected upper bound for the runtime of the Univariate GOMEA algorithm on the Onemax domain. In Onemax we have a fitness function f that counts the number of one bits in a string of length n. The optimum x^* is found when the string consists entirely out of ones making $f(x^*) = n$. The population is randomly generated with a chance $\frac{1}{c}$ to have a one bit at a given index in a given individual. In most cases $c = 2$ and benchmark algorithms tend to solve this Onemax problem in $\mathcal{O}(n \ln(n))$ steps. We will try and create bounds that match this benchmark. Because the algorithm randomly selects a donor per index, the odds of making an improvement for a given parent and for a given bit index where the parent has a zero, is at first equal to the beginning distribution: $\frac{1}{c}$. During the process the number of one bits in the population will increase but those new one bits will not follow uniform random distribution. This is because if we make an improvement and increase the number of ones at a certain index i, the odds that the next improvement is at that index is greater because the algorithm deterministically iterates over all the indexes of all the members of the population. This is a problem, since the level based theorem asks for a random distribution to sample the new population from. We solve this by only looking at the original distribution for opportunities for improvement. If we sample an individual and we want to improve a certain index with a zero, the odds of selecting an individual with a one bit at that location is given by only looking at the original (unchanging) distribution hence always giving the chance $\frac{1}{c}$ for improvement for that index of that individual.

4.5.1 Normal level based theorem.

We define the fitness level $A_j := \{x \mid f(x) = j\}$ giving us $m = n$ fitness levels. We now look at the odds of making an improvement. Assuming the population is at level j the odds of making an improvement z_j are lower bound by one minus $n - j$ failures:

$$z_j = 1 - (1 - 1/c)^{(n-j)} \Rightarrow z_* = \frac{1}{c} \tag{13}$$

Filling in $z_* = 1/c$ and $\lambda = \mathcal{O}\ln(n)$ for I and II we get: $E[T] \leq \mathcal{O}\left(|\mathcal{F}| \cdot c^{9/2} \cdot n \ln(n) \ln(c^{7/2} \ln(n))\right)$ iterations. We note that the only FOS structures in $\mathcal{F}$ are the n string indexes so we get $|\mathcal{F}| = n$, if c is a constant and not dependent on n we get: $E[T] \leq \mathcal{O}\left(n^2 \log(n) \log(\log(n))\right)$. These bounds are higher than the set benchmark. These bounds however, are a direct result of the current approach: we always do n function calls per iteration and have a population of $\ln(n)$ individuals, if we also have n levels then the expected upper bound will be at least $n^2 \ln(n)$ and that is too high. If we want any improvement, we need to redefine the levels.

4.5.2 Redefining levels.

For simplicity in writing we denote $b(x)$ to be the function indicating the amount of 'bad' indexes in a candidate solution as $b(x) = n - f(x)$. Now for a number $r \geq 1$ we define the fitness levels A as $A_j := \{x \mid n \cdot (\frac{1}{r})^j \leq b(x) < n \cdot (\frac{1}{r})^{j+1}\}$. Note that this definition implies that the maximal amount of levels m is reached when $(\frac{1}{r})^m = \frac{1}{n}$ implying that $m = \frac{ln(1/n)}{ln(1/r)} = \frac{ln(n)}{ln(r)}$. We want to keep r as high as possible to reduce the amount of levels that we create.

