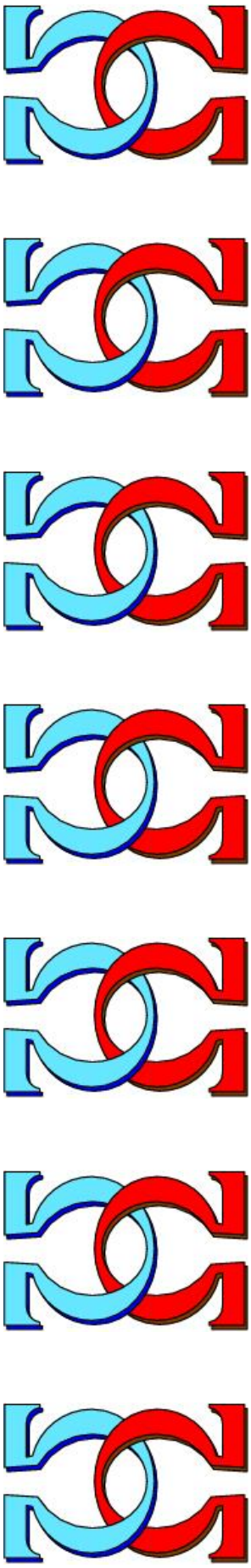


CDMTCS
Research
Report
Series

Runtime Analysis of a (1+1) Adaptive Memetic Algorithm and the Maximum Clique Problem

Michael J. Dinneen
Kuai Wei

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

CDMTCS-424
August 2012 (updated June 2013)

Centre for Discrete Mathematics and
Theoretical Computer Science

Runtime Analysis of a (1+1) Adaptive Memetic Algorithm and the Maximum Clique Problem

Michael J. Dinneen and Kuai Wei*

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

{mjd,kuai}@cs.auckland.ac.nz

Abstract

A memetic algorithm is an evolutionary algorithm augmented with a local search. For many applications, researchers have applied variations of memetic algorithms and have gained very positive experimental results. But the theory of these variations of memetic algorithms is still underdeveloped.

This paper defines the *(1+1) adaptive memetic algorithm* (AMA) with a dynamic mutation probability, and analyzes two types of local searches. We then propose different classes of functions for studying the performance of evolutionary algorithms. We give time complexity analysis that proves our two local searches can outperform each other on different functions. Also we show that memetic algorithms with dynamic mutation probabilities can outperform memetic algorithms with static mutation probabilities, and vice versa.

Then, we focus on the NP-hard Maximum Clique Problem, and show the success of our proposed (1+1) AMA. We propose a new metric (expected running time to escape a local optimal), and show how this metric dominates the expected running time of finding a maximum clique. Then based on this new metric, we show the above analyzed algorithms are expected to find a maximum clique on planar graphs, bipartite graphs and sparse random graphs in a polynomial time in the number of vertices.

Also based on our new metric, we will show that if an algorithm takes an exponential time to find a maximum clique of a graph, it must have been trapped into at least one local optimal which is extremely hard to escape. Furthermore, we will show that our proposed (1+1) AMA with a random permutation local search is expected to escape these (hard to escape) local optimal cliques drastically faster than the well-known basic (1+1) EA. The success of our experimental results also shows the benefit of our adaptive strategy combined with the random permutation local search.

*Corresponding author.

1 Introduction

A Memetic Algorithm (MA) is a meta-heuristic algorithm which combines an Evolutionary Algorithm (EA) and a local search algorithm. It is generally believed that MAs are successful because they inherit both the exploratory search ability of evolutionary algorithms and the neighborhood search ability of local search methods [BS04]. This property has led to many implementations of MAs, and the highlighted experimental results have verified the advantages of MAs. In 2011, an overview of MAs shows the usefulness of MAs in many applications [NCM12].

The experimental studies of MAs have grown rapidly, but theoretical studies have not kept up with the state-of-the-art of MAs. Since MAs combine EAs and local searches, to study the theory of MAs, we need to start from EAs first.

1.1 (1+1) EA and its variants

There are a number of theoretical investigations on EAs in the literature. A survey can be found in [OHY07b]. In short, the time complexity analysis of EAs started from the basic (1+1) EA on simple pseudo-boolean functions [Rud98] in 1998, Onemax [Rud98] in 1998, Trap Functions [DJW98] in 1998, and plateaus of constant fitness [JW01] in 2001. In 2002, Droste, Jansen and Wegener [DJW02] summarized the basic (1+1) EA, where most theoretic studies of EAs are based on this algorithm. The term (1+1) represents that the population size of parents and children are both one. After the basic (1+1) EA has been studied, a very important progress is the analysis on population-based EAs, such as the $(\mu + 1)$ EA [Wit06], $(1 + \lambda)$ EA [JJW05]. Also many researches have focused on showing the crossover operation is essential in EAs such as [JW02, KST11, QYZ11].

