2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

Runtime Analysis Comparison of Two Fitness Functions on a Memetic Algorithm for the Clique Problem

Kuai Wei and Michael J. Dinneen
Department of Computer Science, University of Auckland, Auckland, New Zealand
{kuai,mjd}@cs.auckland.ac.nz

***Abstract*—It is commonly accepted that a proper fitness function can guide the algorithm to find a global optimum solution faster. This paper will use the runtime analysis to provide the theoretical evidence that a small change of the fitness function (additional one step looking forward) can result in a huge performance gap in terms of finding a global optimum solution. It also shows that the fitness function that gives the best results in an Memetic Algorithm on the Clique Problem is entirely instance specific. In detail, we will formalize a (1+1) Restart Memetic Algorithm with a Best-Improvement Local Search, and run them on two different fitness functions, f_{OL} and f_{OPL}, to solve the Clique Problem respectively. We then construct two families of graphs, G_1 and G_2, and show that, for the first family of graphs G_1, the (1+1) RMA on the fitness function f_{OPL} drastically outperforms the (1+1) RMA on the fitness function f_{OL}, and vice versa for the second family of graphs G_2.**

I. Introduction

Memetic Algorithms (MAs) are a wide class of randomized search heuristics that hybridize Evolutionary Algorithms (EAs) with local searches. There are many examples in which MAs have successfully been used to solve various kinds of problems [13]. This motivates the desire for a better understanding of MAs by using runtime analysis.

MAs and EAs both apply mutations and recombinations, and the runtime analysis of these operations have been studied a lot in EAs. Some of the examples are: the study on dynamic mutation approach [6], [5], [11], [28]; the recombination (also known as crossover) operation [10], [12], [17]; and the analysis of population-based EAs, [9], [30]. Therefore, theoretical analyses of MAs have mainly focused on the impact of the local searches. The first runtime analysis on MAs is the (1+1) MA in 2006 [22] by Sudholt. Later on, in 2008, he [23] analyzed the (1+1) MA with variable-depth search to overcome local optimum on three binary combinatorial problems: Mincut, Knapsack, and Maxsat. Similarly, in 2009, the same author [24] showed that changing the depth of local searches or the frequency of applying local searches in MAs will reduce the performance from polynomial to super-polynomial. Furthermore, the interaction of mutations and local searches has attracted much attention. For example, Sudholt and Zarges [25] analyzed the interaction of two different mutations with local search for Vertex Coloring in 2010. Then, Dinneen and Wei [6], in 2013, analyzed a dynamic mutation with two different local searches on some artificially created functions; and in the same year, the authors [5] analyzed a (1+1) Adaptive MA on the clique problem and showed that, for any local optima that is hard to escape, the (1+1) Adaptive MA is expected to overcome the local optima super-polynomially faster than the basic (1+1) EA. A few latest surveys of runtime analysis on MAs and EAs can be found in [3], [8], [14], [15].

These runtime analyses explain the success of some experiments. For example, an adaptive mutation approach in an MA [4] gained success in experiments, and theoretical evidences were studied in [6], [5]. However, runtime analysis progress still lags behind the experimental approach. In particular, many experimental studies observed that, for a single problem, it is possible to construct different fitness functions, and running the same algorithm on these fitness functions can result in a huge performance gap, for example [1], [2], [19], [26]. This observation has motivated a great amount of studies on the fitness landscape theory, as presented in a few recent surveys [16], [18]. However, to the best of our knowledge, no runtime analyses show the evidence for this observation, which is the aim of this paper.

The paper is structured as follows. In Section II, we will formalize two fitness functions, f_{OL} and f_{OPL}, for solving the Clique Problem. Then propose the (1+1) Restart Memetic Algorithm (RMA). Next, in Section III, we will construct a family of graphs (G_1), and show that, to solve the Clique Problem on the graph family G_1, the (1+1) RMA on f_{OPL} is expected to take $O(n^2)$ fitness evaluations, but the (1+1) RMA on f_{OL} is expected to take a super-polynomial number of fitness evaluations. In Section IV, we will construct another family of graphs (G_2) and show the opposite trend, i.e. the (1+1) RMA on f_{OPL} is expected to take a super-polynomial number of fitness evaluations, but the (1+1) RMA on f_{OL} is expected to take $O(n^{2.5})$ fitness evaluations. Finally, a conclusion will be given in Section V.

II. Algorithm Definitions

A. The Maximum Clique Problem

A *clique* of a graph is a subset of vertices from this graph such that every two vertices in the subset are connected by an edge. The *Clique Problem* is the NP-hard problem of finding the largest size of a clique in a graph. For a given graph $G = (V = \{v_1, v_2, ..., v_n\}, E)$, a bit string $x = (x_1, x_2, ..., x_n) \in \{0,1\}^n$ defines a clique potential solution (an induced subgraph) where $x_i = 1$ represents that the vertex v_i is selected. We say x represents a clique if each selected vertex in x is connected to all other selected vertices in x, i.e. $\{(v_i, v_j) \mid x_i = x_j = 1 \text{ and } i \neq j\} \subseteq E$.

978-1-4799-1488-3/14/$31.00 ©2014 IEEE

There are some runtime analyses studying EAs on the Clique Problem, in which different fitness functions have been studied, e.g., Storch in 2006 [20] used the function f_1 as follows:

$$f_1(x) = \begin{cases} \texttt{ONES}(x), & \text{if x represents a clique,} \\ -\infty, & \text{otherwise;} \end{cases}$$

and the same author in 2007 [21] constructed another function f_2 as follows:

$$f_2(x) = \begin{cases} \texttt{ONES}(x), & \text{if x represents a clique,} \\ -\texttt{ONES}(x), & \text{otherwise.} \end{cases}$$

Note that even though these two fitness functions both encode the problem of finding a maximum clique, and both can be calculated within $O(n^2)$ time, the function $f_2(x)$ provides some additional information which is very important for MAs. In other words, if the mutation jumps to a bit string that does not represent a clique, $f_2(x)$ can guide the local search quickly discover a clique, but $f_1(x)$ cannot. In this paper, we construct another two fitness functions, $f_{\texttt{OL}}$ and $f_{\texttt{OPL}}$, for solving the Clique Problem.

Definition 1. The fitness function $f_{\texttt{OL}}$ (ONES, LACKEDGES) is defined as follows:

$$f_{\texttt{OL}}(x) = \begin{cases} \texttt{ONES}(x), & \text{if x represents a clique,} \\ -\texttt{LACKEDGES}(x), & \text{otherwise,} \end{cases}$$

where $\texttt{ONES}(x)$ is the number of ones in x; and $\texttt{LACKEDGES}(x)$ is the number of missing edges such that the subgraph becomes a clique.

