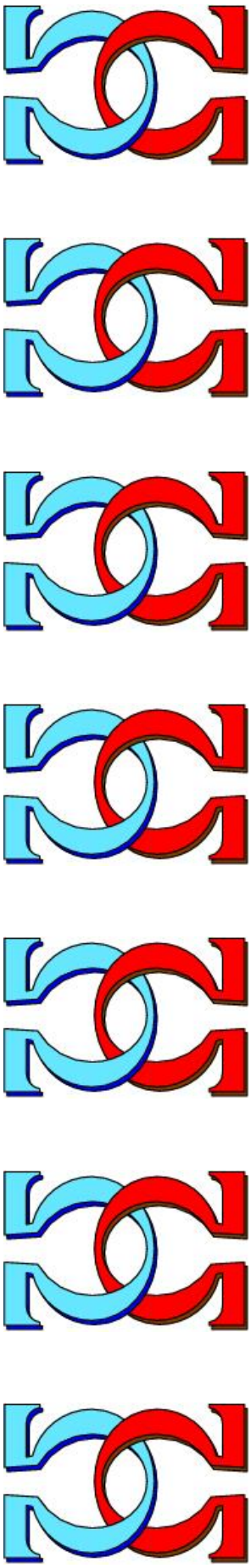


CDMTCS
Research
Report
Series

Comparing Two Local Searches in a (1+1) Restart Memetic Algorithm on the Clique Problem

Kuai Wei
Michael J. Dinneen

Department of Computer Science,
University of Auckland,
Auckland, New Zealand

CDMTCS-447
December 2013

Centre for Discrete Mathematics and
Theoretical Computer Science

Comparing Two Local Searches in a (1+1) Restart Memetic Algorithm on 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

In recent years, the advantage afforded by using multiple local searches in a Memetic Algorithm to solve one problem (a single fitness function), has been verified in many successful experiments. However, theoretical studies cannot explain why Memetic Algorithms with multiple local searches often outperform Memetic Algorithms with a single local search in these experiments. In this paper, we will formalize a (1+1) Restart Memetic Algorithm and two different local searches, and run them on a single fitness function to solve the Clique Problem. We then show that there are two families of graphs such that, for the first family of graphs, Memetic Algorithms with one local search drastically outperform Memetic Algorithms with the other local search, and vice versa for the second family of graphs. Furthermore, we propose a (1+1) Restart Memetic Algorithm with an Alternative Local Search, and show that the proposed algorithm is expected to solve the Clique Problem on both families of graphs efficiently. Lastly, we verify our theoretical results by experiments.

1 Introduction

Memetic Algorithms (MAs) are a wide class of randomized search heuristics that hybridize Evolutionary Algorithms (EAs) with local searches [17]. There are many examples in which MAs have successfully been used to solve various kinds of problems [2, 17]. This motivates the desire for a better understanding of MAs by using runtime analysis.

Since MAs combine EAs and local searches, in order to study the theory of MA, we need to begin with EAs first. Some of the earliest runtime analyses of EAs started with the basic (1+1) EA on, for example, simple pseudo-boolean functions [24], Onemax [24], Trap Functions [6], and plateaus of constant fitness [11]. Droste, Jansen and Wegener [7] summarized the basic (1+1) EA on which most theoretical studies of EAs are based. The term (1+1) represents that, a) the population size of parents and children are both one, and b) an elitist selection is used, i.e. the next generation will be chosen from

both parents and children. After analyzing the ability of EAs to solve the previously mentioned, artificially-created functions, many researches then went on to analyze the suitability of EAs for solving main-stream combinatorial optimization problems such as the Maximum Matching Problem [9], the Minimum Spanning Tree Problem [19], and the Partition Problem [30]. Meanwhile, very important progress was made in the analysis of population-based EAs, e.g. the $(\mu + 1)$ EA [31] and the $(1 + \lambda)$ EA [10]. Also, many researches have focused on showing that the recombination (also known as crossover) operation is essential in EAs, e.g. [12, 14, 23]. More details of runtime analysis on EAs can be found in [1, 13, 20, 21].

Shortly after the EAs were formalized and analyzed, Sudholt [26] proposed the first (1+1) MA in 2006, and compared it with the basic (1+1) EA on some artificially created functions. Because both MAs and EAs apply mutations and recombinations, and these operations have already been studied in EAs, the theoretical analyses of MAs have mainly focused on the impact of the local searches. For example, in 2008, Sudholt [28] analyzed the (1+1) MA with variable-depth search to overcome local optima on three binary combinatorial problems: Mincut, Knapsack, and Maxsat; similarly, in 2009, Sudholt [29] 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 [27] analyzed the interaction of two different mutations with local search for Vertex Coloring in 2010; Dinneen and Wei [4], in 2013, analyzed a dynamic mutation with two different local searches on some artificially created functions; and in the same year, Dinneen and Wei [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.

Theoretical studies of MAs have attracted many more researchers. The results of these studies explain the success of some experiments. For example, in 2011, Dinneen, Lin and Wei [3] proposed an MA with an adaptive mutation approach and gained success in experiments, and a runtime comparison between the dynamic mutation approach and the static mutation approach in MAs was studied in [4]. However, theoretical progress still lags behind the experimental approach. In particular, one trend in experimental research is to use multiple local searches in MAs to solve a single problem (one fitness function). In 2001, Krasnogor and Smith [15] did experiments and showed that even for a single problem class (TSP), the local search operator that gives the best results in an MA is entirely instance specific. Then, in 2005, Krasnogor and Smith [16] presented a taxonomy of memetic algorithms that provided a functional framework for using multiple local searches. This trend is also known as the multimeme algorithm [18], where each local search is a meme that is used adaptively or self-adaptively (see a survey in [22]). Furthermore, Coevolving Memetic Algorithms have been studied that coevolve the local search operators together with the mainstream population [25].

This trend of applying multiple local searches in MAs created new challenges in the runtime analysis of MAs. A few theoretical studies show that, on one artificially created function, MAs with one local search drastically outperform MAs with another local search, but the results are reversed when the same algorithms are applied to a

different function [4, 8, 26, 28]. However, to the best of our knowledge, no runtime analyses show that, for a single problem (one fitness function), the local search operator that gives the best results in an MA is entirely instance specific, which is the aim of this paper.

In this paper, we will formalize a (1+1) Restart Memetic Algorithm (RMA) and two local searches, the Random Complete Local Search (RCLS) and the Random Permutation Local Search (RPLS). Then we will show that, for the Clique Problem, there exists two families of graphs, such that:

1. For the first family of graphs (G_{RC}), the (1+1) RMA with the RCLS is expected to find the maximum clique within $O(n^2)$ fitness evaluations; but the (1+1) RMA with the RPLS is expected to take a super-polynomial number of fitness evaluations to find the maximum clique.

2. For the second family of graphs (G_{RP}), the (1+1) RMA with the RCLS is expected to take a super-polynomial number of fitness evaluations to find a maximum clique; but the (1+1) RMA with the RPLS is expected to find a maximum clique within $O(n^{1.5})$ fitness evaluations.

Lastly, we will propose a (1+1) Restart Memetic Algorithm with an Alternative Local Search (ALS). We then show that, for any graph of the family G_{RC}, the proposed algorithm is expected to find the maximum clique within $O(n^2)$ fitness evaluations; and for any graph of the family G_{RP}, the proposed algorithm is expected to find a maximum clique within $O(n^{2.5})$ fitness evaluations.

The paper is structured as follows. In Section 2, we will formalize the fitness function $f_{\texttt{OPL}}$ for the Clique Problem, then propose the (1+1) Restart Memetic Algorithm (RMA), and two local searches, the Random Complete Local Search (RCLS) and the Random Permutation Local Search (RPLS). Next, in Section 3, we will construct a family of graphs (G_{RC}), and show that the (1+1) RMA with the RCLS is expected to find the maximum clique on any graph of the family G_{RC} within a polynomial number of fitness evaluations, but the (1+1) RMA with the RPLS is expected to take a super-polynomial number of fitness evaluations to find the maximum clique. In Section 4, we will construct another family of graphs (G_{RP}) and show the opposite, i.e. the (1+1) RMA with the RPLS is expected to find a maximum clique on any graph of the family G_{RP} within a polynomial number of fitness evaluations, but the (1+1) RMA with the RCLS is expected to take a super-polynomial number of fitness evaluations to find a maximum clique. In Section 5, we will propose a (1+1) RMA with an Alternative Local Search, and show that the proposed algorithm is expected to find a maximum clique on any graph of both families (G_{RC} and G_{RP}) within a polynomial number of fitness evaluations. Our experiments will be given in Section 6. Finally, Section 7 concludes this paper and suggests possible avenues for future research.

2 Algorithm definitions

In this section we give basic definitions of our algorithms and we begin with the following standard notations that will be used throughout this paper.

1. $f(n) = \omega(g(n)) \leftrightarrow \forall k > 0,\ \exists n_0,\ \forall n > n_0,\ g(n) \cdot k < f(n)$.
2. $f(n) = \Omega(g(n)) \leftrightarrow \exists k > 0,\ \exists n_0,\ \forall n > n_0,\ g(n) \cdot k \leq f(n)$.
3. $f(n) = o(g(n)) \leftrightarrow \forall \epsilon > 0,\ \exists n_0,\ \forall n > n_0,\ f(n) < g(n) \cdot \epsilon$.
4. $f(n) = O(g(n)) \leftrightarrow \exists k > 0,\ \exists n_0,\ \forall n > n_0,\ f(n) \leq g(n) \cdot k$.
5. $f(n) = \Theta(g(n)) \leftrightarrow \exists k_1 > 0,\ \exists k_2 > 0,\ \exists n_0,\ \forall n > n_0,\ g(n) \cdot k_1 \leq f(n) \leq g(n) \cdot k_2$.
6. $\lim_{n\to\infty} (1 + 1/n)^n = e$.

2.1 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. In this section, we will formalize a fitness function $f_{\texttt{OPL}}$ for the Clique Problem.

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. $\{(v_i, v_j) \mid x_i = x_j = 1 \text{ and } i \neq j\} \subseteq E$.

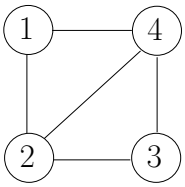
Definition 1. The fitness function $f_{\texttt{OPL}}$ (ONES, P[otential]V[ertices], LACKEDGES) is defined as follows:

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

where $\texttt{ONES}(x)$ is the number of ones in x; $\texttt{PV}(x)$ is the number of zeros in x such that, when each of these zeros is flipped individually, a larger clique is obtained. And $\texttt{LACKEDGES}(x)$ is the number of missing edges such that the subgraph becomes a clique.