Meantime, after the basic (1+1) EA were analyzed, some variants of EAs have been formalized and analyzed on some artificially created functions. These studies help us understand what characteristics of these algorithms may make their optimizations easier or harder than the basic (1+1) EA. Examples are the Dynamic (1+1) EA [JW06] in 2006, (1+1) MA [Sud06] in 2006, (1+1) Genetic Programming [DNO11] in 2011 and (1+1) AMA [DW13] in 2013. Note the Dynamic (1+1) EA shows the usefulness of a dynamic mutation approach for EAs and in this paper we claim that dynamic mutation is also crucial in MAs.

Apart from analyzing those toy functions (`ONEMAX`, `BIN`, and `LEADINGONES`, etc.), many researches have started to analyze the EAs for main-stream combinatorial optimization problems such as the Maximum Matching Problem [GW03] in 2003, the Minimum Spanning Tree Problem [NW04] in 2004, and the Partition Problem [Wit05] in 2005.

All these theoretical studies help us to understand how EAs and their variants find a global optimum on specific problems. However, we need more rigorous research on NP-hard problems (which sometimes needs exponential time). A few results studied NP-hard problems but with restriction on the input cases. Such as Storch analyzed the Maximum Clique Problem but only in planar graphs in 2006 [Sto06]; Oliveto, He and Yao analyzed the Vertex Cover Problem, but only focused on Papadimitriou-Steiglitz graphs in 2007 [OHY07a], and on bipartite graphs in 2008 [OHY08]; Witt analyzed the Vertex Cover Problem but only on sparse random graphs in 2012 [Wit12]; and Sudholt

and Zarges analyzed the Vertex Coloring Problem on bipartite graphs, sparse random graphs and planar graphs in 2010 [SZ10].

This implies that the reason why Evolutionary Algorithms are efficient on NP-hard problems is still underdeveloped. In order to make a step to this goal, we investigate the running time of two EAs and our proposed AMAs on the Maximum Clique Problem (MCP). Because the MCP is NP-complete, which implies every other problem in NP can be transformed into MCP in polynomial time, our study in this paper is also related to other NP-hard problems.

1.2 (1+1) MA and our (1+1) AMA

After studying the basic (1+1) EA and its variants, we come back on the MAs. In 2006, a theoretical analysis of a (1+1) MA from [Sud06] defined a simple MA with a fixed mutation probability and a local search, which provided another insight into the interaction of mutation and local search. Some related studies can be found in [Sud09, SZ10, Sud11, Wit12]. However, there are still some gaps between theory and applications.

Recently, for many applications, researchers have applied MAs with a dynamic mutation probability, or MAs with different local search approaches. Both have gained very positive experimental results. For example, we recently in [DLW11] designed a adjusting mutation approach and an improved local search in MA and gained success on scheduling problems. However, these results do not prove, from the theory point of view, the reason why the dynamic mutation probability can achieve better performance, or why different local search approaches have such diverse performances. The theory of these variations of MAs is still underdeveloped and is the focus of this paper.

This paper will define a (1+1) Adaptive Memetic Algorithm (AMA) to analyze the dynamic mutation probability using two different (but natural) local search approaches– Random Permutation Local Search (RPLS) and Random Complete Local Search (RCLS). The paper is structured as follows. In Section 2, we formalize a (1+1) AMA and two different local search approaches.

In Section 3, we first analyze the running time of the (1+1) AMA on different functions. Secondly, we analyze two local search approaches, and prove that these local search approaches can drastically outperform each other on different functions. Lastly, we prove that on some functions, the (1+1) AMA can drastically outperform each static (1+1) MAs; while on some other functions, a static (1+1) MA can drastically outperform the (1+1) AMA.