Assume we have an individual at level j. Worst case we have the maximal amount of 'bad genes': $b(x) = n(\frac{1}{r})^j$. To create an individual in the next level we would have to make at least $n(\frac{1}{r})^j - n(\frac{1}{r})^{j+1} = (1 - \frac{1}{r}) \cdot n(\frac{1}{r})^j$ improvements. Here we see that the larger r is, the greater the leap will be before we cross a level. If the leap is too large then the worst case chance to make the leap z_* becomes too small. We also noted that r must be as large as possible in order to reduce the number of levels m. We demand that r is small enough to ensure that $(1 - \frac{1}{r}) < \frac{1}{c}$ and then choose r to be large enough within that frame. If we look to determine the $z_j s$ and z_* to satisfy I, we must look the odds of gaining a level. We can make an improvement when we sample a parent which has a one at that index. Just as in the normal case, this chance is lower bound by $\frac{1}{c}$. Noting that we can make $n(\frac{1}{r})^j$ improvements this gives us a binomial distribution $Bin(n(\frac{1}{r})^j, \frac{1}{c})$. We can use this binomial distribution to calculate an upper bound for the chance of not gaining a level using once again Multiplicative Chernoff bounds. Note that the expected value is $\mu = p \cdot n(\frac{1}{r})^j$.

$$P(X \leq (1-\delta)\mu) \leq e^{-\frac{\delta^2\mu}{2}} \tag{14}$$

μ is the expected value being $p \cdot n(\frac{1}{r})^j$ and filling the rest in gives: $(1-\frac{1}{r}) \cdot n(\frac{1}{r})^j = (1-\delta)p \cdot n(\frac{1}{r})^j \Rightarrow \delta = 1 + \frac{1}{rp} - \frac{1}{p}$. Substituting these values in the equation gives:

$$Pr(X \leq (1-\frac{1}{r}) \cdot n(\frac{1}{r})^j) \leq EXP[-\frac{(1+\frac{1}{rp}-\frac{1}{p})^2(\frac{1}{r})^j \cdot pn}{2}] \tag{15}$$

Note that this is smallest when j is largest, filling in the maximal value for j, $m = \ln(\frac{1}{n})/\ln(\frac{1}{r})$ grants us:

$$Pr(X \leq (1-\frac{1}{r}) \cdot n(\frac{1}{r})^j) \leq EXP[-\frac{(1+\frac{1}{rp}-\frac{1}{p})^2 p}{2}] \tag{16}$$

We can now fill in the axioms of the adjusted theorem:

- I) $z_* = 1 - EXP[-\frac{(1+\frac{1}{rp}-\frac{1}{p})^2 p}{2}]$
- II) $\lambda = \mathcal{O}(\ln(n))$.

And $m = ln(n)/ln(r)$. Filling in any fitting value for r will result in an upper bound. The benchmark form of Onemax has $c = 2$. In this case r is some small constant and $m = \mathcal{O}(\ln(n))$ and the expected running time $E[T]$ will be upper bound by $\mathcal{O}(n\ln(n) \cdot \ln(n)\ln(\ln(n)))$ which is still higher than the benchmark time of $\mathcal{O}(n\ln(n))$. Even if $\frac{1}{c}$ would be very close to 1 and we would be able to choose $r = \mathcal{O}(\ln(n))$ we would only get $m = \frac{\ln(n)}{\ln(\ln(n))}$ and $E[T] \leq \mathcal{O}\left(n\frac{\ln(n)}{\ln(\ln(n))}\ln(n)\ln(\ln(n))\right) = \mathcal{O}\left(n\ln^2(n)\right)$ and still be a factor $\ln(n)$ off. This also seems to be the maximal potential for the Level Based Theorem: we have a population size λ of $\mathcal{O}(\ln(n))$ and worst case do order n function calls per iteration. The only way that the level based theorem could achieve an upper bound of $\mathcal{O}(n\ln(n))$ is if we were to have a constant amount of levels and would get there in a constant number of generations but we cannot model that using the Level Based Theorem without z_* getting too high.

5 Other methods to determine upper bounds.

We have applied the Level Based Theorem to the Univariate GOMEA and the best upper bound we have gotten was $\mathcal{O}(n\ln(n)\lambda) = \mathcal{O}\left(n\ln(n)^2\right)$, if we initialize the population with a chance near one to get a 1 bit at a given index. We have noted before that we suspect the Univariate GOMEA to have a runtime of $\mathcal{O}(n\lambda)$ and that we thus suspect the Level Based Theorem to give too high bounds. This section will use another general theory called Martingale theory to show that this is the case and will also provide a direct proof.