Example 2. For a given graph G, displayed below, we have:

1. $f_{\texttt{OPL}}(1101) = 3$ because $x = (1101)$ is a clique that consists of vertices 1, 2 and 4.
2. $f_{\texttt{OPL}}(1111) = -1$ because we need to add one edge $(1, 3)$.
3. $f_{\texttt{OPL}}(1100) = 2.25$ because x represents a clique that consists of vertices 1 and 2, and because there is a potential vertex 4, such that by flipping the 4-th bit, a larger clique will be obtained.
4. $f_{\texttt{OPL}}(0101) = 2.5$ because x represents a clique that consists of vertices 2 and 4, and because there are two potential vertices 1 and 3, such that flipping either the first or the third bit will obtain a larger clique.



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

2.2 Algorithms to be analyzed

The algorithms we will analyze on the Clique Problem are the (1+1) Restart Memetic Algorithm (RMA) with two different local searches, the Random Complete Local Search (RCLS) and the Random Permutation Local Search (RPLS). Note, these algorithms all try to maximize the function $f = f_{\texttt{OPL}}$. The time complexity analysis in this paper looks at the number of evaluations of this fitness function $f_{\texttt{OPL}}$. The algorithms are stated as below:

Algorithm 3. (1+1) RMA.

1. Initialize the mutation probability $p = 1/n$, set $\texttt{gen} := 0$.
2. Choose $x = 0^n$.
3. $y := \texttt{Mutation}(x)$.
4. $z := \texttt{LocalSearch}(y)$.
5. If $f(z) \geq f(x)$ then $x := z$.
6. $\texttt{gen} := \texttt{gen} + 1$.
7. If $(\texttt{gen} \bmod \lambda) = 0$ then go to step 2.
8. Stop if any stopping criterion is met, otherwise, go to step 3.

Note, the $\texttt{Mutation}(x)$ flips each bit of x independently with probability $p = 1/n$. $\texttt{LocalSearch}(y)$ will be given below. Also 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.

2.3 Two local searches

As stated before, the local search in MAs can have many variations. In this paper, we will analyze a *Random Complete Local Search* (RCLS), which uses a steepest ascent pivot rule (also known as best-improvement), and a *Random Permutation Local Search* (RPLS), which uses a greedy pivot rule (also known as first-improvement).

Algorithm 4. Random Complete Local Search (RCLS). 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 $\texttt{BestNeighborSet}$.
4. Go to step 1.

Note, $\texttt{Hamming}(x, y)$ is the number of different bits between x and y. Also note that the RCLS will evaluate all n neighbors and then randomly select one of the best neighbors. Thus, according to the fitness function $f_{\texttt{OPL}}$ in Definition 1, we have:

1. If the bit string after the mutation does not represent a clique, the RCLS will keep flipping bits from ones to zeros until it finds a clique. Furthermore, the RCLS will take n fitness evaluations to flip one bit from one to zero. Thus, it will take at most n^2 fitness evaluations to find a clique. After the RCLS finds a clique, it will keep flipping bits from zeros to ones until it finds a local optimal clique, which again, will take at most n^2 fitness evaluations.
2. If the bit string after the mutation represents a clique, the RCLS will keep flipping bits from zeros to ones until it finds a local optimal clique, which takes at most n^2 fitness evaluations.

Therefore, the RCLS will stop on the fitness function $f_{\texttt{OPL}}$ within $2n^2$ fitness evaluations.

Unlike the RCLS, which evaluates all n neighbors before selecting one to flip, the RPLS randomly generates a permutation to represent the sequence of bits to search and executes the flipping as soon as the fitness evaluation improves. The algorithm is stated as below:

Algorithm 5. Randomized Permutation Local Search (RPLS). For a given string $x = (x_1, x_2, \cdots, x_n) \in \{0, 1\}^n$:

1. Generate a random permutation $\texttt{Per}$ of length n.
2. $i := 0$, $\texttt{NoImproveCount} := 0$.
3. $y := \texttt{flip}(x, \texttt{Per}[i])$.
4. If $f(y) > f(x)$ then $x := y$, $\texttt{NoImproveCount} := 0$.
5. $\texttt{NoImproveCount} := \texttt{NoImproveCount} + 1$.
6. $i := (i + 1) \bmod n$.
7. Stop if $\texttt{NoImproveCount} = n$. Otherwise, go to step 3.

Here $\texttt{flip}(x, \texttt{Per}[i])$ denotes that the $\texttt{Per}[i]$-th bit in x is flipped, and $\texttt{Per}[i]$ is the i-th number in the permutation $\texttt{Per}$.

Note that the RPLS moves to the first neighbor solution that improves the fitness, so we have:

1. If the bit string after the mutation does not represent a clique, the RPLS will take at most n fitness evaluations to check all n bits and flip every bit from one to zero if this flipping improves the fitness value. That is to say, the RPLS will find a clique within n fitness evaluations. Then the RPLS will take at most n fitness evaluations to flip bits from zeros to ones in order to find a local optimal clique.

2. If the bit string after the mutation represents a clique, the RPLS will take at most n fitness evaluations to find a local optimal clique.

Therefore, the RPLS will stop on the fitness function $f_{\texttt{OPL}}$ within $2n$ fitness evaluations.

Example 6. Suppose the bit string $x = (0, 0, 0, 0)$, and $\texttt{Per} = (3, 2, 1, 4)$. So the RPLS will first check a possible flipping for the third bit in x to get $x' = (0, 0, 1, 0)$. If $f(x') > f(x)$ then $x := x'$. This check sequence follows $\texttt{Per}$ in a cyclic fashion. That is, after checking the fourth bit in x, the PRLS will restart checking the third bit in x.

Therefore, the RCLS will take at most $2n^2$ fitness evaluations to find a local optima on the fitness function $f_{\texttt{OPL}}$; and the RPLS will take at most $2n$ fitness evaluations to find a local optima. In the rest of this paper, we will use (1+1) RMA_RCLS to denote that the (1+1) RMA is using the RCLS as the local search, and (1+1) RMA_RPLS to denote that the (1+1) RMA is using the RPLS as the local search.

3 A family of graphs on which the (1+1) RMA_RCLS outperforms the (1+1) RMA_RPLS

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

3.1 G_{RC} and its landscape

Definition 7. The graph $G_{RC}(t)$ 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 $k \equiv (l-1) \bmod t$.

Now 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 $v_{0,1}(x) = 0$.

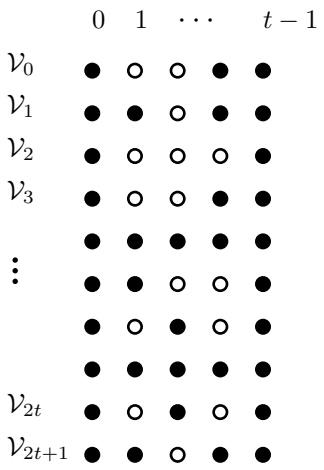


Figure 1: $(2t+2)$-by-t matrix for $G_{RC}(t)$

Figure 2 demonstrates the edge deleting rules for the graph $G_{RC}(t)$, 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 7; Case 2 in Figure 2 demonstrates all edges that need to be deleted, according to Rule 2 in Definition 7; 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 7.

Claim 8. *For the graph $G_{RC}(t)$, if the bit string x represents a clique, $V(x)$ can have 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 7, every vertex is connected to at most one vertex in the same column. □

Claim 9. *For the graph $G_{RC}(t)$, 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 7, 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, since $V(x)$ contains

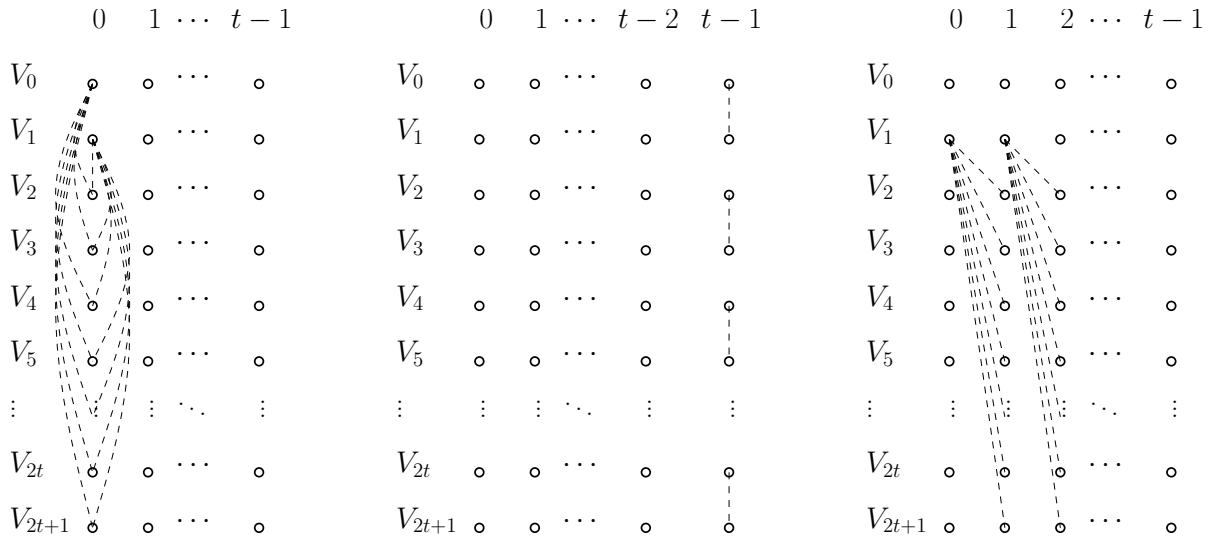


Figure 2: Illustrating Cases 1, 2 and 3 for Definition 7.

at least one vertex in each column, and there are t columns, $V(x)$ contains at least t vertices. □

Claim 10. *Any maximal clique can have two vertices in the last column of the graph $G_{RC}(t)$ if these two vertices are $v_{0,t-1}$ and $v_{1,t-1}$, otherwise, it can have one vertex.*

Proof. Due to Rule 1 and Rule 2 in Definition 7, $v_{0,t-1}$ and $v_{1,t-1}$ are connected, but any vertex in $\bigcup_{i=2}^{2t+1}\{v_{i,t-1}\}$ is not connected to all other vertices in the same column. □

Claim 11. *The only maximum clique of the graph $G_{RC}(t)$ is $V_{global} = V_0 \cup V_1$, which contains $2t$ vertices.*

Proof. Firstly, due to the edge rules in Definition 7, there exists a $2t$ clique of $V_0 \cup V_1$. Secondly, according to Claim 8, 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 8, $V(y)$ must contain two vertices in each column to be a $2t$ clique. Furthermore, due to Claim 10, $V(y)$ 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 7. 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. □

3.2 (1+1) RMA_RCLS on $G_{RC}(t)$

In this subsection, we say a bit string is on the *Path* if it represents a clique that only contains vertices in $V_0 \cup V_1$.