In Section 4, we focus on the Clique Problem. The Clique Problem is defined in Section 4.1. In Section 4.2, we first define our new metric, and analyze the upper bounds of each algorithm to find a maximum clique, then show that those algorithms are expected to find a maximum clique on certain families of sparse graphs in polynomial time. In Section 4.3, we show that if a graph needs an exponential time to escape a local optimal clique, the (1+1) AMA with RPLS will be drastically faster to escape than the basic (1+1) EA. In Section 4.4, experimental results provides a running time comparison among the (1+1) EA [DJW02], the Dynamic (1+1) EA [JW06], (1+1) MA [Sud06], the (1+1) AMA [DW13], and the SMA [PHK11] on the Maximum Clique Problem.

Our conclusions and future work will be given in Section 5.

2 Algorithm Definitions

In this section we give basic definitions of our algorithms and we begin with the following standard notation 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 Algorithms to be compared with ours

The algorithms that we will use to compare with our proposed (1+1) Adaptive Memetic Algorithm are the (1+1) EA [DJW02], the Dynamic (1+1) EA [JW06] and the (1+1) MA [Sud06]. Note these algorithms all try to maximize a function $f : \{0, 1\}^n \to \mathcal{R}$. The time complexity analysis in this paper looks at the number of evaluations of this fitness (objective) function. The algorithms are stated as below:

Algorithm 1. (1+1) EA.

1. $p_m := 1/n$.
2. Choose randomly an initial bit string $x \in \{0, 1\}^n$.
3. Repeat the following mutation step:
   (a) Compute x' by flipping independently each bit x_i with probability p_m.
   (b) If $f(x') \geq f(x)$ then $x := x'$.

Algorithm 2. Dynamic (1+1) EA.

1. Choose a sequence $p_t(n) \in (0, 1/2)$ called mutation probabilities for step t.
2. Choose $x \in \{0, 1\}^n$ uniformly at random. $t := 1$.
3. Let y be the result of flipping each bit in x independently with probability $p_t(n)$ (mutation).
4. If $f(y) \geq f(x)$ then $x := y$ (selection).
5. Increase t by 1.
6. Stop if meet some stopping criterion; otherwise, go to step 3.

where $p_t(n) = 2^{t^*}/n$ with $t^* \equiv (t-1) \bmod (\lceil \log n \rceil - 1)$.

Algorithm 3. (1+1) MA.

1. Choose $x \in \{0,1\}^n$ uniformly at random.
   $x := \texttt{LocalSearch}(x)$.
2. $y := x$. Flip every bit in y with probability p_m.
   $y := \texttt{LocalSearch}(y)$.
3. If $f(y) \geq f(x)$ then $x := y$.
4. Go to step 2.

where the Local Search in Step 1 and 2 is a Random Complete Local Search which we will state in Algorithm 5.

2.2 (1+1) Adaptive Memetic Algorithm

The (1+1) Adaptive Memetic Algorithm (AMA) also tries to maximize a function $f : \{0,1\}^n \to \mathcal{R}$. The time complexity analysis of the (1+1) AMA looks at the number of evaluations of this fitness (objective) function.

The term "adaptive" denotes that the mutation probability is adjusted dynamically. In every generation, if the offspring has a better fitness value than its parent, we treat this as a positive feedback, and decrease the mutation probability. So that the next mutation will search the nearby solution space. Otherwise, if the offspring could not perform better than its parent, we treat it as a negative feedback, and want to enlarge the searching space by increasing the mutation probability.

Recall (1+1) represents one individual and one offspring, and the (1+1) AMA contains a mutation operation with a dynamic mutation probability, and a local search operation. A template of the generic algorithm is as follows:

Algorithm 4. (1+1) AMA for functions $f : \{0,1\}^n \to \mathcal{R}$

1. Initialize the mutation probability $p \in [0,1]$.
2. Choose $x \in \{0,1\}^n$ uniformly at random.
3. $y := \texttt{Mutation}(x)$.
4. $z := \texttt{LocalSearch}(y)$.
5. $p := \texttt{Adaptive}(f(x), f(z), p)$.
6. If $f(z) \geqslant f(x)$ then $x := z$.
7. Stop if meet some stopping criterion.
8. Otherwise, go to step 3.

We do not yet specify any stopping criterion since we will analyze it on different functions. In general, the stopping criterion can be either a certain number of iterations that the algorithm has executed, a certain duration of time, finding an expected fitness value $f(x)$, etc. The functions `Aadaptive` and `LocalSearch` will be stated in the subsections below.

2.2.1 Adaptive mutation probability

Since the mutation probability is adjusted dynamically, with the initial p in step 1 of Algorithm 4 being $1/n$, the mutation function in step 3 independently flips every bit in x with probability p.