5.1 Martingale theory.

One of the things that makes the Level Based Theorem valuable is that it presents a general framework to determine an upper bound for any algorithm. The theorem almost only looks at the odds of improvement and that makes it applicable to any EA. One could make the argument that having only a little too high bounds is OK if your framework provides generality. Therefore we determine the upper bound of the Univariate GOMEA algorithm using another general theory, Martingale theory. Martingale theory is a model similar to Markov chains that predicts future winnings based on past events. We will use Martingale theory to model the progression that the Univariate GOMEA makes per iteration and will then use a general inequality, Azuma's inequality, to get an upper bound for the expected runtime.

5.1.1 Modeling Univariate GOMEA.

The Level Based Theorem ran into the problem that it had a population size λ of $\mathcal{O}(\ln(n))$ and that we had to assume that we do $\mathcal{O}(n)$ function evaluation calls per iteration. That means that if we want to have tighter bounds we need to have a near constant number of generations and that could not be achieved within the framework. Therefore we try a different approach and we model the progression of the Univariate GOMEA per iteration and not generation.

We note that in the Univariate model, all the indexes (or problem variables) improve independently from the others. Therefore we look at only one bit position and for the argument to work we assume that we only evaluate if the index still has a zero bit. In the end of the section we will give a justification for this assumption and note that this assumption leads to a shorter and better proof. We provide an upper bound on the runtime by showing that for any index (or problem variable) it takes order λ function evaluations to have the entire population have a 1-bit at that index. If that happens n times the entire population has converged to the optimal solution and thus the algorithm terminates in $\mathcal{O}(n\lambda)$ function evaluations.

5.1.2 The proof.

Assume we have population size λ and choose an arbitrary index. Denote X_t to be the number of individuals in the population that has a one at that index at a given iteration t. We only iterate over zero bits, so per iteration we can either improve X_t by one by selecting a donor which already has a one at that position, or remain the same. It follows that:

$$E[X_{t+1} \mid X_t] = (1 + \frac{1}{\lambda})X_t \tag{17}$$

We now define $Y_t := (1 + \frac{1}{\lambda})^{-k} X_t$ it follows that $E[Y_t \mid Y_0] = Y_0$ and thus we have defined a martingale. We now denote T as the first time that the entire population only consists out of one bits, $T := min\{t \mid X_t = \lambda\}$. Note that T is again a stopping time. We prove that the odds of T being of greater order than λ are near zero. We look at the odds of T being greater than an arbitrary N and then apply Azuma's inequality to give an upper bound near zero when N approaches λ.

Theorem 5. Azuma's inequality
Suppose Y_k is a martingale and for all $k \leq N$, $|Y_k - Y_{k-1}| < c_k$.
Then almost surely for any N:

$$Pr(Y_N - Y_0 \geq 0) \leq EXP\left[\frac{-t^2}{2\sum_{k=1}^{N} c_k}\right]$$

We first look at the maximal difference between Y_{k+1} and Y_k:

$$\begin{aligned}
|Y_{k+1} - Y_k| &= |(1 + \frac{1}{\lambda})^{-k-1} X_{k+1} - (1 + \frac{1}{\lambda})^{-k} X_k| \\
&= |((1 + \frac{1}{\lambda})^{-1} - 1)(1 + \frac{1}{\lambda})^{-k} X_k + (1 + \frac{1}{\lambda})\{0, 1\}| \\
&\leq |((1 + \frac{1}{\lambda})^{-1} - 1)(1 + \frac{1}{\lambda})^{-k} X_k| \\
&\leq \frac{1}{(1+\lambda)}(1 + \frac{1}{\lambda})^{-k}\lambda \\
&\leq (1 + \frac{1}{\lambda})^{-k} = c_k
\end{aligned}$$