Theorem 12. *For any constant λ with $\lambda = \Theta(1)$, the (1+1) RMA_RCLS is expected to find the maximum clique of the graph $G_{RC}(t)$ within $O(n^2)$ fitness evaluations.*

Proof. Part 1. According to Algorithm 3, 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. Thus, within $\Theta(1)$ restarts, we expect to have a mutation that gets a bit string y, with $y = 0^n$. Furthermore, once the mutation produces the bit string y, the RCLS will first add a vertex from V_0 to the current clique, i.e. it is on the *Path*. This is because that:

1. Let y' be a bit string that adds a vertex $v_{0,k}$ $(0 \leq k \leq t-1)$ to y. Then y' represents a clique of size one, and $\texttt{PV}(y') = \texttt{PV}(y) - 2t$, i.e. there are $2t$ vertices that are in $\texttt{PV}(y)$ but not in $\texttt{PV}(y')$. In detail, these $2t$ vertices are $\bigcup_{i=2}^{2t+1}\{v_{i,k}\}$, due to Rule 1 in Definition 7.

2. Let y'' be a bit string that adds a vertex that is not in V_0 to y. Then y'' represents a clique of size one, and $\texttt{PV}(y'') \leq \texttt{PV}(y) - 2t - 1$, i.e. there are at least $2t+1$ vertices that are in $\texttt{PV}(y)$ but not in $\texttt{PV}(y'')$. In detail, these $2t+1$ vertices are a) $2t$ vertices in the same column due to Rule 1 in Definition 7; and b) one vertex in a different column due to Rule 3 in Definition 7.

Therefore, because we will restart the algorithm within every $\Theta(1)$ generations, and in each generation, the RCLS will stop on the fitness function within $2n^2$ fitness evaluations (see Algorithm 4), the (1+1) RMA_RCLS is expected to find a clique on the *Path* within $O(n^2)$ fitness evaluations.

Part 2. Now we show that, apart from the maximum clique bit string, for any bit string z on the *Path*, the best neighbors of z must be on the *Path* as well. We prove this by assuming that there exists a bit string z' that is one of the best neighbors of z and that is not on the *Path*, and then showing that there exists a bit string z'', that is also a neighbor of z, yet has a larger fitness value than z', thus z' does not exist.

Firstly, since z' is not on the *Path*, and is a better neighbor of z (z represents a clique on the *Path*), we know that z' must be a bit string which flips one bit of z from zero to one. Furthermore, let this flipped vertex be $v_{i,k}$, we know that $v_{i,k} \notin \{V_0 \cup V_1\}$, i.e. $2 \leq i \leq t+1$. Since z' is a better neighbor of z, z' is also a clique, and we have $\texttt{PV}(z') \leq \texttt{PV}(z) - 2t - 2$. This is because there are at least $2t+2$ vertices in $\texttt{PV}(z)$ but not in $\texttt{PV}(z')$. In detail, these $2t+2$ vertices are:

1. The flipped vertex $v_{i,k}$.

2. One vertex $v_{1,(k-1) \bmod t}$ due to Rule 3 in Definition 7.

3. $2t$ vertices of $v_{j,k}$ for all $0 \leq j \leq 2t+1$ and $\lfloor i/2 \rfloor \neq \lfloor j/2 \rfloor$ due to Rule 1 in Definition 7.

Secondly, let z'' be a neighbor of z that flips the vertex $v_{0,k}$, this means:

1. z'' must be a better neighbor of z. This is because a), the vertex $v_{0,k}$ must not be chosen by z, otherwise z' can not be a better neighbor of z; and secondly, the vertex $v_{0,k}$ is connected to all vertices in z since they are in $V_0 \cup V_1$, which is the maximum clique.

2. $\texttt{PV}(z'') = \texttt{PV}(z) - 2t - 1$. This is because there are exactly $2t+1$ vertices in $\texttt{PV}(z)$, but not in $\texttt{PV}(z'')$. In detail, these $2t+1$ vertices are a) the flipped vertex $v_{0,k}$, and b) $2t$ vertices of $v_{j,k}$ for all $2 \leq j \leq 2t+1$ due to Rule 1 in Definition 7.

Thus, we have a bit string z'' that is a better neighbor of z, and has a larger fitness value than z'. This conflicts with our assumption that z' is one of the best neighbors of z.

We have shown that, apart from the maximum clique bit string, for any bit string z on the *Path*, the best neighbors of z must be on the *Path* as well. Then we know that once the RCLS finds a bit string on the *Path*, it will check its neighbors and move to one of the best neighbors that are also on the *Path*. Thus, the RCLS will keep adding vertices from $V_0 \cup V_1$ until it finds the maximum clique of $V_0 \cup V_1$. This will take at most $2tn$ fitness evaluations (recall that $n = t(2t+2)$). Also, we expect to have a mutation to find a clique on the *Path* within $O(n^2)$ fitness evaluations (see Part 1). Thus the (1+1) RMA_RCLS is expected to find the maximum clique of the graph $G_{RC}(t)$ within $O(n^2 + 2tn)$ fitness evaluations, which is $O(n^2)$ due to $n = t(2t+2)$. □

3.3 (1+1) RMA_RPLS on $G_{RC}(t)$

We have shown that the (1+1) RMA_RCLS with $\lambda = \Theta(1)$ is expected to find the maximum clique of the graph $G_{RC}(t)$ efficiently. We now show that the (1+1) RMA_RPLS is not well suited for finding the maximum clique of the graph $G_{RC}(t)$ within a polynomial number of fitness evaluations, no matter what λ we choose for the restart condition.

Theorem 13. *For any static or dynamic variable λ, the probability of the (1+1) RMA_RPLS finding the maximum clique of the graph $G_{RC}(t)$ within a polynomial number of fitness evaluations is super-polynomially (to n) close to zero.*

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

Part 1. Let y be the first bit string found by the algorithm, after the start or each restart, that represents a clique. We claim that $V(y)$ is expected to have a constant number of vertices in $V_0 \cup V_1$ with a probability super-polynomially close to one.

According to Algorithm 3, since the algorithm starts or restarts with $x = 0^n$, and the mutation probability is $1/n$, the bit string after the mutation is expected to have a constant number of ones with a probability super-polynomially close to one. Then we have:

1. If the bit string after the mutation represents a clique, Part 1 is achieved.

2. If the bit string after the mutation is all zeros, i.e. 0^n, the RPLS will search its neighbors according to its random permutation array, and move to the first neighbor which represents a clique of size one. Hence, $V(y)$ contains a constant number of vertices in $V_0 \cup V_1$.

3. If the bit string after the mutation does not represent a clique, the function $-\texttt{LACKEDGES}$ in Definition 1 will guide the RPLS to flip some bits from ones to zeros in order to find a clique. Thus, the number of vertices in $V(y)$ is still a constant.

Part 2. Assume Part 1 has succeeded, i.e. the first clique found after start or each restart has a constant number of vertices in $V_0 \cup V_1$. Let z be the first bit string found by the algorithm, after start or each restart, that represents a local optimal clique. We claim that $V(z)$ is expected to have a constant number of vertices in $V_0 \cup V_1$ with a probability super-polynomially close to one.

Starting from the first bit string that represents a clique in Part 1, the RPLS will keep checking neighbors according to the sequence of its random permutation array, then moving to the first neighbor that has a better fitness value until no better neighbors exist. For any column k with $0 \leq k \leq t-1$, due to $|\bigcup_{j=2}^{2t+1}\{v_{j,k}\}| = t|\{v_{0,k}, v_{1,k}\}| = 2t$, the probability of the random permutation array guiding the RPLS to check a vertex in $\{v_{0,k}, v_{1,k}\}$ before checking any other vertices in the same column is $1/(t+1)$. Thus, if adding any vertex $v_{i,k}$ $(0 \leq i \leq 2t+1)$ from the column k to the current clique creates a larger clique, then the probability of the RPLS adding a vertex from $\bigcup_{j=2}^{2t+1}\{v_{j,k}\}$ is t times the probability of the RPLS adding a vertex from $\{v_{0,k}, v_{1,k}\}$.

Given above, and because there are t columns in the graph, if we for now ignore the consequences of Rule 3 in Definition 7, the first local optimal clique, that the RPLS will find, is expected to have only one column that includes vertices from $V_0 \cup V_1$. The probability of z having $\omega(1)$ columns that include vertices from $V_0 \cup V_1$ is super-polynomially close to zero.

Even though Rule 3 in Definition 7 might cause the RPLS to add more vertices from $V_0 \cup V_1$ to the current clique, we will show that it is expected to increase with no more than a constant number of vertices from $V_0 \cup V_1$ as well. Now we look at Rule 3 in Definition 7:

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 $k \equiv (l-1) \bmod t$.

Before the RPLS reaches the local optimal z, there may be some cases as follows: there exists a column k, $0 \leq k \leq t-1$, such that the current clique contains the vertex $v_{1,k}$, but does not contain any vertices in the next column $(k+1) \bmod t$. Thus adding any vertex from $\bigcup_{i=2}^{2t+1}\{v_{i,(k+1) \bmod t}\}$ will not improve the fitness, but adding any vertex from $\{v_{0,(k+1) \bmod t}, v_{1,(k+1) \bmod t}\}$ will improve the fitness. Consequently, the RPLS will choose to add a vertex from $\{v_{0,(k+1) \bmod t}, v_{1,(k+1) \bmod t}\}$ to the clique no matter the sequence of the random permutation. After the RPLS has added the vertex $v_{1,(k+1) \bmod t}$ to the clique, it might keep influencing the column $(k+2) \bmod t$ in which the RPLS can only add vertices from $\{v_{0,(k+2) \bmod t}, v_{1,(k+2) \bmod t}\}$, and then the column $(k+3) \bmod t$, and so on.

However, we claim that if a clique includes a vertex from V_1, but does not include any vertices in the next μ columns for any $\mu = \omega(1)$, then the probability of the RPLS adding vertices from $V_0 \cup V_1$ in the next μ columns is super-polynomially close to zero (we will show this in the next paragraph). Thus, since we only expect to have a constant number of vertices chosen from $V_0 \cup V_1$ in Part 1, Rule 3 in Definition 7 is only expected to result in an increase of a constant number of vertices from $V_0 \cup V_1$.