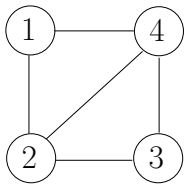
Definition 2. The fitness function $f_{\texttt{OPL}}$ (ONES, P[otential]V[ertices], LACKEDGES) is defined as follows:

$$f_{\texttt{OPL}}(x) = \begin{cases} \texttt{ONES}(x) + \texttt{PV}(x)/n, & \text{if x represents a clique,} \\ -\texttt{LACKEDGES}(x), & \text{otherwise,} \end{cases}$$

where the additional function $\texttt{PV}(x)$ is the number of zeros in x such that, when each of these zeros is flipped individually, a larger clique is obtained.

Example 3. For a given graph G, displayed below, we have:

1) If $x = (1101)$, then $f_{\texttt{OL}}(x) = f_{\texttt{OPL}}(x) = 3$ because x represents a clique that consists of vertices 1, 2 and 4.
2) If $x = (1111)$, then $f_{\texttt{OL}}(x) = f_{\texttt{OPL}}(x) = -1$ because we need to add one edge $(1, 3)$ to the graph so that the bit string represents a clique.
3) If $x = (0101)$, then $f_{\texttt{OL}}(x) = 2$ and $f_{\texttt{OPL}}(x) = 2.5$. Since x represents a clique that consists of vertices 2 and 4, $\texttt{ONES}(x) = 2$. There are two potential vertices 1 and 3, such that flipping either the first or the third bit will obtain a larger clique, so $\texttt{PV}(x)/n = 0.5$.



Note that if x does not represent a clique, since the function LACKEDGES provides more information than the function ONES, the functions $f_{\texttt{OL}}$ and $f_{\texttt{OPL}}$ can guide the local search to find a clique faster than the functions f_1 and f_2. Besides, if x is a subset of a larger clique, and has many neighbor cliques of which the clique size are all larger than x by one, then $f_{\texttt{OL}}$ cannot distinguish these neighbors, while $f_{\texttt{OPL}}$ can distinguish some of them because it provides an additional one step looking forward.

Also note that both $f_{\texttt{OL}}$ and $f_{\texttt{OPL}}$ can be calculated within $O(n^2)$ time. This is why we claim that the fitness change is small, due to the fact that we can also construct another fitness function that looks two steps forward or even more, which may not be calculated within $O(n^2)$ time. For example, a fitness function that looks n steps forward can guide the BILS (Algorithm 5) directly moving towards a maximum clique, but that fitness function is too expensive to calculate.

A maximum clique for a graph G is a global optimal solution x that maximizes both $f_{\texttt{OL}}(x)$ and $f_{\texttt{OPL}}(x)$.

B. Algorithms to be analyzed

Algorithm 4. (1+1) RMA.

1) Set the number of generations $\texttt{gen} := 0$, $x = 0^n$.
2) $y := x$. Flip every bit in y with probability $1/n$.
3) $z := \texttt{LocalSearch}(y)$.
4) If $f(z) \geq f(x)$ then $x := z$.
5) $\texttt{gen} := \texttt{gen} + 1$.
6) If $\texttt{gen} = \lambda$ then go to step 1.
7) Stop if any stopping criterion is met, otherwise, go to step 2.

Note that the algorithm will restart in every λ generations. We will analyze the impact of λ on the algorithm's ability to find a global optimal solution. The local search $\texttt{LocalSearch}(y)$ can have a local search pool that contains many different local searches, and the algorithm can dynamically or adaptively choose one of them to execute. Since we want to focus on the fitness functions, we only analyze a simple *Best-Improvement Local Search* (BILS), which uses a steepest ascent pivot rule.

Algorithm 5. Best-Improvement Local Search (BILS). For a given string $x \in \{0, 1\}^n$:

1) $\texttt{BestNeighborSet} := \left\{ \begin{array}{l} y \mid f(y) > f(x), \texttt{Hamming}(x, y) = 1, \text{ and} \\ \forall z \text{ with } \texttt{Hamming}(x, z) = 1 : f(y) \geq f(z) \end{array} \right\}$.
2) Stop and return x if $\texttt{BestNeighborSet} = \emptyset$.
3) x is randomly choosen from BestNeighborSet.
4) Go to step 1.

Note, $\texttt{Hamming}(x, y)$ is the number of different bits between x and y. Since the BILS will evaluate all n neighbors and then randomly select one of the best neighbors, and according to the fitness functions $f_{\texttt{OL}}$ in Definition 1 and $f_{\texttt{OPL}}$ in Definition 2, we have:

1) If the bit string after the mutation represents a clique, the BILS will keep flipping bits from zeros to ones until it finds a local optimal clique, which takes at most n^2 fitness evaluations.
2) If the bit string after the mutation does not represent a clique, the BILS will keep flipping bits from ones to zeros until it finds a clique, which takes at most n^2 fitness evaluations.

Therefore, the BILS will stop on both fitness functions $f_{\texttt{OL}}$ and $f_{\texttt{OPL}}$ within $2n^2$ fitness evaluations.

In the rest of this paper, we will use (1+1) RMA_OL to denote that the (1+1) RMA is running on the fitness function $f_{\texttt{OL}}$, and (1+1) RMA_OPL to denote that the (1+1) RMA is running on the fitness function $f_{\texttt{OPL}}$.

Due to space limitations we provide proof sketches for the main theorems and redirect the reader to [27] for full proofs or [29] for similar proof techniques related to these same graph families for local search comparison (instead of fitness function comparison).

III. A FAMILY OF GRAPHS ON WHICH THE (1+1) RMA_OPL OUTPERFORMS THE (1+1) RMA_OL

In this section, we will construct a family of graphs G_1, and show that the (1+1) RMA_OPL is expected to find the maximum clique on any graph of the family G_1 within a polynomial number of fitness evaluations, while the (1+1) RMA_OL is expected to take a super-polynomial number of fitness evaluations to find the maximum clique on any graph of the family G_1.