We now look at the chance of T being greater than N. If T is greater than N it means that after N steps the population still has not fully converged and that X_N is thus smaller than λ.

$$\begin{aligned}
Pr(T > N) &= Pr(X_N \leq \lambda) = Pr(Y_N(1 + \frac{1}{\lambda})^N < \lambda) \\
&= Pr(Y_N \leq \lambda(1 + \frac{1}{\lambda})^{-N}) = Pr(Y_N - Y_0 \leq -(Y_0 - \lambda(1 + \frac{1}{\lambda})^{-N})
\end{aligned}$$

These two equations allow us to apply Azuma's inequality with $t = -(Y_0 - \lambda(1+\frac{1}{\lambda})^{-N})$ and $c_k = (1+\frac{1}{\lambda})^{-k}$. We also note that $\sum c_k$ is a geometric series.

$$\begin{aligned} Pr(T > N) =& \leq exp\left(\frac{-t^2}{2 \cdot \sum_{k=1}^{N} c_k^2}\right) \\ &= exp\left(\frac{-(Y_0 - \lambda(1+\frac{1}{\lambda})^{-N})^2}{2 \cdot \sum_{k=1}^{N}(1+\frac{1}{\lambda})^{-k}}\right) \\ &= exp\left(\frac{-(Y_0 - \lambda(1+\frac{1}{\lambda})^{-N})^2}{2 \cdot \frac{1-(1+\frac{1}{\lambda})^{-N}}{\frac{1}{\lambda}}}\right) \\ &= \exp\left(\frac{-(Y_0 - \lambda(1+\frac{1}{\lambda})^{-N})^2}{2\lambda(1-(1+\frac{1}{\lambda})^{-N})}\right) \end{aligned}$$

If we recall the previous analysis made on the runtime we know that we must be done in $N = \mathcal{O}(\lambda)$ timesteps. Noting $Y_0 = X_0 = \lambda/c$ for c constant so it follows that:

$$Pr(T > N) \leq \mathcal{O}\left(exp\left(\frac{-((\frac{1}{c} - (1+\frac{1}{\lambda})^{-\lambda})\lambda)^2}{2\lambda(1-(1+\frac{1}{\lambda})^{-\lambda})}\right)\right) \tag{18}$$

Now noting that $(1+\frac{1}{\lambda})^{-\lambda}$ will go to zero for λ large enough we get: $Pr(T > N) \leq \mathcal{O}\left(exp\left(-\lambda\right)\right)$. If λ is large enough this makes the chance of not terminating after order λ evaluations near zero and thus $E[T] \leq \mathcal{O}\left(n\lambda\right) = \mathcal{O}\left(n\ln(n)\right)$ $\square$.

Justifying only evaluating zeroes. This analysis worked because we assumed that we only did a function evaluation when we encountered a parent with a zero bit. The reason why we can assume this, is the following: if for any index i we select a donor with the same value at i, we do not need to make a change and thus do not need to evaluate. So we evaluate in two cases: When we have a 1 bit and the donor has a zero bit, or the other way around. On average throughout the runtime of the algorithm, these chances are equally great and so if we only look at evaluating one bits, we can take those number of evaluations and multiply them by two. This realization leads to the following new proof for a runtime of $\mathcal{O}\left(n\ln(n)\right)$ function calls: Note that we only evaluate when either the parent has a zero bit and the donor a one bit, or the other way around. On average, the odds of either happening are the same. So on average, we have a chance of $1/2$ per function call to make an improvement. Noting that we have order n bit positions per order $\ln(n)$ individuals leads to having an expected runtime of $E[T] = \mathcal{O}\left(n\ln(n)\right)$ function evaluations.