Let us assume that there exists a column k, and the RPLS will keep adding vertices from $V_0 \cup V_1$ in the columns $(k+1) \bmod t, (k+2) \bmod t, \cdots, (k+\mu) \bmod t$ for any μ with $\mu = \omega(1)$. We now show that this probability is super-polynomially close to zero. Note, for our assumption to succeed, the random permutation array needs to guide the RPLS to check the vertex $v_{1,(k+1) \bmod t}$ before checking all vertices from $\bigcup_{i=2}^{2t+1} \bigcup_{j=(k+2) \bmod t}^{(k+\mu) \bmod t} \{v_{i,j}\}$ (the probability of getting this exact permutation array is $1/2t(\mu-1)$). Otherwise, if the RPLS checks one vertex from $\bigcup_{i=2}^{2t+1} \bigcup_{j=(k+2) \bmod t}^{(k+\mu) \bmod t} \{v_{i,j}\}$ before checking the vertex $v_{1,(k+1) \bmod t}$, it will find out that adding this vertex to the clique will improve the fitness. Since the RPLS moves every time it finds a better neighbor, our assumption fails. Meanwhile, the permutation array needs to guide the RPLS to check the vertex $v_{1,(k+2) \bmod t}$ before checking all vertices from $\bigcup_{i=2}^{2t+1} \bigcup_{j=(k+3) \bmod t}^{(k+\mu) \bmod t} \{v_{i,j}\}$, and so on. In other words, for each variable l with $1 \leq l < \mu$, the permutation array needs to guide the RPLS to check the vertex $v_{1,(k+l) \bmod t}$ before checking all vertices in $\bigcup_{i=2}^{2t+1} \bigcup_{j=(k+l+1) \bmod t}^{(k+\mu) \bmod t} \{v_{i,j}\}$. The probability of this permutation array occurring is:

$$\frac{1}{2t(\mu-1)} \frac{1}{2t(\mu-2)} \cdots \frac{1}{2t} = \frac{1}{(2t)^{\mu-1}(\mu-1)!},$$

which is super-polynomially close to zero since $\mu = \omega(1)$ and $n = t(2t+2)$.

Thus, $V(z)$ contains a constant number of vertices in $V_0 \cup V_1$ with a probability super-polynomially close to one.

Part 3. Assume Part 2 has succeeded. After z is found, we claim that the (1+1) RMA_RPLS will either a) restart (the next restart will find another z that represents a clique with a constant number of vertices in $V_0 \cup V_1$ as well), or b) become trapped in a local optimal clique $V_{\texttt{local}}$ with $V_{\texttt{local}} \cap (V_0 \cup V_1) = \emptyset$ before it finds the global optimal clique of $V_{\texttt{global}} = V_0 \cup V_1$ with a probability super-polynomially close to one.

Since $V(z)$ has a constant number of vertices in $V_0 \cup V_1$ (assuming Part 2 has succeeded), and because any local optimal clique has at least one vertex in each column (see Claim 9), $V(z)$ must have $\Theta(t)$ vertices in $\bigcup_{i=2}^{2t+1} V_i$. Thus, within one mutation and the following RPLS, the ratio that the probability of the (1+1) RMA_RPLS finding any local optimal clique that has no vertices in $V_0 \cup V_1$ over the probability of the (1+1) RMA_RPLS finding the global optimal clique of $V_0 \cup V_1$ is super-polynomially large.

Now we look at the case of the (1+1) RMA_RPLS with multiple mutations, where each mutation is followed by one RPLS. We will also show that the (1+1) RMA_RPLS is expected to become trapped in a local optima that has no vertices in $V_0 \cup V_1$, before it finds the global optimal clique of $V_0 \cup V_1$.

We say that a mutation and the following RPLS is a *success* if it produces a new string z' in which, a) $V(z')$ has more (or less) vertices from $V_0 \cup V_1$ than the current $V(z)$, and b) z' is accepted as the new current string. In other words, z' is a *success* if

$|V(z) \cap (V_0 \cup V_1)| \neq |V(z') \cap (V_0 \cup V_1)|$ and $f_{\texttt{OPL}}(z') \geq f_{\texttt{OPL}}(z)$. In each *success*, if there are α consecutive columns, where in each column, $V(z)$ contains vertices in $\bigcup_{i=2}^{2t+1} V_i$, but $V(z')$ contains vertices in $V_0 \cup V_1$, we call it an α-column *positive change*, since it is approaching the global optimal solution. On the other hand, if there are α consecutive columns, where in each column, $V(z)$ contains vertices in $V_0 \cup V_1$, but $V(z')$ contains vertices in $\bigcup_{i=2}^{2t+1} V_i$, we call it an α-column *negative change*, since it is approaching the local optimal solutions with no vertices in $V_0 \cup V_1$. Overall, a *success* might include some *positive changes* and/or some *negative changes*. For example, Figure 3 is a success on the graph $G_{RC}(5)$ which has a 1-column *negative change* in the column 1 and a 1-column *positive change* in the column 4.

Recall that $V(z)$ has a constant number of vertices in $V_0 \cup V_1$. Thus, to find the global optimal clique, we need the total number of columns that undergo *positive changes* to be $\Theta(t)$ larger than the total number of columns that undergo *negative changes*. But to find a local optimal clique with no vertices in $V_0 \cup V_1$, we only need the total number of columns that undergo *negative changes* to be $\Theta(1)$ larger than the total number of columns that undergo *positive changes*. Note that the probability of a *success* changing $\omega(1)$ columns is super-polynomially close to zero since the probability of one mutation changing $\omega(1)$ bits is super-polynomially close to zero. Thus, the algorithm needs $\Omega(t)$ *successes* to find the global optimal clique. Furthermore, we claim that after a number of *successes*, the total number of columns that undergo *negative changes* is expected to be $\omega(1)$ times the total number of columns that undergo *positive changes* (we will show this later). This implies that within $\omega(1)$ and $o(t)$ *successes*, the algorithm a) has not found the global optimal clique, and b) has found a local optimal clique with no vertices in $V_1 \cup V_2$. Thus, with overwhelming probability, the (1+1) RMA_RPLS will become trapped in a local optimal clique with no vertices in $V_0 \cup V_1$ before finding the global optimal clique.

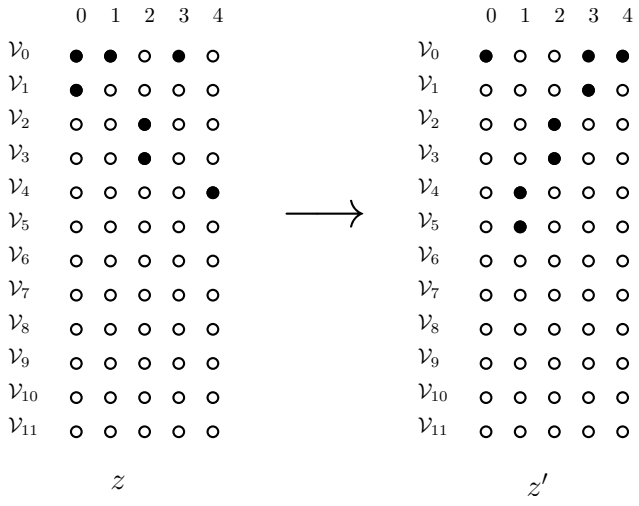


Figure 3: A success for the graph $G_{RC}(5)$

Now we show that after a number of *successes*, the total number of columns that

undergo *negative changes* is expected to be $\omega(1)$ times the total number of columns that undergo *positive changes*.

Note that, for a column k, if the current clique contains vertices from $\bigcup_{i=2}^{2t+1}\{v_{i,k}\}$ in this column, there is only one possible way to achieve a 1-column *positive change*, i.e. the mutation and the RPLS needs to remove the two vertices that the current clique has chosen in the column k, and add the vertices from $\{v_{0,k}, v_{1,k}\}$ to the clique. Then, after achieving this 1-column *positive change* for the column k, there are t possible ways to get a 1-column *negative change*, i.e. the mutation and the RPLS can remove the two vertices from $\{v_{0,k}, v_{1,k}\}$, and add another two vertices from either $\{v_{2,k}, v_{3,k}\}$, $\{v_{4,k}, v_{5,k}\}$, $\cdots$, $\{v_{2t-2,k}, v_{2t-1,k}\}$, or $\{v_{2t,k}, v_{2t+1,k}\}$ to the clique. Furthermore, the probability of achieving each of the possible ways of getting the 1-column *negative change* is the same as the probability of achieving the 1-column *positive change*. This is because they all need to change the same number of vertices. Thus, after the algorithm has gained a 1-column *positive change*, the probability of the algorithm gaining a corresponding 1-column *negative change*, i.e. the clique will now choose vertices from $\bigcup_{i=2}^{2t+1} V_i$ in this column, is t times the probability of the algorithm gaining the 1-column *positive change*.

Similarly, if the algorithm has gained an α-column *positive change* ($\alpha \geq 1$), there are t^α ways of getting an α-column *negative change*. That is to say, the probability of the algorithm getting a corresponding α-column *negative change* is t^α times the probability of the algorithm gaining the α-column *positive change*.

Equivalently, if the current clique has gained an α-column *negative change*, the probability of the algorithm gaining a corresponding α-column *positive change* is $1/t^\alpha$ times the probability of gaining the α-column *negative change*.

Overall, after a number of *successes* (each *success* includes some *positive changes* and/or *negative changes*), the total number of columns that undergo *negative changes* is expected to be $\omega(1)$ times the total number of columns that undergo *positive changes*.

Part 4. Assume Part 3 has succeeded, i.e. the (1+1) RMA_RPLS has been trapped in a local optimal solution x with $V(x) \cap (V_0 \cup V_1) = \emptyset$. We claim that, before the next restart, the probability of the (1+1) RMA_RPLS finding the global optimal clique within a polynomial number of fitness evaluations is super-polynomially close to zero.

Since the current clique has no vertices in $V_0 \cup V_1$, to find the global optimal clique, we need the total number of columns that undergo *positive changes* to be t larger than the total number of columns that undergo *negative changes*. However, as shown in Part 3, after a number of *successes*, the total number of columns that undergo *negative changes* is expected to be $\omega(1)$ times the total number of columns that undergo *positive changes*. Therefore, the probability of the (1+1) RMA_RPLS finding the global optimal clique within a polynomial number of fitness evaluations is super-polynomially close to zero. □

4 A family of graphs on which the (1+1) RMA_RPLS outperforms the (1+1) RMA_RCLS

In this section, we will construct a family of graphs G_{RP} and show that the (1+1) RMA_RPLS is expected to find a maximum clique within a polynomial number of fitness evaluations, while the (1+1) RMA_RCLS is expected to find a maximum clique within a super-polynomial number of fitness evaluations.

4.1 G_{RP} and its landscape

Definition 14. The graph $G_{RP}(t)$ has the same number of vertices as the graph $G_{RC}(t)$, 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 $V_0, V_1, \cdots, V_{2t+1}$ 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 $0 \leq k \leq t-1$, $0 \leq i \leq 2t+1$, $0 \leq j \leq 2t+1$ 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 $0 \leq l \leq t-1$.

Now 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 4. 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 4) to denote that $v_{i,k} \in V(x)$, and $v_{i,k}(x) = 0$ (a hollow circle in Figure 4) otherwise. In Figure 4, $v_{0,0}(x) = 1$ and $v_{0,1}(x) = 0$.