Definition 6. The graph $G_1(t)$ has $n = t(2t+2)$ vertices for the variable t. We separate all vertices into $(2t+2)$ disjoint sets $V_0, V_1, \cdots, V_{2t+1}$ such that $V_i \cap V_j = \emptyset$ and $|V_i| = t$ for $0 \leq i \leq 2t+1$, $0 \leq j \leq 2t+1$ and $i \neq j$. We use $v_{i,k}$ to refer the k-th vertex in V_i $(0 \leq k \leq t-1)$. To make it easier to understand the edge set E, we first assume it is a complete graph, and then delete edges according to the following rules:

1) Delete the edge between $v_{i,k}$ and $v_{j,k}$ for all variables i, j, k with $\lfloor i/2 \rfloor \neq \lfloor j/2 \rfloor$, $0 \leq k \leq t-1$, $0 \leq i \leq 2t+1$ and $0 \leq j \leq 2t+1$.
2) Delete the edge between $v_{i,t-1}$ and $v_{j,t-1}$ for all variables i, j with $2 \leq i \leq 2t+1$ and $\lfloor i/2 \rfloor = \lfloor j/2 \rfloor$.
3) Delete the edge between $v_{1,k}$ and $v_{j,l}$ for all variables j, k, l with $2 \leq j \leq 2t+1$, $0 \leq k \leq t-1$, $0 \leq l \leq t-1$ and $k \equiv (l-1) \bmod t$.

Visually we fill all vertices from the sets $V_0, V_1, \cdots, V_{2t+1}$ into a $(2t+2)$-by-t matrix with each set V_i being the i-th row vector, as shown in Figure 1. Hence, $v_{i,j}$ is the vertex in row i and column j. We use $V(x)$ to denote the set of vertices x has chosen; recall $x = (x_1, x_2, \ldots, x_n)$. Then we use $v_{i,k}(x) = 1$ (a solid circle in Figure 1) to denote that $v_{i,k} \in V(x)$, and $v_{i,k}(x) = 0$ (a hollow circle in Figure 1) otherwise. In Figure 1, $v_{0,0}(x) = 1$ and $v_{0,1}(x) = 0$.

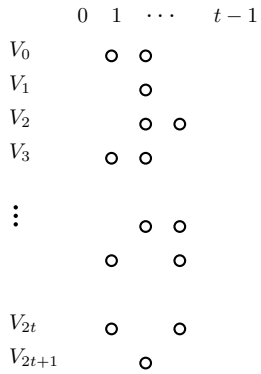


Fig. 1. A $(2t+2)$-by-t matrix representation for $G_1(t)$ and $G_2(t)$.

Figure 2 demonstrates the edge deleting rules for the graph $G_1(t)$, in which dashed lines denote that the edges are deleted. In detail, Case 1 in Figure 2 is an example of deleting edges that are connected to the vertices $v_{0,0}$ and $v_{1,0}$, according to Rule 1 in Definition 6; Case 2 in Figure 2 demonstrates all edges that need to be deleted, according to Rule 2 in Definition 6; and Case 3 in Figure 2 is an example of deleting edges that are connected to the vertices $v_{1,0}$ and $v_{1,1}$, according to Rule 3 in Definition 6.

Note that the idea of the edge rules in Definition 6 is to limit the graph $G_1(t)$ to only have one maximum clique of $2t$ vertices, which is the top two rows in Figure 1; meanwhile, it has many misleading local optimal cliques of $(2t-1)$ vertices in the bottom $2t$ rows in Figure 1. Observe the $\texttt{PV}$ function in $f_{\texttt{OPL}}$ will guide the BILS to add vertices in the top two rows, which is the maximum clique. On the other hand, since $f_{\texttt{OL}}$ cannot provide the "one step looking forward" function, and because the size of the bottom $2t$ rows is t times the size of the top two rows, the BILS on the $f_{\texttt{OL}}$ is more likely to become trapped into a local optimal that contains many vertices in the bottom $2t$ rows. Overall, we aim to show that the (1+1) RMA_OPL drastically outperforms the (1+1) RMA_OL on this family of graphs when finding the maximum clique.

Claim 7. *For the graph $G_1(t)$, if the bit string x represents a clique, $V(x)$ has at most two vertices in each column, i.e. $\forall k, 0 \leq k \leq t-1 : \sum_{i=0}^{2t+1} v_{i,k}(x) \leq 2$.*

Proof. Due to Rule 1 in Definition 6, every vertex is connected to at most one vertex in the same column. □

Claim 8. *For the graph $G_1(t)$, if the bit string x represents a maximal clique, $V(x)$ contains at least one vertex in each column, i.e. $\forall k, 0 \leq k \leq t-1 : \sum_{i=0}^{2t+1} v_{i,k}(x) \geq 1$. Furthermore, $V(x)$ contains at least t vertices, i.e. $|V(x)| \geq t$.*

Proof. Recall the rules in Definition 6, for each column k $(0 \leq k \leq t-1)$, the vertex $v_{0,k}$ is connected to all vertices in

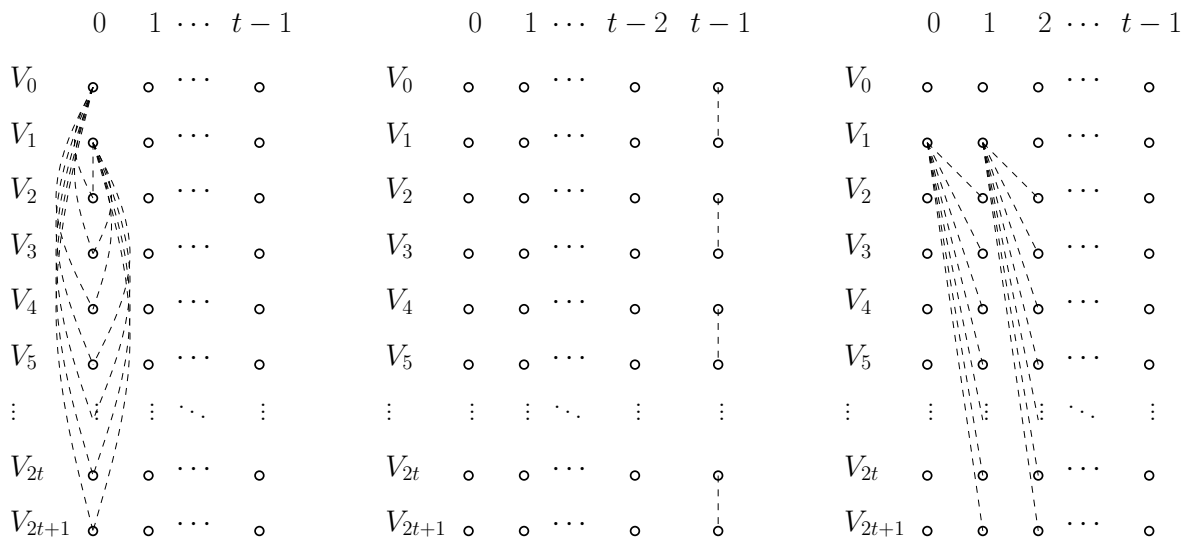


Fig. 2. Illustrating Cases 1, 2 and 3 for the graph $G_1(t)$ of Definition 6.