6 Discussion on the Level Based Analysis

This paper has applied the Level Based Theorem and has compared the received bounds with bounds derived by other methods. Doing this gave insight in the current problems with the Level Based Theorem. This section will discuss the problems of the theorem in regard to accuracy, both accuracy in general and the difficulty of obtaining accuracy on z_*, the problems that come with the definition of levels and the problems that arise when using the general level based theorem on specific tailor-made algorithms.

6.0.1 The accuracy of the Level Based Theorem.

The theorem has two main problems regarding accuracy. First we show that the theorem gives too high bounds in general. This forces one to have very tight bounds on all the variables of the theorem and that leads to the second problem, the difficulty of finding tight bounds on z_*.

When applying the Level based theorem, assuming that we initialize a population with a chance near 1 to get a 1 bit at a given index, we still got bounds that were an order $\ln(n)$ off. One of the reasons for this is that the current version theorem has too large constants. We already briefly mentioned that in section 4.4 when adjusting the third axiom. The analysis is based on four main parameters: the number of levels m, the lower bound chance of making an improvement z_*, a selection operator parameter δ and a tweaking parameter γ_0. If those four are set then according to the level based theorem we know a lower bound for the population size λ and an upper bound for the expected runtime $E[T]$:

$$E[T] \leq \left(\frac{454}{\delta^{\frac{9}{2}}}\right)\left(m\lambda\left(1+\ln(1+\frac{\delta^{7/2}\lambda}{203})\right)+\sum_{j\in[m]}\frac{1}{z_j}\right) \tag{19}$$

Whilst making adjustments in section 4.4 we noted that it was reasonable to denote that $\delta = z_*$. Now assume that we have a combinatorial optimization problem with size n and an elitist algorithm that always has a chance $\frac{1}{n}$ to gain a level. It is clear to see that this would give an expected runtime of order $n^2\lambda$. We note that in the case of an elitist algorithm, we are not bound by $G3$ and we will thus leave our population size to be λ and unspecified. Filling in the Level Based Analysis with $z_* = \delta = \frac{1}{n}$ and $m = n$ gives:

$$E[T] \leq \mathcal{O}\left(n^{\frac{11}{2}}\lambda\ln(\lambda)\right) \tag{20}$$

The upper bound is more than a factor n^3 off the actual expected upper bound. There is currently a version of the Level Based Theorem being developed that has a tighter upper bound for the expected runtime [13]. This version states:

$$E[T] \leq \left(\frac{8}{\delta^2}\right)\left(m\lambda\left(1+\ln(1+\frac{\delta\lambda}{2})\right)+\sum_{j\in[n]}\frac{1}{z_j}\right) \tag{21}$$

Filling in $z_* = \delta = \frac{1}{n}$ and $m = n$ in this version yields a better result:

$$E[T] \leq \mathcal{O}\left(n^3\lambda\ln(\lambda)\right) \tag{22}$$

But is still more than an order n off.

6.0.2 The difficulty of determining z_*.

z_* is a lower bound on the chance of sampling an improvement for any level. Determining a correct z_* is a difficult process of balancing the number of levels m with z_*. This is because the Level Based Theorem looks at only the chance of breaking out of a level, so the algorithm always has to have at least order m generations. If we know that our algorithm can go faster than that, we need to redefine the levels to reduce m (just as we did in section 4.5.2). But redefining the levels has severe consequences for z_* and that can make getting tight bounds on z_* even more difficult. We now know that the theorem has a tendency to give slightly too high bounds in general. This means that because the general theorem is a factor off, any inaccuracy on z_* will be multiplied by that factor. The fact that determining tight bounds for z_* is inherently a difficult balance between z_* and m, that the user often has to deal with difficult algebraic methods to get those bounds (like multiplicative Chernoff bounds) and that these bounds have to be very tight, make it that it is often easier to determine the runtime of the algorithm with a direct proof.

6.0.3 Problems that arise with levels.

There are two main problems that arise when using 'fitness levels' in the way the Level Based Theorem does. The first one is that you have to choose between modeling leaps or steady progress and the second one is that you often inaccurately model the domain.