Figure 5 demonstrates the edge deleting rules for the graph $G_{RP}(t)$, in which dashed lines denote that the edges are deleted. In detail, Case 1 in Figure 5 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 14; Case 3 in Figure 5 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 14; Case 4 in Figure 5 is an example of deleting edges that are connected to the vertices $v_{1,0}$ and $v_{1,1}$, according to

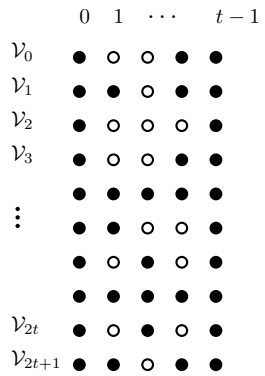


Figure 4: $(2t+2)$-by-t matrix for $G_{RP}(t)$

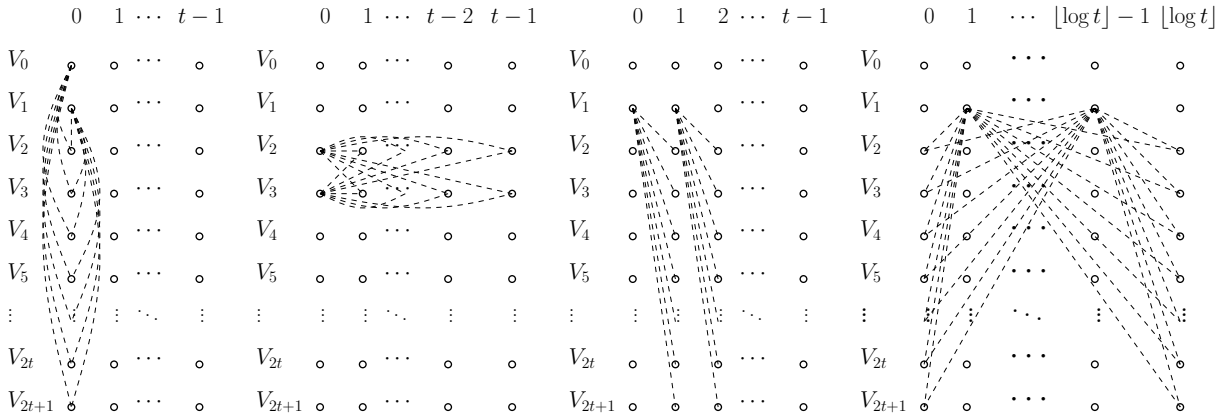


Figure 5: Illustrating Cases 1, 3, 4 and 5 for Definiton 14.

Rule 4 in Definition 14; and Case 5 in Figure 5 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 14.

Claim 15. *For the graph $G_{RP}(t)$, if the bit string x represents a clique, $V(x)$ can have 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 14, every vertex is connected to at most one vertex in the same column. □

Claim 16. *For the graph $G_{RP}(t)$, 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 14, 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 17. *In the last column of the graph $G_{RP}(t)$, a maximal clique can have either a) one vertex from $\{v_{0,t-1}, v_{1,t-1}\}$, or b) two vertices in the last column.*

Proof. Due to Rule 1 and Rule 2 in Definition 14, $v_{0,t-1}$ and $v_{1,t-1}$ are disconnected, thus any maximal clique can only contain one vertex from $\{v_{0,t-1}, v_{1,t-1}\}$. Meanwhile, any vertex $v_{i,t-1}$, $2 \leq i \leq 2t+1$, is connected to the vertex $v_{j,t-1}$ with $i \neq j$ and $\lfloor i/2 \rfloor = \lfloor j/2 \rfloor$. Also, $\forall v \in V, v \neq v_{i,t-1}$ and $v \neq v_{j,t-1} : (v, v_{i,t-1}) \in E \Leftrightarrow (v, v_{j,t-1}) \in E$. Furthermore, any maximal clique must contain at least one vertex in each column due to Claim 16. Thus, if any maximal clique does not contain any vertices from $\{v_{0,t-1}, v_{1,t-1}\}$, it will choose a pair of vertices in the last column. □

Claim 18. *For any bit string x that represents a maximal clique of the graph $G_{RP}(t)$, if $V(x)$ does not contain any vertices in $V_0 \cup V_1$, then it must contain $2t$ vertices and is a maximum clique. Otherwise, $V(x)$ must contain less than $2t$ vertices.*

Proof. Firstly, due to Claim 15, 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 14, 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 Claim 17. 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 14. In this case, the clique size of x will be less than $2t$.

□

4.2 (1+1) RMA_RPLS on $G_{RP}(t)$

Theorem 19. *For any constant λ with $\lambda = \Theta(1)$, the (1+1) RMA_RPLS is expected to find a maximum clique of the graph $G_{RP}(t)$ within $O(n^{1.5})$ fitness evaluations.*

Proof. Part 1. According to Algorithm 3, 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. Thus, 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 RPLS will take at most $2n$ fitness evaluations in each generation (see Algorithm 5), we expect to have a mutation that does not flip any bits within $O(n)$ fitness evaluations.

Part 2. Proceeding from Part 1, i.e. the mutation does not flip any bits, now we look at the probability of the RPLS finding a local optimal clique that does not contain any vertices in $V_0 \cup V_1$. This local optimal clique is also a maximum clique, according to Claim 18.

Note that the RPLS will keep checking neighbors according to the sequence of its random permutation array, then moving to the first neighbor that has a better fitness value, until no better neighbors exist. Also, according to Claim 16, when the RPLS finds a local optimal clique, this clique must contain at least one vertex in each column. Therefore, if we look at each column, and focus on the first vertex in the column that the RPLS will add to the current clique, there must exist a sequence of columns $C = (c_1, c_2, \ldots, c_t)$, such that the RPLS will first add one vertex from the column c_1 to the current clique, then add another vertex from the column c_2 to the current clique, and so on. Note that in each column k, $|\bigcup_{i=2}^{2t+1}\{v_{i,k}\}| = t|\{v_{0,k}, v_{1,k}\}| = 2t$. Hence, in the first column (c_1), the probability of the random permutation array guiding the RPLS to add a vertex from $V_0 \cup V_1$ to the current clique is $1/(t+1)$. That is to say, in the column c_1, the probability of the random permutation array guiding the RPLS to add a vertex from $\bigcup_{i=2}^{2t+1} V_1$ to the current clique is $1 - 1/(t+1) = t/(t+1)$. Also note that, if the RPLS has added any vertex from $\bigcup_{i=2}^{2t+1} V_i$ in the column c_1 to the clique, then, in the column c_2, there are two vertices from $\bigcup_{i=2}^{2t+1} V_i$ that are disconnected with the current clique, due to Rule 3 in Definition 14. Therefore, in the column c_2, the number of vertices from $\bigcup_{i=2}^{2t+1} V_i$ that can be added to the current clique to get a larger clique will be decreased to $2t-2$. Meanwhile, the number of vertices from $V_0 \cup V_1$ in the column c_2 that can be added to the current clique to get a larger clique will be at most two (this value can be less than two because that the Rules 4 and 5 in Definition 14 might prevent the addition of the vertex from V_1 in the column c_2 as a means of improving the clique size). Thus, the probability of the random permutation array guiding the RPLS to add a vertex from $\bigcup_{i=2}^{2t+1} V_i$ in the column c_2 to the current clique is at least $(t-1)/t$. That is to say, in the column c_i, if the current clique does not contain any vertices in $V_0 \cup V_1$, i.e. there are $i-1$ columns from which the current clique contains vertices in $\bigcup_{i=2}^{2t+1} V_i$, then, the probability of the random permutation array guiding the RPLS to add a vertex from $\bigcup_{i=2}^{2t+1} V_i$ in the column c_i to the current clique is at least $(t-i+1)/(t-i+2)$. Overall, the probability of the random permutation array guiding the RPLS to find a local optimal clique that does not contain any vertices in $V_0 \cup V_1$ is at least $\frac{t}{t+1}\frac{t-1}{t}\cdots\frac{1}{2} = 1/(t+1)$.

Therefore, if we have $O(t)$ mutations that all produce the bit string 0^n, the RPLS is expected to find a local optima that does not contain any vertices in $V_0 \cup V_1$. This local optima is also a maximum clique due to Claim 18. Also, according to Part 1, we expect to have a mutation that produces the bit string 0^n within $O(n)$ fitness evaluations. Hence, the (1+1) RMA_RPLS is expected to find a maximum clique within $O(tn)$ fitness evaluations, which is $O(n^{1.5})$ since $n = t(2t+2)$. □

4.3 (1+1) RMA_RCLS on $G_{RP}(t)$

We have shown that the (1+1) RMA_RPLS with $\lambda = \Theta(1)$ is expected to find a maximum clique of the graph $G_{RP}(t)$ within a polynomial number of fitness evaluations. We now show that the (1+1) RMA_RCLS is not well suited for finding a maximum clique of the graph $G_{RP}(t)$ within a polynomial number of fitness evaluations no matter what λ we choose for the restart condition.

Theorem 20. *For any static or dynamic variable λ, the probability of the (1+1) RMA_RCLS finding a maximum clique of the graph $G_{RP}(t)$ within a polynomial number of fitness evaluations is super-polynomially close to zero.*

Proof. We prove this by showing that, from the start or from each restart, the (1+1) RMA_RCLS 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. Let y be the first bit string found by the algorithm after the start or each restart that represents a clique. We claim that $V(y)$ is expected to have a constant number of vertices in $\bigcup_{i=2}^{2t+1} V_i$.

According to Algorithm 3, since the algorithm starts with $x = 0^n$, the bit string after mutation is expected to have a constant number of bits flipped with a probability super-polynomially close to one. If the bit string after the mutation represents a clique, then Part 1 is achieved. Or if the bit string after the mutation is still 0^n, the RCLS will flip one bit from zero to one in order to get a clique, then Part 1 is achieved. Or the bit string after the mutation has some ones, but it does not represent a clique, then the RCLS will search all neighbors and remove some ones to zeros until the bit string represents a clique. In this case, $V(y)$ still contains a constant number of vertices in $\bigcup_{i=2}^{2t+1} V_i$.

Part 2. Assume Part 1 has succeeded, let z be the first bit string found by the algorithm after the start or each restart that represents a local optimal clique. We claim that z is expected to have a constant number of vertices in $\bigcup_{i=2}^{2t+1} V_i$.

After y is found, the RCLS will keep checking the current string's neighbors, then moving to one of the best neighbors until it reaches a local optimal solution, i.e. z. Since $V(y)$ only contains a constant number of vertices, there are still $\Theta(t)$ columns from which y has not chosen any vertices. Let us call them *un-chosen columns*. Furthermore, among these *un-chosen columns*, we call a column a *negative column* if $V(z)$ contains a vertex from $V_0 \cup V_1$ in this column due to the fact that it is moving further away from all global optimal cliques. Alternatively, if $V(z)$ contains a vertex from $\bigcup_{i=2}^{2t+1} V_i$ in an *un-chosen column*, we call it a *positive column*. We claim that, when the RCLS reaches z, there is at most one *positive column* while all other *un-chosen columns* will be *negative columns* (we will show this in the next paragraph). And for each *negative column*, the RCLS will not add any vertices from $\bigcup_{i=2}^{2t+1} V_i$ in this column to the clique due to Rule 1 in Definition 14. Also, the *positive column* can have at most two vertices from $\bigcup_{i=2}^{2t+1} V_i$, due to Claim 15. Thus, $V(z)$ is expected to contain a constant number of vertices in $\bigcup_{i=2}^{2t+1} V_i$.