other columns. Thus, $V(x)$ will either contain some vertices from $\bigcup_{i=1}^{2t+1}\{v_{i,k}\}$, or contain the vertex $v_{0,k}$. Furthermore, since $V(x)$ contains at least one vertex in each column, and there are t columns, $V(x)$ contains at least t vertices. □

Claim 9. *The only maximum clique of the graph $G_1(t)$ is $V_{global} = V_0 \cup V_1$, which contains $2t$ vertices.*

Proof. Firstly, due to the edge rules in Definition 6, there exists a $2t$ clique of $V_0 \cup V_1$. Secondly, according to Claim 7, any clique can have at most two vertices in each column, and there are t columns in the graph, thus, the maximum clique size can not be larger than $2t$. This means that $V_0 \cup V_1$ is a maximum clique.

Now we show that $V_0 \cup V_1$ is the only $2t$ clique in the graph. We assume that there is another $2t$ clique y. Due to Claim 7, $V(y)$ must contain two vertices in each column to be a $2t$ clique. Furthermore, due to Rule 1 and Rule 2 in Definition 6, $V(y)$ can only contain $v_{0,t-1}$ and $v_{1,t-1}$ in the last column. Therefore, $V(y)$ contains some vertices in $V_0 \cup V_1$. Since $V(y) \neq V_0 \cup V_1$, and $|V(y)| = |V_0 \cup V_1| = 2t$, $V(y)$ must contain some vertices in $\bigcup_{i=2}^{2t+1} V_i$. So, there exists at least one column k, such that $V(y)$ must contain vertices from $V_0 \cup V_1$ in this column, and which must also contain vertices from $\bigcup_{i=2}^{2t+1} V_i$ in the next column. Then in the column k, $V(y)$ can not contain the vertex $v_{1,k}$, due to Rule 3 in Definition 6. Hence, $V(y)$ only contains one vertex in the column k, which conflicts with the requirement that $V(y)$ must contain two vertices in each column. Thus y does not exist. □

Theorem 10. *For any constant λ independent of n, the (1+1) RMA_OPL is expected to find the maximum clique of the graph $G_1(t)$ within $O(n^2)$ fitness evaluations.*

Proof sketch. Since the algorithm starts with $x = 0^n$, and the mutation probability is $1/n$, the probability that the mutation does not flip any bits is $(1-1/n)^n$, which approaches $1/e$ as n increases. Therefore, within $\Theta(1)$ restarts, we expect to have a mutation that gets the bit string $y = 0^n$.

From 0^n, the function PV in f_{OPL} will always guide the BILS to add vertices from the top two rows into the clique, until it finds the maximum clique of $V_0 \cup V_1$. This is because that, suppose the current clique is $V(y)$, for any column k in the graph such that $V(y)$ does not contain any vertex in this column, we have:

1) If $V(y)$ includes the vertex of $v_{1,k-1}$, then Rule 3 in Definition 6 will prevent the BILS from adding any vertex, that is in the bottom $2t$ rows and that is also in the column k, into $V(y)$ to obtain a larger clique.
2) If $V(y)$ does not include the vertex of $v_{1,k-1}$, then Rule 3 in Definition 6 will distinguish the neighbors of y, so adding the vertex $v_{0,k}$ into $V(y)$ will have a better fitness value than adding any vertex in the bottom $2t$ rows from the column k.

Therefore, after the BILS adds the vertex $v_{0,k}$, it will not consider adding any vertex in the bottom $2t$ rows of the column k due to Rule 1 in Definition 6.

Recall that the algorithm will restart in every $\Theta(1)$ generations, and in each generation, the BILS will stop on the f_{OPL} within $2n^2$ fitness evaluations (see Algorithm 5), so the (1+1) RMA_OPL is expected to find the bit string 0^n within $O(n^2)$ fitness evaluations. Then, the BILS will keep checking all n neighbors of the current clique, and adding one vertex from the top two rows into the current clique, until it finds the maximum clique. This will take $t \cdot n = O(n^{1.5})$ fitness evaluations (note $n = t(2t+2)$). Overall, the (1+1) RMA_OPL is expected to find the maximum clique of the graph $G_1(t)$ within $O(n^2)$ fitness evaluations. □

We now introduce the Global Gambler's Ruin that we will use to prove Theorem 12.

Theorem 11. *Global Gambler's Ruin (Happ et al. [7])*

Let X_0, X_1, X_2,$\cdots$ be random variables describing a Markov process over the state space $\mathbb{N}_0$. For constants $a, b \in \mathbb{R}$ with $0 \leq a < b \leq 1$ let the random variable T denote the earliest point in time $t \geq 0$ that satisfies $X_t \leq an$.

If there exist constants $\delta > 1$ and $C > 0$ such that the three conditions

1) $P[X_0 \geq bn] = 1 - 2^{-\Omega(n)}$,
2) $P[X_{t+1} - X_t = j \mid X_t] \geq \delta^j \cdot P[X_{t+1} - X_t = -j \mid X_t]$ *for all $j \geq 1$, $t \geq 0$, and $an < X_t < bn$,*
3) $\sum_{j \geq 1} \delta^j \cdot P[X_{t+1} - X_t = -j \mid X_t] \leq C$ *for all $t \geq 0$ and $X_t \geq bn$*

hold then $T \geq \delta^{1/3 \cdot (b-a)n}$ with probability $1 - 2^{-\Omega(n)}$.

Theorem 12. *For the variable λ, no matter if it is static, adjusted dynamically or adaptively, the probability of the (1+1) RMA_OL finding the maximum clique of the graph $G_1(t)$ within a polynomial number of fitness evaluations is super-polynomially close to zero.*

Proof sketch. We prove the theorem by showing that the three conditions in Theorem 11 hold with probabilities super-polynomially close to one, so that the probability of the algorithm finding the maximum clique within a polynomial number of fitness evaluations is super-polynomially close to zero.