As we mentioned earlier, the theorem assumes that you traverse all levels. Specifically it provides no tools to model any leaps in fitness value that your algorithm might make, except for redefining the levels such that the entire leap is one level. When you redefine a level to be larger than one fitness value, you can only look at the chance to traverse that large level in one go, so any steady progress that your algorithm might make in that level has to be neglected. This becomes a larger problem when dealing with algorithms that have multiple operators that can make improvements. Take a canonical crossover algorithm as an example: the recombination operator tends to make large jumps in fitness values whereas the mutation operator tends to make steady progress. When you define a level, you must choose whether the level is large, and thus only the recombination operator has a chance to break through or whether the level is small and you fully rely on the steady progress of the mutation operator. In this way, you lose some of the effectiveness of the algorithm when modeling it. The problem becomes even greater when you have more than two operators that can make an improvement. The Linkage Tree GOMEA recombines based on a binary tree of problem variables. Each level of depth of the tree is able to make a leap in fitness of a different size, yet the theorem will only allow one depth level of the entire tree to be fully effective per fitness level.

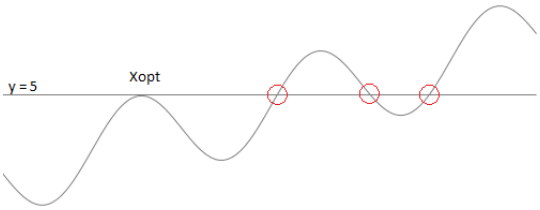

Figure 1: an illustration of how we collapse other points to local optima.

The second problem with the levels is that they are defined on fitness value, not on what the solutions with those fitness values look like. This becomes a problem when dealing with local optima. Assume that we have only one mutation operator, and a continuous domain with a local optimum x_{opt} at fitness value 5 and several other local optima. It is hard for a mutation operator to break out of a local optimum so when on the local optimum, it will be hard for the algorithm to go from fitness 5 to 6. The domain might contain many more points that have fitness value 5 and that pose no difficulty for the algorithm to make progress. Because the levels are based on fitness value alone, all those points are collapsed together with x_{opt} in one fitness level. The Level Based Theorem looks at the lower bound chance to get an improvement out of a single level, so we must take the very low chance to get out of x_{opt}. Effectively, we assume that every point with fitness value 5 behaves like x_{opt} making the domain harder than it actually is. This way of modeling levels and using those levels leads to multiplying the number of local optima in your model, possibly resulting in higher bounds on the runtime.

6.1 Future Work.

Section 2.3 introduces three forms of GOMEA algorithms: Univariate, Marginal Product and Linkage Tree. This paper provides upper bounds for only the Univariate GOMEA algorithm on the Onemax domain but it would be interesting to know accurate upper bounds for other forms or other domains. In this section we provide loose upper bounds for several domains and GOMEA algorithms as a benchmark for those who are interested in finding upper bounds for these algorithms using the Level Based Theorem. We mentioned that the Level Based Theorem does not provide the most suitable framework for these algorithms and it will be interesting to see what bounds other methods can provide. Note that we do not give rigorous proofs but rather proof sketches that give a loose upper bound.

6.1.1 Leading Ones with Univariate.

Leading one rates the fitness of a population as the number of consecutive ones at the beginning. $f_{LO}(x) := \sum_{i=0}^{n} \prod_{j=0}^{i} x[j]$ with the starting population initialized with a chance $\frac{1}{c}$ to have a 1-bit at a given index. If we define $A_j := \{x \in \Omega \mid f_{LO}(x) = j\}$ we get $E[T] \leq \mathcal{O}\left(n^2 \ln(n)\right)$.