Now we show that when the RCLS reaches z, there is at most one *positive column* while all other columns will be *negative columns*. We assume that there is more than one *positive column*, then we can select any two *positive columns* of column k and l ($k \neq l$), and show a contradiction. First, let $v_{i,k}$ be the first vertex in the column k that the RCLS will add to the clique, and let $v_{j,l}$ be the first vertex in the column l that the RCLS will add to the clique. Since the column k and the column l are both *positive columns*, we know $2 \leq i \leq 2t+1$ and $2 \leq j \leq 2t+1$. Furthermore, let us assume that the RCLS will add the vertex $v_{i,k}$ before adding the vertex $v_{j,l}$. Let x be the bit string before the RCLS adds the vertex $v_{i,k}$, and x' be the bit string after the RCLS adds the vertex $v_{i,k}$ to x. Also, let s be the number of vertices in the column k that are in $\texttt{PV}(x)$, i.e. $s = |\texttt{PV}(x) \cap \left(\bigcup_{j=0}^{2t+1}\{v_{j,k}\}\right)|$. Then we say $\texttt{PV}(x') \leq \texttt{PV}(x) - s - 1$ due to the fact that there are at least $s+1$ vertices in $\texttt{PV}(x)$, but not in $\texttt{PV}(x')$. In detail, these $s+1$ vertices are:

1. $s-1$ vertices in the column k. This is because the vertex $v_{i,k}$ is connected to only one vertex in the same column, thus all other vertices would not be in $\texttt{PV}(x')$. Note the vertex $v_{i,k}$ is in $\texttt{PV}(x)$, but not in $\texttt{PV}(x')$.

2. 2 vertices in the column l. In detail, these two vertices are $v_{i,l}$ and $v_{i',l}$, with $\lfloor i/2 \rfloor = \lfloor i'/2 \rfloor$ and $i \neq i'$. This is because a) the vertices $v_{i,l}$ and $v_{i',l}$ are not connected to the vertex $v_{i,k}$, due to Rule 3 in Definition 14, hence, they are not in $\texttt{PV}(x')$; and b) $v_{i,l}$ and $v_{i',l}$ must be in $\texttt{PV}(x)$. We prove this by assuming that $v_{i,l}$ or $v_{i',l}$ is not in $\texttt{PV}(x)$, and then show a contradiction. If $v_{i,l}$ or $v_{i',l}$ is not in $\texttt{PV}(x)$ because of Rule 3 in Definition 14, then the vertex $v_{i,k}$ should not be in $\texttt{PV}(x)$ as well. Hence, the RCLS will not add the vertex $v_{i,k}$ to x. Or if $v_{i,l}$ or $v_{i',l}$ is not in $\texttt{PV}(x)$ because of Rule 4 or Rule 5 in Definition 14, then the column l can not be a *positive column*. Therefore, the vertices $v_{i,l}$ and $v_{i',l}$ must be in $\texttt{PV}(x)$.

Meanwhile, there exists another solution x'', of which $V(x'') = V(x) \cup \{v_{0,k}\}$. Thus, x'' is a neighbor of x. Furthermore, we claim that x'' has a larger fitness value than x' (we will show this later). Therefore, the RCLS will not choose to move to x', which contradicts our assumption. (Recall that the RCLS only moves to one of the best neighbors of the current solution).

Firstly, x'' must represent a clique. This is because $V(x)$ does not contain any vertices in the column k, and $v_{0,k}$ is connected to all vertices in other columns. Therefore, adding the vertex $v_{0,k}$ into x still represents a clique. Thus $\texttt{ONES}(x'') = \texttt{ONES}(x')$.

Secondly, according to the edge rules in Definition 14, the vertex $v_{0,k}$ is connected to all vertices that are not in the column k. Thus, for all vertices in other columns, if they are in $\texttt{PV}(x)$, then they will still be in $\texttt{PV}(x'')$ as well. This means that $\texttt{PV}(x'') \geq \texttt{PV}(x) - s > \texttt{PV}(x')$. Hence, x'' has a larger fitness value than x'.

Part 3. Assume Part 2 has succeeded, we claim that before the next restart, the probability of the algorithm finding a maximum clique within a polynomial number of fitness evaluations is super-polynomially close to zero.

Firstly, we look at the probability of the algorithm using one mutation and the RCLS to find a global optimal clique. Since $V(z)$ only contains a constant number of vertices in

$\bigcup_{i=2}^{2t+1} V_i$, it must contain $\Theta(t)$ vertices in $V_0 \cup V_1$ due to Claim 16. Thus, the probability of the algorithm finding a global optimal clique with one mutation and the following RCLS is super-polynomially close to zero.

Secondly, we look at the probability of the algorithm finding a maximum clique within a polynomial number of fitness evaluations, by using multiple mutations, with each followed by one RCLS. Recall Rule 5 in Definition 14, for a section of $\lfloor \log t \rfloor$ consecutive columns from the column k_1 to the column k_2 with $0 \leq k_1 \leq t-1$, $0 \leq k_2 \leq t-1$, $k_2 - k_1 = \lfloor \log t \rfloor$ and $k_1 \bmod \lfloor \log t \rfloor = 0$, every vertex from $\bigcup_{k=k_1}^{k_2} \{v_{1,k}\}$ is disconnected with every vertex from $\bigcup_{i=2}^{2t+1} \{v_{i,k_1}, v_{i,k_2}\}$. For each of the above sections of $\lfloor \log t \rfloor$ consecutive columns, if the current local optimal clique only contains $2\lfloor \log t \rfloor$ vertices from $V_0 \cup V_1$ in this section, we call it a *hard-to-escape section*. This is because if we want to remove some vertices that are in this section of columns and also in $V_0 \cup V_1$ from the current clique, and add the same number of vertices that are in this section of columns and also in $\bigcup_{i=2}^{2t+1} V_i$ to the clique, then the only way is to remove all $2\lfloor \log t \rfloor$ vertices that are in $V_0 \cup V_1$, and add another $2\lfloor \log t \rfloor$ vertices that are in $\bigcup_{i=2}^{2t+1} V_i$ and that can be reached with one mutation and the following RCLS. The probability of achieving this is super-polynomially close to zero. Alternatively, if one mutation and the following RCLS only removes a constant number of vertices in this section from $V_0 \cup V_1$, and adds vertices from $\bigcup_{i=2}^{2t+1} V_i$ to the clique, then the clique size will be decreased. Hence, the algorithm will not accept this solution, and we cannot rely on using multiple mutations and RCLSs to avoid the current clique from containing vertices in $V_0 \cup V_1$ in this section.

Now recall that $V(z)$ contains a constant number of vertices from $\bigcup_{i=2}^{2t+1} V_i$, therefore, it is expected to have $\omega(1)$ number of the above *hard-to-escape sections*. Thus, the probability of avoiding the current clique from containing vertices in $V_0 \cup V_1$ is super-polynomially close to zero, since the algorithm needs to change $\Theta(\log t)$ vertices with one mutation and the following RCLS. □

5 (1+1) RMA with an Alternative Local Search

In this section, we will propose a (1+1) Restart Memetic Algorithm (RMA) with an Alternative Local Search, which is denoted as (1+1) RMA_ALS. Then we will show that the proposed algorithm is expected to find a maximum clique of both families of graphs, G_{RC} and G_{RP}, within a polynomial number of fitness evaluations. The algorithm is stated as below:

Algorithm 21. (1+1) RMA_ALS.

1. Initialize the mutation probability $p = 1/n$,
   set `gen` := 0, the local search flag `lsf` := 1.
2. Choose $x := 0^n$.
3. $y :=$ `Mutation`(x).
4. If `lsf` $= 1$ then $z :=$ `RCLS`(y) else $z :=$ `RPLS`(y).
5. If $f(z) \geq f(x)$ then $x := z$.

6. $\texttt{gen} := \texttt{gen} + 1$.

7. If $(\texttt{gen} \bmod \lambda) = 0$ then $\texttt{lsf} := 1 - \texttt{lsf}$, go to step 2.

8. Stop if any stopping criterion is met, otherwise, go to step 3.

Note that the (1+1) RMA_ALS will first use the RCLS as the local search until the next restart, then it will switch to the RPLS until the second restart, and so on.

Corollary 22. *For any constant λ with $\lambda = \Theta(1)$, the (1+1) RMA_ALS is expected to find the maximum clique of the graph $G_{RC}(t)$ within $O(n^2)$ fitness evaluations.*

Proof. Recall that the RCLS will take at most $2n^2$ fitness evaluations in each generation (see Algorithm 4), and the RPLS will take at most $2n$ fitness evaluations in each generation (see Algorithm 5). Also recall Theorem 12, the (1+1) RMA_RCLS is expected to find the maximum clique on the family G_{RC} within $\Theta(1)$ generations. Therefore, because the (1+1) RMA_ALS will alternatively apply the RCLS and the RPLS in every λ generations, the algorithm is expected to find the maximum clique on the family G_{RC} within $O(n^2)$ fitness evaluations. □

Corollary 23. *For any constant λ with $\lambda = \Theta(1)$, the (1+1) RMA_ALS is expected to find a maximum clique of the graph $G_{RP}(t)$ within $O(n^{2.5})$ fitness evaluations.*

Proof. Recall that the RCLS will take at most $2n^2$ fitness evaluations in each generation (see Algorithm 4), and the RPLS will take at most $2n$ fitness evaluations in each generation (see Algorithm 5). Also recall Theorem 19, the (1+1) RMA_RPLS is expected to find a maximum clique on any graph of the family G_{RP} within $O(t)$ generations. Therefore, because the (1+1) RMA_ALS will alternatively apply the RCLS and the RPLS in every λ generations, the algorithm is expected to find a maximum clique on any graph of the family G_{RP} within $O(tn^2)$ fitness evaluations, which is $O(n^{2.5})$ since $n = t(2t + 2)$. □

6 Experiments

We now finally examine the experimental results regarding the behavior of the (1+1) RMA with different local searches on the families of graphs G_{RC} and G_{RP}. We are particularly interested in the number of fitness evaluations that each algorithm needs to find a maximum clique as the order n of the graph grows. We run each of our algorithms on each graph 50 times. Since we have shown that some algorithms are expected to take a super-polynomial number of fitness evaluations to find a maximum clique on certain graphs, we stop the algorithm and mark it as a fail if these 50 runs do not stop within 100 hours.