Let X_p denote the number of columns in which the current clique contains vertices in the bottom $2t$ rows at time p. Note that $0 \leq X_p \leq t$ and $n = t(2t+2)$. According to Claim 8, any local optimum clique contains at least one vertex in each column, then the target state is $X_T = 0$, which denotes that the local optimum clique does not contain any vertex in the bottom $2t$ rows, which is the maximum clique of $V_0 \cup V_1$. Let $a = 1/4$, $b = 1/2$, the constants $\delta = 2$ and $C = \sum_{j \geq 1} j \cdot 2^j/(j!)$.

Part 1. Condition 1 holds with a probability super-polynomially close to one. Firstly, since the algorithm starts or restarts from 0^n, the first clique found by the algorithm from 0^n has a constant number of vertices in the top two rows with a probability super-polynomially close to one. This is because that the probability of the mutation flipping $k = \omega(1)$ bits in the top two rows is $\binom{2t}{k}(1/n)^k (1-1/n)^{2t-k} \leq \frac{1}{k!}\left(\frac{2t}{n}\right)^k$, which is super-polynomially close to zero due to $n = t(2t+2)$.

Secondly, the first local optimum clique found by the algorithm, after the start or after each restart, has $O(\log t)$ vertices in the top two rows with a probability super-polynomially close to one. This is because that a) the fitness function f_{OL} cannot help the BILS to distinguish those neighbors that all improve the clique size by one; and b) the size of the bottom $2t$ rows is t times the size of the top two rows, i.e., most neighbors that improve the clique size by one are obtained by adding one vertex from the bottom $2t$ rows. Since the local optimal clique has at least one vertex in each column (Claim 8) and any clique has at most two vertices in the same column (Claim 7), then with a probability super-polynomially close to one, there are more than $(1/2)t$ columns from which the first local optimal clique contains vertices from the bottom $2t$ rows, i.e., $X_0 \geq bn$.

Part 2. Condition 2 holds. Suppose the current state $X_p \in (an, bn)$. Let $\delta = 2$. Then we show that $P[X_{p+1} - X_p = j \mid X_p] \geq \delta^j \cdot P[X_{p+1} - X_p = -j \mid X_p]$. Since $X_p \in (an, bn)$, there are more than $t/2$ columns that the current clique contains vertices from the top two rows, and less than $t/2$ columns that the current clique contains vertices from the bottom $2t$ rows. And due to the size of the bottom $2t$ rows is t times the size of the top two rows, the total number of cliques that can achieve $P[X_{p+1} - X_p = j]$ is at least t^j times the total number of cliques that can achieve $P[X_{p+1} - X_p = -j]$. Furthermore, the probabilities to obtain each of these cliques are the same. Therefore, $P[X_{p+1} - X_p = j \mid X_p] \geq t^j \cdot P[X_{p+1} - X_p = -j \mid X_p] > \delta^j \cdot P[X_{p+1} - X_p = -j]$.

Part 3. Condition 3 holds. Note that for each column that undergoes a positive drift or a negative drift, the mutation needs to flip at least one bit in that column. Therefore, to jump j steps towards the target state, i.e., $X_{p+1} - X_p = -j$, the mutation needs to flip at least j bits. Then we say that the probability of $X_{p+1} - X_p = -j$ is at most $\sum_{q=j}^{n} \binom{n}{q} \cdot (1/n)^q \leq \sum_{q=j}^{n} 1/(q!)$. Therefore, $\sum_{j \geq 1} \delta^j P[X_{p+1} - X_p = -j \mid X_p] \leq \sum_{j \geq 1} j \cdot \delta^j/(j!) = \sum_{j \geq 1} j \cdot 2^j/(j!)$, which is a constant (note that $j!$ grows faster than 4^j when $j > 4$). □

IV. A FAMILY OF GRAPHS ON WHICH THE (1+1) RMA_OL OUTPERFORMS THE (1+1) RMA_OPL

In this section, we will construct a family of graphs G_2 and show that the (1+1) RMA_OL is expected to find a maximum clique within a polynomial number of fitness evaluations, while the (1+1) RMA_OPL is expected to take a super-polynomial number of fitness evaluations to find a maximum clique.

Definition 13. The graph $G_2(t)$ has the same number of vertices as the graph $G_1(t)$, but the edge connections are different. It has $n = t(2t+2)$ vertices for the variable t. We separate all vertices into $(2t+2)$ disjoint sets $V_0, V_1, \ldots, V_{2t+1}$ such that $V_i \cap V_j = \emptyset$ and $|V_i| = t$ for $0 \leq i \leq 2t+1$, $0 \leq j \leq 2t+1$ and $i \neq j$. We use $v_{i,k}$ to refer the k-th vertex in V_i $(0 \leq k \leq t-1)$. To make it easier to understand the edge set E, we first assume it is a complete graph, and then delete edges according to the following rules:

1) Delete the edge between $v_{i,k}$ and $v_{j,k}$ for all variables i, j, k with $0 \leq k \leq t-1$, $0 \leq i \leq 2t+1$, $0 \leq j \leq 2t+1$ and $\lfloor i/2 \rfloor \neq \lfloor j/2 \rfloor$.
2) Delete the edge between $v_{0,t-1}$ and $v_{1,t-1}$.
3) Delete the edge between $v_{i,k}$ and $v_{j,l}$ for all variables i, j, k, l with $2 \leq i \leq 2t+1$, $2 \leq j \leq 2t+1$, $\lfloor i/2 \rfloor = \lfloor j/2 \rfloor$, $0 \leq k \leq t-1$, $0 \leq l \leq t-1$ and $k \neq l$.
4) Delete the edge between $v_{1,k}$ and $v_{j,l}$ for all variables j, k, l with $2 \leq j \leq 2t+1$, $0 \leq k \leq t-1$, $0 \leq l \leq t-1$ and $k \equiv (l-1) \bmod t$.
5) Delete the edge between $v_{1,k}$ and $v_{j,l}$ for all variables j, k, l with $2 \leq j \leq 2t+1$, $0 \leq k \leq t-1$ and $l \in \{k - k \bmod \lfloor \log t \rfloor, k + \lfloor \log t \rfloor - k \bmod \lfloor \log t \rfloor\}$ where $0 \leq l \leq t-1$.

Again we fill all vertices from the sets $V_0, V_1, \ldots, V_{2t+1}$ into a $(2t+2)$-by-t matrix with each set V_i being the i-th row vector, as shown in Figure 1. Hence, $v_{i,j}$ is the vertex in row i and column j. We use $V(x)$ to denote the set of vertices $x = (x_1, x_2, \ldots, x_n)$ has chosen. Then we use $v_{i,k}(x) = 1$ (a solid circle in Figure 1) to denote that $v_{i,k} \in V(x)$, and $v_{i,k}(x) = 0$ (a hollow circle in Figure 1) otherwise.