Proof: Note that if you take an individual at level j then this individual has a zero bit at index $j+1$. The odds of making an improvement are given by the odds of selecting a donor which has a 1 bit at position $j+1$ and those odds are lower bound by the start distribution $\frac{1}{c}$. So $z_* = \delta = \frac{1}{c}$. Noting that $\frac{1}{c}$ is constant, it follows that $E[T] \leq n^2 \ln(\ln(n)) \ln(\ln(\ln(n)))$

6.1.2 K-L-plateau with Onemax.

Definition 16. *We define a k-L-plateau function to be fitness function f based on a fitness function f_{base} to be:* $f(x) = case(f_{base}(x))$

- $f_{base}(x) < L \rightarrow f_{base}(x)$
- $L \leq f_{base}(x) \leq L + k \rightarrow L$
- $f_{base}(x) > L + K \rightarrow f_{base}(x)$

This function gives the fitness values of its base function but starting from a certain level L it will give the value L for k improvements, this will create a flat plateau of fitness values. The problem with k-L-plateau fitness functions is that the algorithm is practically blind for improvements. It cannot see if it is going in the right or wrong direction which has the risk of iterating left and right without ever breaking out of the plateau.
Note that the elitism of our algorithm guarantees that once we have reached level L, we will not fall down. The algorithm will continue as usual after reaching fitness $L + k$. This is why we estimate the runtime as an upper bound for the runtime of the normal problem plus the expected time to breach the plateau.

6.2 Onemax.

Univariate GOMEA. Assume we have a K-jump on a Onemax domain. Note that up to the jump, the algorithm is simply doing Onemax with a known time of $\mathcal{O}(n\lambda \ln(\lambda n))$. The difficulty lies in the jump. Assume that the leap starts at level J, the elitism of the algorithm ensures that your fitness cannot go below level J. Since we are blind during the jump, we must assume the worst case and calculate the possibility of making the leap in one go. The worst case if the jump level $J = n - k$. this is fairly easy to see: if it is i less, you can try i more times since you never drop below J. We know make a general assumption on how the algorithm progresses: If the algorithm makes an improvement at an index i, the chance that an other individual also gets an improvement at i is increased. This makes it so that at level j, the number of 1-bits are not uniformly distributed over an individual, we actually see that the 1-bits have all gathered in a 'block' of 1-bits and that there are a random number of k' indexes that are still 'free variables' that have roughly the initial distribution. If we encounter a bit position that is contained in the 'block', the odds of matching with a donor that has a 1-bit in that index as well is nearly one because all the other individuals will have the same converged 'block' of indexes. This behavior ensures

that we do not have to look at all the indexes in the block so to break out of the leap we only have to encounter k' donors that all have a 1-bit at the index of these free variables. That happens with chance lower bound by the initial distribution $\frac{1}{c}$ so it follows that the runtime is $\mathcal{O}(c^{k'})$. If we've almost converged to the global optimum, k' is very close to the size of the leap k.

LTGA. We now provide an upper bound for the Linkage Tree Genetic Algorithm introduced in [12] and described in section 2.3. This algorithm has a binary tree of indexes as a FOS structure. The leaves of the tree work the same as the Univariate GOMEA does, noting that GOMEA is elitist we can use the upper bound of Univariate GOMEA for the canonical part of the K-Plateau. We only determine the bounds on actually making the leap. We again take the same assumption that all the gained 1-bits are not randomly distributed but rather gained in a 'block' of indexes. If the leap starts at level j, we will have about k free variables left. Again we note that the problem is the hardest when the leap is at the end, specifically when the leap size $K = k/2$. We first rely on the algorithm to recognize the non-free variables as a block meaning that there is a subset that contains the largest part of the block of indexes. Because the complement if a subset in $\mathcal{F}$ is also in $\mathcal{F}$ we have a subset $F_0 \in \mathcal{F}$ that contains all k free variables. Because the tree is roughly binary these subsets we have two subsets $F_1, F_2 \in \mathcal{F}$ that divide F_0 roughly into two.
To break through the plateau we need k improvements. So we look at the odds of selecting a parent with k zero's in F_1 and matching it with a donor that has all ones in that subset.