Firstly, we tested the (1+1) RMA_RCLS, the (1+1) RMA_RPLS and the (1+1) RMA_ALS on the family G_{RC}. Recall we have shown that both the (1+1) RMA_RCLS and the (1+1) RMA_ALS are expected to find the maximum clique on any graph of the family G_{RC} within a polynomial number of fitness evaluations with $\lambda = \Theta(1)$ (see

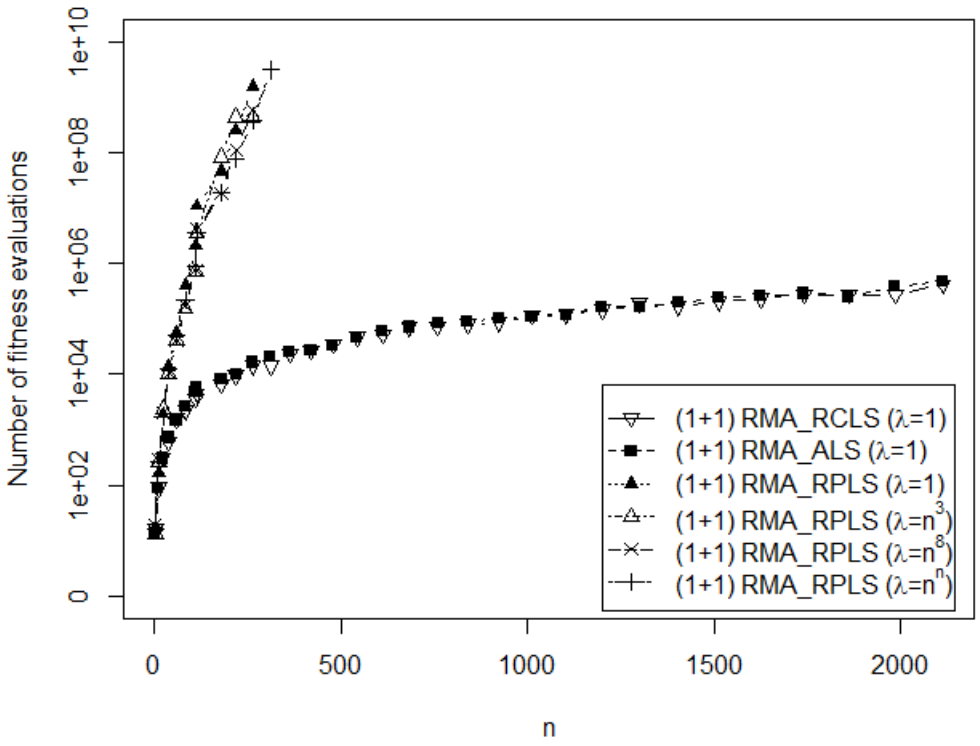


Figure 6: A comparison of the average number of fitness evaluations (on a log-scale) as n grows for the (1+1) RMA_RCLS ($\lambda = 1$), the (1+1) RMA_ALS ($\lambda = 1$), and the (1+1) RMA_RPLSs ($\lambda \in \{1, n^3, n^8, n^n\}$) on the $G_{RC}(t)$, $n = t(2t + 2)$.

Theorem 12 and Corollary 22). Therefore, we tested these two algorithms with $\lambda = 1$. Also recall that, for any static or dynamic λ, the (1+1) RMA_RPLS is expected to take a super-polynomial number of fitness evaluations to find the maximum clique on any graph of the family G_{RC} (see Theorem 13). Therefore, we tested the (1+1) RMA_RPLS on the family G_{RC} with four different λs, $\lambda \in \{1, n^3, n^8, n^n\}$, respectively. The results of the above tested algorithms on the graphs of the family G_{RC} are shown in Figure 8. A comparison of the average number of fitness evaluations (on a log-scale) for all algorithms tested on the family G_{RC} is shown in Figure 6.

Secondly, we tested the (1+1) RMA_RCLS, the (1+1) RMA_RPLS and the (1+1) RMA_ALS on the family G_{RP}. Recall we have shown that both the (1+1) RMA_RPLS and the (1+1) RMA_ALS are expected to find a maximum clique on any graph of the family G_{RP} within a polynomial number of fitness evaluations with $\lambda = \Theta(1)$ (see Theorem 19 and Corollary 23). Therefore, we tested these two algorithms with $\lambda = 1$. Also recall that, for any static or dynamic λ, the (1+1) RMA_RCLS is expected to take a super-polynomial number of fitness evaluations to find a maximum clique on any graph of the family G_{RP} (see Theorem 20). Therefore, we tested the (1+1) RMA_RCLS on the

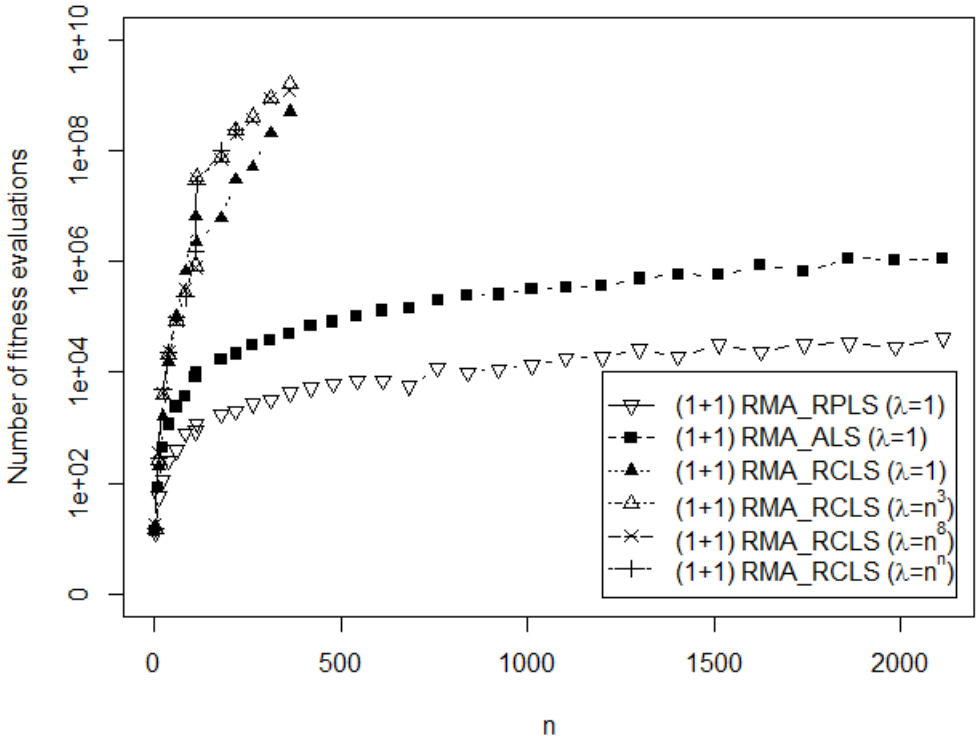


Figure 7: A comparison of the average number of fitness evaluations (on a log-scale) as n grows for the (1+1) RMA_RPLS ($\lambda = 1$), the (1+1) RMA_ALS ($\lambda = 1$), and the (1+1) RMA_RCLSs ($\lambda \in \{1, n^3, n^8, n^n\}$) on the $G_{RP}(t)$, $n = t(2t + 2)$.

family G_{RP} with four different λs, $\lambda \in \{1, n^3, n^8, n^n\}$, respectively. The results of the above tested algorithms are shown in Figure 9. A comparison of the average number of fitness evaluations (on a log-scale) for all algorithms tested on the family G_{RP} is shown in Figure 7.

7 Conclusion and future work

We have formalized a (1+1) Restart Memetic Algorithm and two local searches, RCLS and RPLS, and run them on the fitness function $f_{\texttt{OPL}}$ to solve the Clique Problem. We then constructed two families of graphs, G_{RC} and G_{RP}, and showed that, for the first family of graphs G_{RC}, the (1+1) RMA_RCLS drastically outperforms the (1+1) RMA_RPLS, and vice versa for the second family of graphs G_{RP}. Furthermore, we proposed a (1+1) Restart Memetic Algorithm with an Alternative Local Search, and showed that the proposed algorithm is expected to solve the Clique Problem on both families of graphs efficiently. This indicates that an MA with multiple local searches can outperform MAs

with a single local search in some instances of the Clique Problem. We also empirically verified our theoretical results with computer programs.

For future work, we suggest analyzing the runtime performance of MAs with more local search operators, and with a dynamic or adaptive strategy that will decide which local search operator should be used.

References

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

[2] E. K. Burke and D. J. L. Silva, "The design of memetic algorithms for scheduling and timetabling problems," in *Recent Advances in Memetic Algorithms, Studies in Fuzziness and Soft Computing*, W. H. N. Krasnogor and J. Smith, Eds., 2004, vol. 166, pp. 289–312.

[3] 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.

[4] M. J. Dinneen and K. Wei, "On the analysis of a (1+1) adaptive memetic algorithm," in *Proceedings of Memetic Computing, MC2013*. IEEE, 2013, pp. 24–31.

[5] ——, "A (1+1) adaptive memetic algorithm for the maximum clique problem," in *2013 IEEE Conference on Evolutionary Computation*, vol. 1, June 20-23 2013, pp. 1626–1634.

[6] S. Droste, T. Jansen, and I. Wegener, "On the optimization of unimodal functions with the (1+1) evolutionary algorithm," in *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*. Springer-Verlag, London, UK, 1998, pp. 13–22.

[7] ——, "On the analysis of the (1+1) evolutionary algorithm," *Theoretical Computer Science*, vol. 276, pp. 51–81, 2002.

[8] C. GieBen, "Hybridizing evolutionary algorithms with opportunistic local search," in *Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference*, ser. GECCO '13, New York, NY, USA, 2013, pp. 797–804.

[9] O. Giel and I. Wegener, "Evolutionary algorithms and the maximum matching problem," in *STACS'03: Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science*. Springer-Verlag, London, UK, 2003, pp. 415–426.

[10] T. Jansen, K. A. D. Jong, and I. Wegner, "On the choice of the offspring population size in evolutionary algorithms," *Evol, Comput.*, vol. 13, no. 4, pp. 413–440, 2005.

[11] T. Jansen and I. Wegener, "Evolutionary algorithms: How to cope with plateaus of constant fitness and when to reject strings of the same fitness," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 6, pp. 589–599, 2001.

[12] ——, "On the analysis of evolutionary algorithms—a proof that crossover really can help," *Algorithmica*, vol. 34, no. 1, pp. 47–66, 2002.

[13] T. Jansen, *Analyzing Evolutionary Algorithms: The Computer Science Perspective*. Springer Berlin Heidelberg, 2013.

[14] T. Kotzing, 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, New York, NY, USA, 2011, pp. 989–996.

[15] N. Krasnogor and J. Smith, "Emergence of profitable search strategies based on a simple inheritance mechanism," in *Proceedings of the 2001 Genetic and evolutionary computation (GECCO 2001)*, N. Krasnogor, Ed., 2001, pp. 432–439.