Figure 3 demonstrates the edge deleting rules for the graph $G_2(t)$, in which dashed lines denote that the edges are deleted. In detail, Case 1 in Figure 3 is an example of deleting edges that are connected to the vertices $v_{0,0}$ and $v_{1,0}$, according to Rule 1 in Definition 13; Case 3 in Figure 3 is an example of deleting edges that are connected to the vertices $v_{2,0}$ and $v_{3,0}$, according to Rule 3 in Definition 13; Case 4 in Figure 3 is an example of deleting edges that are connected to the vertices $v_{1,0}$ and $v_{1,1}$, according to Rule 4 in Definition 13; and Case 5 in Figure 3 is an example of deleting edges that are connected to the vertices $v_{1,1}$ and $v_{\lfloor \log t \rfloor - 1,1}$, according to Rule 5 in Definition 13.

Note that according to Definition 13, the graph $G_2(t)$ has many maximum cliques of $2t$ vertices, in fact, any maximal clique that does not contain any vertex in the top two rows is a maximum clique (note that Rule 2 in Definition 13 is different from Rule 2 in Definition 6). Similarly, the PV function in f_{OPL} will guide the BILS to add vertices in the top two rows (note that the BILS on f_{OPL} will be less interested in the bottom $2t$ rows due to Rule 3 in Definition 13). However, the clique of the top two rows now becomes a local optimal clique, thus the BILS on f_{OPL} will get trapped easily. On the other hand, due to the size of the bottom $2t$ rows is t times the size of the top two rows, the FILS on f_{OL} is more likely to discover a maximum clique in the bottom $2t$ rows. Furthermore, we add Rule 5 in Definition 13 to prevent the (1+1) RMA_OPL from escaping a local optimal clique and moving towards a global optimal clique.

Claim 14. *For the graph $G_2(t)$, if the bit string x represents a clique, $V(x)$ has at most two vertices in each column, i.e. $\forall k, 0 \leq k \leq t-1 : \sum_{i=0}^{2t+1} v_{i,k}(x) \leq 2$.*

Proof. Due to Rule 1 in Definition 13, every vertex is connected to at most one vertex in the same column. □

Claim 15. *For the graph $G_2(t)$, if the bit string x represents a maximal clique, $V(x)$ contains at least one vertex in each column, i.e. $\forall k, 0 \leq k \leq t-1 : \sum_{i=0}^{2t+1} v_{i,k}(x) \geq 1$. Furthermore, $V(x)$ contains at least t vertices, i.e. $|V(x)| \geq t$.*

Proof. Recall the rules in Definition 13, for each column k ($0 \leq k \leq t-1$), the vertex $v_{0,k}$ is connected to all vertices in other columns. Thus $V(x)$ will either contain some vertices from $\bigcup_{i=1}^{2t+1}\{v_{i,k}\}$ or contain the vertex $v_{0,k}$. Furthermore, due to $V(x)$ contains at least one vertex in each column, and there are t columns, $V(x)$ contains at least t vertices. □

Claim 16. *For the graph $G_2(t)$, if the bit string x represents a maximal clique, then for each row V_i with $2 \leq i \leq 2t+1$, $V(x)$ contains at most one vertex in V_i, i.e., $\forall i, 2 \leq i \leq 2t+1 : |V(x) \cap V_i| \leq 1$.*

Proof. The claim can be induced from Rule 3 in Definition 13. □

Claim 17. *For any bit string x that represents a maximal clique of the graph $G_2(t)$, if $V(x)$ does not contain any vertices in $V_0 \cup V_1$, then it contains $2t$ vertices and is a maximum clique. Otherwise, $V(x)$ contains less than $2t$ vertices.*

Proof. Firstly, due to Claim 14, any clique can have at most two vertices in each column, and there are t columns in the graph, thus the maximum clique size can not be larger than $2t$. Also, according to the edge rules in Definition 13, if a maximal clique $V(x)$ does not contain any vertices in $V_0 \cup V_1$, then it must contain exactly two vertices in each column. Thus, it has $2t$ vertices, which is a maximum clique.

Secondly, if $V(x)$ contains some vertices in $V_0 \cup V_1$, x belongs to one of the following two cases:

1) $V(x)$ only contains vertices in $V_0 \cup V_1$. Therefore, it can only contain one vertex in the last column, due to Rule 2 in Definition 13. In this case, the clique size of x will be less than $2t$.
2) $V(x)$ also contains some vertices in $\bigcup_{i=2}^{2t+1} V_i$. Therefore, there exists a column k, such that $V(x)$ contains vertices from $V_0 \cup V_1$ in this column, and contains vertices from $\bigcup_{i=2}^{2t+1} V_i$ in the next column. Hence, $V(x)$ contains only the vertex $v_{0,k}$ in the column k due to Rule 4 in Definition 13. In this case, the clique size of x will be less than $2t$. □

Definition 18. For the graph $G_2(t)$ and a maximum clique x, a section is called *hard-to-escape* if a) it contains $\lfloor \log t \rfloor$ consecutive columns, that starts from a column k_1 with $k_1 \bmod \lfloor \log t \rfloor = 0$; and b) x contains exact two vertices in $V_0 \cup V_1$ from each column of this section.

Claim 19. *For each* hard-to-escape *section in Definition 18, if the mutation and the following local search wants to find another clique of which the clique size is not decreased, but has less vertices in $V_0 \cup V_1$ in this section, then the only way is to remove all vertices in $V_0 \cup V_1$ in this section, and add the same number of vertices in $\bigcup_{i=2}^{2t+1} V_i$ in this section. Furthermore, the probability to achieve this jump is super-polynomially close to zero.*

Proof. Due to Rules 1 and 4 in Definition 13, there is no way to achieve a small jump that only changes a subset of this section of columns, from choosing vertices in the top two rows to choose vertices in the bottom $2t$ rows, without decreasing the clique size. Note that for each column k in this section, there are three vertices ($v_{0,k}$, $v_{1,k}$ and $v_{1,(k-1) \bmod t}$) are not adjacent to the vertices in the bottom $2t$ rows in the column k.