$$P(\textit{parent has k zeroes}) \geq \frac{(k!)^2}{(2k)!} \approx \mathcal{O}\left(\frac{\left(\sqrt{k}k^k\right)^2}{\sqrt{k}(2k)^{2k}}\right) = \mathcal{O}\left(\frac{\sqrt{k}k^{2k}}{4^k k^{2k}}\right) = \mathcal{O}\left(\frac{\sqrt{k}}{4^k}\right) \tag{23}$$

The parent has to have all zeros and the donor has to have all ones making the odds square giving a runtime of $\mathcal{O}(\frac{8^k}{k})$.

7 Conclusion.

This paper presents an application of the Level Based Theorem to the Univariate Gene-pool Optimal Mixing Evolutionary Algorithm. It elaborates on earlier publications of Drift Analysis and their proofs and discusses the recombination version of the Level Based Analysis. It then presents adjustments to the Level Based Theorem for when applying it to GOMEA algorithms and it provides a theoretical upper bound of order $n\ln(n)^2$ on the runtime of the Univariate GOMEA algorithm on the Onemax domain. Tighter bounds are found using Martingale Theory and a direct proof. The use of the Level Based Theorem is compared to these two methods. This comparison is used to discuss the current problems of the Level Based Theorem regarding its current accuracy and the difficulty of use. This paper then elaborates on how the definition of levels will always inaccurately model the problem domain and on how the general approach based on traditional EAs can lead to problems when analyzing tailor-made algorithms.

Acknowledgments. A special thanks to D-C Dang and P.K. Lehre for all the time they have devoted to me and this paper during my time in England. Their input was both helpful and inspirational and I thank them for all the time invested.

References

[1] A.E. Eiben, G. Rudolph, *Theory of evolutionary algorithms: A bird eye view*, Theoretical Computer Science 229, 1999, (3-9)

[2] Jun He, Xin Yao, *Drift analysis and average time complexity of Evolutionary Algorithms*, Artificial Intelligence 127, 2001

[3] B Doerr, D Johannsen, C Winzen. *Multiplicative Drift Analisys*, Algoritmetica, Volume 64, Issue 4, Springer, 2012, 673-697

[4] P.K.Lehre, *Fitness-Levels for Non-Elitist Populations*, Proceedings of the 2011 Annual Conference on Genetic and Evolutionary Computation, GECCO'11, 2011, (2075-2082)

[5] P.K.Lehre, D-C Dang, *Refined Upper Bounds on the Expected Runtime of Non-elitist Populations from Fitness-Levels*, Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, GECCO'14, 2014, (1367-1374)

[6] P.K.Lehre, D-C Dang, *Simplified Runtime Analysis of Estimation of Distribution Algorithms*, Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, GECCO'15, 2015, (513-518)

[7] Hajek, B. *Hitting-time and occupation-time bounds implied by drift analysis with applications*, Advances in Applied probability, 1982, (502-525)

[8] D. Corus, D-C Dang, A. V. Eremeev, P. K. Lehre, *Level-Based Analysis of Genetic Algorithms and Other Search Processes*, Parallel Problem Solving from Nature - PPSN XIII. Volume 8672, Springer 2014, (912-921)

[9] K. Sigman, *Stopping Times*, Lecture notes, University of Colombia, 2009

[10] Jun He, Xin Yao, *A study of drift analysis for estimating computation time of evolutionary algorithms*, Natural Computing 3, 2004, (21-35)

[11] D Thierens, P Bosman, *Optimal mixing evolutionary algorithms*, Proceedings of the 2011 Annual Conference on Genetic and Evolutionary Computation, GECCO'11, 2011, (617-624)

[12] D Thierens, *The Linkage Tree Genetic Algorithm*, Parallel Problem Solving from Nature - PPSN XI. 2010, (264-273)

[13] P.K. Lehre, personal communication April 2016.