[16] ——, "A tutorial for competent memetic algorithms: model, taxonomy, and design issues." *IEEE Trans. Evolutionary Computation*, vol. 9, no. 5, pp. 474–488, 2005.

[17] F. Neri, C. Cotta, and P. Moscato, *Handbook of Memetic Algorithms*. Studies in Computational Intelligence, 2011, vol. 379.

[18] F. Neri, J. Toivanen, G. L. Cascella, and Y.-S. Ong, "An adaptive multimeme algorithm for designing hiv multidrug therapies," *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, vol. 4, no. 2, pp. 264–278, Apr. 2007.

[19] F. Neumann and I. Wegener, "Randomized local search, evolutionary algorithms and the minimum spanning tree problem," in *Proceedings of the annual conference on Genetic and evolutionary computation (GECCO'04)*, 2004, pp. 713–724, lNCS 3102. Springer.

[20] F. Neumann and C. Witt, *Bioinspired Computation in Combinatorial Optimization: Algorithms and Their Computational Complexity*, 1st ed. New York, NY, USA: Springer-Verlag New York, Inc., 2010.

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

[22] Y.-S. Ong, M.-H. Lim, N. Zhu, and K.-W. Wong, "Classification of adaptive memetic algorithms: a comparative study," *Trans. Sys. Man Cyber. Part B*, vol. 36, no. 1, pp. 141–152, Feb. 2006. [Online]. Available: http://dx.doi.org/10.1109/TSMCB.2005.856143

[23] C. Qian, Y. Yu, and Z. H. Zhou, "An analysis on recombination in multi-objective evolutionary optimization," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation (GECCO '11)*, N. Krasnogor, Ed. ACM, New York, NY, USA, 2011, pp. 2051–2058.

[24] G. Rudolph, "Finite markov chain results in evolutionary computation: A tour d'horizon," *Fundamenta Informaticae*, vol. 35, no. 1–4, pp. 67–89, 1998.

[25] J. E. Smith, "Coevolving memetic algorithms: A review and progress report," *Trans. Sys. Man Cyber. Part B*, vol. 37, no. 1, pp. 6–17, Feb. 2007.

[26] D. Sudholt, "On the analysis of the (1+1) memetic algorithm," in *Proceedings of the 8th annual conference on Genetic and evolutionary computation.*, 2006, pp. 493–500.

[27] 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)*, ser. LNCS, vol. 6506. Springer, 2010, pp. 340–352.

[28] D. Sudholt, "Memetic algorithms with variable-depth search to overcome local optima," in *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, ser. GECCO '08, 2008, pp. 787–794.

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

[30] C. Witt, "Worst-case and average-case aximations by simple randomized search heuristic," in *Proc. of the 22nd Annual Symposium on Theoretical Aspects of Computer Science (STACS'05)*, ser. LNCS, vol. 3804. Springer, 2005, pp. 44–56.

[31] ——, "Runtime analysis of the (μ+1) ea on simple pseudo-boolean functions," in *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, Seattle, Washington, USA*, 2006, pp. 651–658.

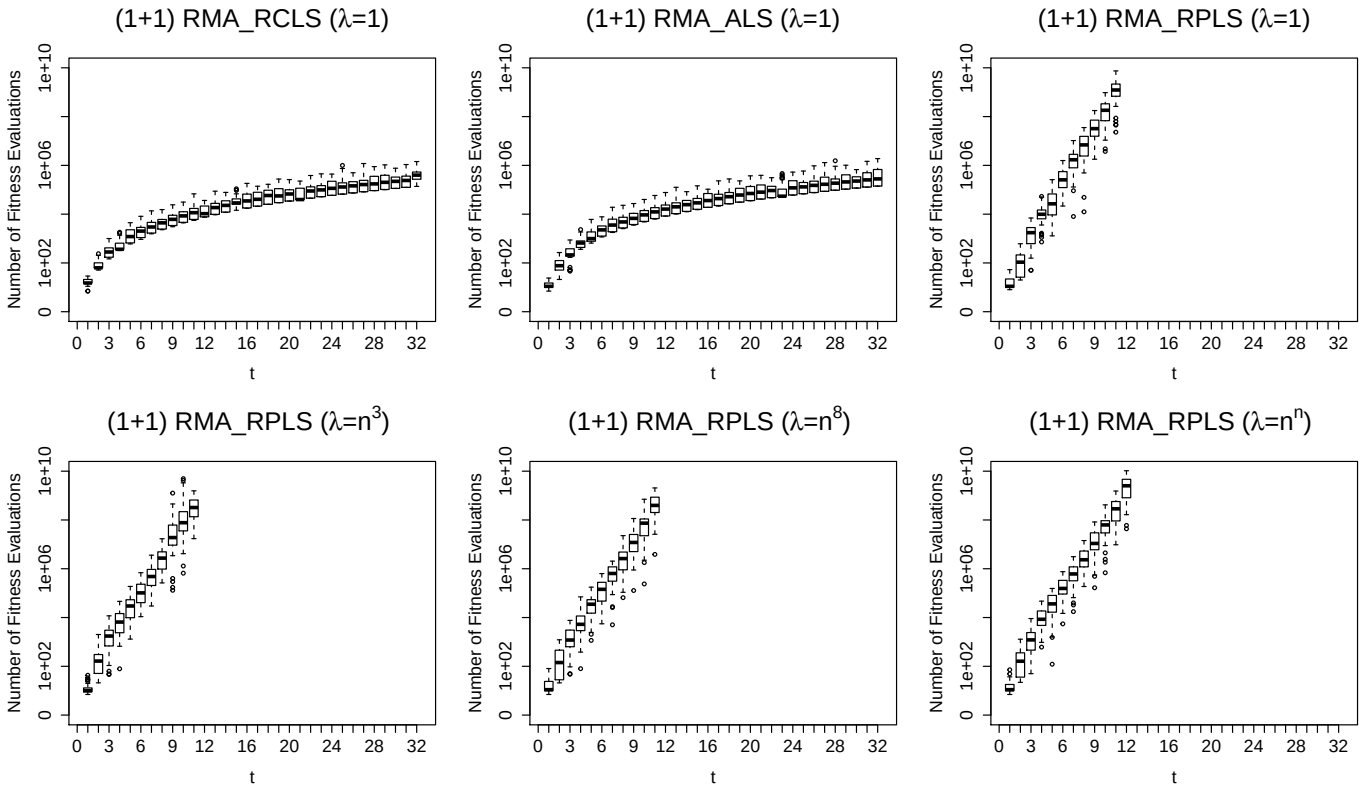


Figure 8: The number of fitness evaluations (on a log-scale) as t grows for the (1+1) RMA_RCLS ($\lambda = 1$), the (1+1) RMA_ALS ($\lambda = 1$), and the (1+1) RMA_RPLSs ($\lambda \in \{1, n^3, n^8, n^n\}$) on the $G_{RC}(t)$, $n = t(2t + 2)$.

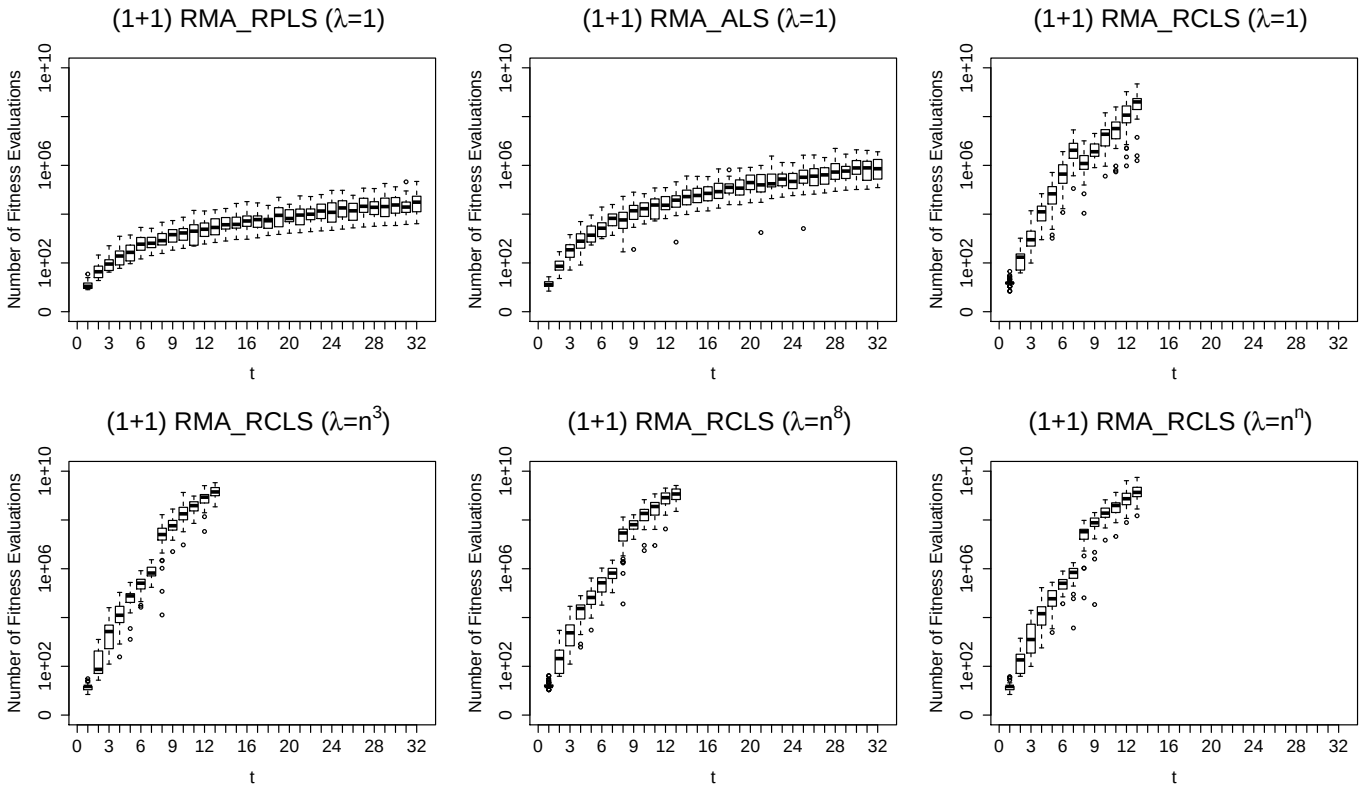


Figure 9: The number of fitness evaluations (on a log-scale) as t grows for the (1+1) RMA_RPLS ($\lambda = 1$), the (1+1) RMA_ALS ($\lambda = 1$), and the (1+1) RMA_RCLSs ($\lambda \in \{1, n^3, n^8, n^n\}$) on the $G_{RP}(t)$, $n = t(2t + 2)$.