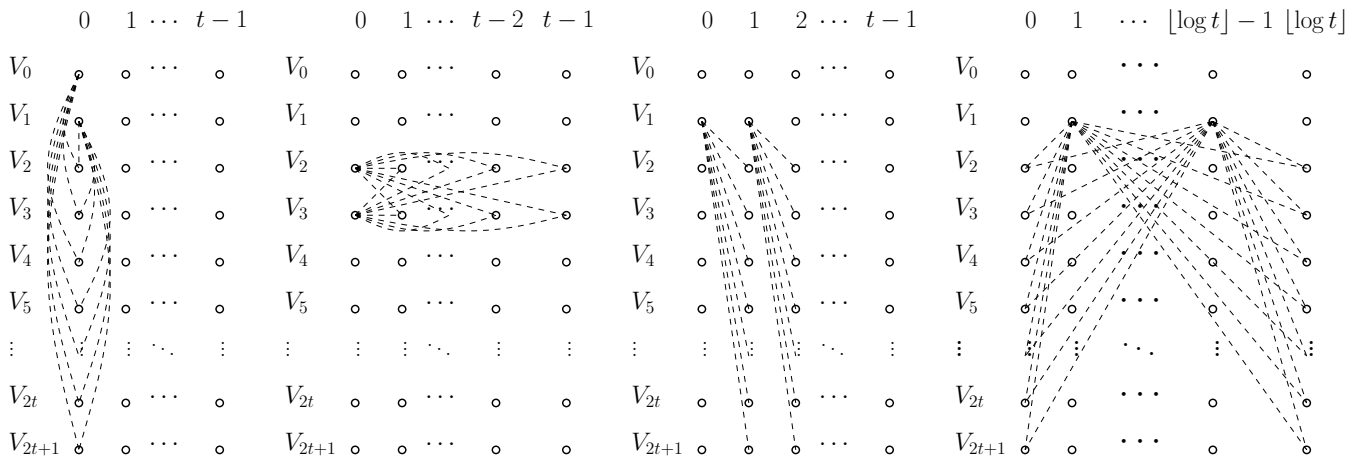


Fig. 3. Illustrating Cases 1, 3, 4 and 5 for the graph $G_2(t)$ of Definition 13.

In addition, due to Rule 5 in Definition 13, both edge columns of this section cannot achieve a jump, from choosing vertices in the top two rows to choose vertices in the bottom $2t$ rows, without decreasing the clique size.

Therefore, the only way is to remove all vertices in $V_0 \cup V_1$ in this section, and add the same number of vertices in $\bigcup_{i=2}^{2t+1} V_i$ in this section. Furthermore, due to the graph is dense, there is no way that the mutation only flips a constant number of bits to result in the following local search flips the remaining $O(\log t)$ bits. Therefore, the probability of achieving this jump to change this *hard-to-escape* section is super-polynomially close to zero. □

Theorem 20. *For any constant λ, the (1+1) RMA_OL is expected to find a maximum clique of the graph $G_2(t)$ within $O(n^{2.5})$ fitness evaluations.*

Proof sketch. Part 1. Same as the proof of Theorem 10, within $\Theta(1)$ restarts, we expect to have a mutation that produces the bit string 0^n. Also, because we will restart the algorithm after each λ generations, and the BILS will take at most $2n^2$ fitness evaluations in each generation (see Algorithm 5), we expect to have a mutation that does not flip any bits within $O(n^2)$ fitness evaluations.

Part 2. From 0^n, the BILS will keep adding vertices into the current clique, and it has a probability $1/(t+1)$ to reach a local optimum that does not contain any vertex in the top two rows, which is a maximum clique due to Claim 17. If we look at each column, and focus on the first vertex in the column that the BILS will add to the current clique, there must exist a sequence of columns $C = (c_1, c_2, \ldots, c_t)$. In the column c_1, the BILS has a probability $t/(t+1)$ to add a vertex from the bottom $2t$ rows into the current clique. Suppose c_1 successes, i.e., the BILS adds a vertex from the bottom $2t$ rows in the column c_1. Then, in the column c_2, due to Rule 3 in Definition 13, the BILS has a probability $(t-1)/t$ to add a vertex from the bottom $2t$ rows into the current clique. Overall, the probability that the BILS keeps adding vertices from the bottom $2t$ rows for every columns in C is

$$\frac{t}{t+1}\frac{t-1}{t}\cdots\frac{1}{2} = 1/(t+1).$$

Therefore, if we have $O(t)$ mutations that all produce the bit string 0^n, the BILS is expected to find a maximum clique. Also, according to Part 1, we expect to have a mutation that produces the bit string 0^n within $O(n^2)$ fitness evaluations. Hence, the algorithm is expected to find a maximum clique within $O(tn^2)$ fitness evaluations, which is $O(n^{2.5})$ since $n = t(2t+2)$. □

Theorem 21. *For the variable λ, no matter it is static, adjusted dynamically or adaptively, the probability of the (1+1) RMA_OPL finding a maximum clique of the graph $G_2(t)$ within a polynomial number of fitness evaluations is super-polynomially close to zero.*

Proof sketch. We prove this by showing that, from the start or from each restart, the (1+1) RMA_OPL is expected to become trapped in a local optimal clique and take a super-polynomial number of fitness evaluations to find a global optimal clique. In detail, our proof consists of three parts, with each part having a failure probability super-polynomially close to zero.

Part 1. The first clique found by the algorithm after the start or each restart has no more than $\log n$ vertices in the bottom $2t$ rows with a probability super-polynomially close to one. This is because that the mutation from 0^n flips no more than $\log n$ bits with a probability super-polynomially close to one. Hence, either this bit string represents a clique, or the BILS will flip some bits from ones to zeros in order to find a clique. Therefore, part 1 holds.

Part 2. After the first clique has been found, the BILS will keep adding vertices from the first row until there is only one column that the current clique does not contain any vertex in this column (this is because that the PV function in f_{OPL} will distinguish the neighbors of the current clique, and guide the BILS to add a vertex from the first row into the current clique). Therefore, with a probability super-polynomially close

to one, there are at most $\log t$ columns from which the first local optimal contains vertices from the bottom $2t$ rows.

Part 3. Assume Part 2 has succeeded, there are $t - \log t$ columns from which the current local optimal clique has vertices from the top two rows. Therefore, there are $\omega(1)$ *hard-to-escape* sections of Definition 18. Then, due to Claim 19, the probability of the algorithm finding a maximum clique within a polynomial number of fitness evaluations is super-polynomially close to zero. □

V. Conclusion and Future Work

This paper provided the theoretical evidence that a small change of the fitness function can result in a huge performance gap in terms of finding a global optimum solution. It also shows that the fitness function that gives the best results in an MA on the Clique Problem is entirely instance specific. In detail, we have formalized a (1+1) Restart Memetic Algorithm with a Best-Improvement Local Search, and run them on two different fitness functions, f_{OL} and f_{OPL}, to solve the Clique Problem respectively. We then constructed two families of graphs, G_1 and G_2, and showed that, for the first family of graphs G_1, the (1+1) RMA on the fitness function f_{OPL} drastically outperforms the (1+1) RMA on the fitness function f_{OL}, and vice versa for the second family of graphs G_2.

For future work, we suggest analyzing the runtime performance of MAs on more fitness functions to solve the same problem, and use a dynamic or adaptive strategy that will decide which fitness function should be used.

References

[1] F. Alabsi and R. Naoum, "Fitness function for genetic algorithm used in intrusion detection system," *Internal Journal of Applied Science and Technology*, vol. 2, no. 4, pp. 129–134, April 2012.

[2] M. Alfonseca, M. Cebrián, and A. Ortega, "A fitness function for computer generated music using genetic algorithms," *WSEAS Transactions on Information Science and Applications*, vol. 3, no. 3, pp. 518–525, March 2006.

[3] A. Auger and B. Doerr, *Theory of Randomized Search Heuristics: Foundations and Recent Developments*. World Scientific Publishing Co., Inc., 2011.

[4] M. J. Dinneen, Z. Y. Lin, and K. Wei, "An intelligent self-adjusting memetic algorithm for solving course scheduling problems," in *3rd International Conference on Information Science and Engineering (ICISE)*, vol. 1, 2011, pp. 140–144.

[5] M. J. Dinneen and K. Wei, "A (1+1) adaptive memetic algorithm for the maximum clique problem," in *Proceedings of Congress on Evolutionary Computation*, CEC'13, vol. 1. IEEE, June 2013, pp. 1626–1634.

[6] ——, "On the analysis of a (1+1) adaptive memetic algorithm," in *Proceedings of Memetic Computing, MC2013*. IEEE, 2013, pp. 24–31.

[7] E. Happ, D. Johannsen, C. Klein, and F. Neumann, "Rigorous analyses of fitness-proportional selection for optimizing linear functions," in *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, GECCO'08. ACM, 2008, pp. 953–960.

[8] T. Jansen, *Analyzing Evolutionary Algorithms: The Computer Science Perspective*, Natural Computing Series. Springer, 2013.

[9] T. Jansen, K. A. De Jong, and I. Wegener, "On the choice of the offspring population size in evolutionary algorithms," *Evolution Computation*, vol. 13, no. 4, pp. 413–440, 2005.

[10] T. Jansen and I. Wegener, "On the analysis of evolutionary algorithms—A proof that crossover really can help," *Algorithmica*, vol. 34, no. 1, pp. 47–66, 2002.

[11] ——, "On the analysis of a dynamic evolutionary algorithm," *Journal of Discrete Algorithms*, vol. 4, no. 1, pp. 181–199, 2006.

[12] T. Kötzing, D. Sudholt, and M. Theile, "How crossover helps in pseudo-boolean optimization," in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO'11, N. Krasnogor, Ed. ACM, 2011, pp. 989–996.

[13] F. Neri, C. Cotta, and P. Moscato, *Handbook of Memetic Algorithms*. Studies in Computational Intelligence, 2012, vol. 379.

[14] F. Neumann and C. Witt, *Bioinspired Computation in Combinatorial Optimization: Algorithms and Their Computational Complexity*, 1st ed. Springer-Verlag, 2010.

[15] P. S. Oliveto, J. He, and X. Yao, "Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results," *International Journal of Automation and Computing*, vol. 4, no. 3, pp. 281–293, 2007.

[16] E. Pitzer and M. Affenzeller, "A comprehensive survey on fitness landscape analysis," in *Recent Advances in Intelligent Engineering Systems*, vol. 378. Springer, 2012, pp. 161–190.

[17] C. Qian, Y. Yu, and Z.-H. Zhou, "An analysis on recombination in multi-objective evolutionary optimization," *Artificial Intelligence*, vol. 204, no. 0, pp. 99–119, 2013.

[18] P. Rohlfshagen and X. Yao, "Dynamic combinatorial optimization problems: A fitness landscape analysis," in *Metaheuristics for Dynamic Optimization*, vol. 433. Springer, 2013, pp. 79–97.

[19] F. A. Sadjadi, "Comparison of fitness scaling functions in genetic algorithms with applications to optical processing," in *Proceeding of SPIE*, vol. 5557, 2004, pp. 356–364.

[20] T. Storch, "How randomized search heuristics find maximum cliques in planar graphs," in *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, GECCO'06. ACM, 2006, pp. 567–574.

[21] ——, "Finding large cliques in sparse semi-random graphs by simple randomized search heuristics," *Theoretical Computer Science*, vol. 386, no. 12, pp. 114 – 131, 2007.

[22] D. Sudholt, "On the analysis of the (1+1) memetic algorithm," in *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, GECCO'06. ACM, 2006, pp. 493–500.

[23] ——, "Memetic algorithms with variable-depth search to overcome local optima," in *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, GECCO'08. ACM, 2008, pp. 787–794.

[24] ——, "The impact of parametrization in memetic evolutionary algorithms," *Theoretical Computer Science*, vol. 410, no. 26, pp. 2511–2528, 2009.

[25] D. Sudholt and C. Zarges, "Analysis of an iterated local search algorithm for vertex coloring," in *Proceedings of the 21st International Symposium on Algorithms and Computation (ISAAC)*, LNCS, vol. 6506. Springer, 2010, pp. 340–352.

[26] M. Szubert, W. Jaśkowski, P. Liskowski, and K. Krawiec, "Shaping fitness function for evolutionary learning of game strategies," in *Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference*, GECCO'13. ACM, 2013, pp. 1149–1156.

[27] K. Wei, "Runtime analysis on the (1+1) memetic algorithms," Ph.D. dissertation, University of Auckland, 2014, to appear.

[28] K. Wei and M. J. Dinneen, "Hybridizing the dynamic mutation approach with local searches to overcome local optima," in *Proceedings of Congress on Evolutionary Computation*, CEC'14. IEEE, July 2014, World Congress on Computaitonal Intelligence. To appear.

[29] ——, "Runtime analysis to compare best-improvement and first-improvement in memetic algorithms," in *Proceeding of the 16th Annual Conference on Genetic and Evolutionary Computation Conference*, GECCO'14. ACM, July 2014, To appear.

[30] C. Witt, "Runtime analysis of the (μ + 1) EA on simple pseudo-boolean functions," *Evolutionary Computation*, vol. 14, no. 1, pp. 65–86, 2006.