The mutation in the (1+1) AMA is mainly used for making a jump in the search space when the algorithm stagnates into a local optimal solution. Meanwhile, the mutation probability $p = 1/n$ means that the expected number of flipped bits is one. So the adaptive function in step 5 is chosen as below:

$$p = \begin{cases} \frac{1}{n}, & \text{if } f(z) > f(x) \text{ or } p = \frac{1}{2} \\ \min\left(2p, \frac{1}{2}\right), & \text{otherwise.} \end{cases}$$

We do not state this dynamic schedule is optimal, but we claim it is reasonable because:

1. If the local search has trapped into a local optimal solution, then we should increase the mutation probability to jump to another region of the search space. In practice, we note that small mutation probabilities are more helpful for improving to a local optimal. However to avoid stagnation, we should increase (say double) the mutation probability.

   Note that if the mutation probability reaches the upper bound $1/2$, we treat it as a trigger and reset the mutation probability to $1/n$ again. This is because we want to preserve the overall geometric distribution of the mutation probabilities.

2. On the other hand, after finishing the mutation and the local search, if we have found a better solution, then we do not want to jump to a distant solution space immediately, but want to search its nearby space. So we decrease the mutation probability.

2.2.2 Local search

As stated before, the local search in step 4 of Algorithm 4 can have many variations. A *Random Complete Local Search* (RCLS) is used quite often in MAs such as the (1+1) MA in [Sud06].

Algorithm 5. Random Complete Local Search (RCLS). For a given string $x \in \{0,1\}^n$:

1. $t := 1$.
2. $\texttt{BestNeighborSet} := \left\{ \begin{array}{l} y \mid f(y) > f(x), \texttt{Hamming}(x,y) = 1, \text{ and} \\ \forall z \text{ with } \texttt{Hamming}(x,z) = 1 \rightarrow f(y) \geq f(z) \} \end{array} \right\}$.

3. Stop and return x if `BestNeighborSet` $= \emptyset$.

4. x is randomly choosen from `BestNeighborSet`.

5. Stop if the stopping criterion holds (see below). Otherwise, go to step 2.

where `Hamming`(x, y) is the number of different bits between x and y. The RCLS performs n fitness evaluations per step. Hence, the stop criterion in step **??** is when $t = n$, or there is no neighbor solution which has a better fitness value, i.e. the set `BestNeighbors` is empty.

In this paper, we formalize a *Random Permutation Local Search* (RPLS) (also known as random bit-climbing algorithm [Dav91]). Unlike RCLS that evaluates n neighbors to execute one flip, 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 6. 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 `Per` of length n.

2. $i := 1$, `WorseCount` $:= 0$.

3. $y :=$ `flip`$(x,$ `Per`$[i])$.

4. If $f(y) > f(x)$ then $x := y$, `WorseCount` $:= 0$.

5. `WorseCount` $:=$ `WorseCount` $+ 1$.

6. $i := (i \mod n) + 1$.

7. Stop if the stopping criterion holds (see below). Otherwise, go to step 3.

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

Note that the RPLS uses only one fitness evaluation per step, so the RPLS will stop when $t = n^2$, or there is no neighbor solution which has a better fitness value, i.e. `WorseCount` $= n$.

Example 7. Suppose string $x = (0, 0, 0, 0)$, and `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 `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 expected running time for the local search in step 4 of Algorithm 4 is n^2 number of fitness evaluations, or there is no neighbor solution which has a better fitness value. Dirk Sudholt [Sud06] used the long 2-path problem to show that the running time of the local search in MAs should be polynomially bounded. Here we do not investigate the actual optimal polynomial bound but simply set an upper bound of n^2 number of fitness evaluations for the above two local searches. In the rest of this paper, we will use AMA_RCLS to denote the algorithm AMA using RCLS as the local search, and use AMA_RPLS to denote the algorithm AMA using RPLS as the local search.

3 Algorithm Analysis and Comparisons

In this section, we will analyze the expected running time of the (1+1) AMA with the above two local search approaches (RCLS and RPLS of Section 2.2.2) on different functions. Then we prove that these two local search approaches (without evolutionary features) can drastically outperform each other on different functions. Finally we study static versus dynamic mutation probabilities for MAs, and show that on some functions, the (1+1) AMA can drastically outperform each static (1+1) MAs; while on some other functions, a static (1+1) MA can drastically outperform the (1+1) AMA.

We first cite some important functions on a given string $x = (x_1, x_2, ..., x_n) \in \{0,1\}^n$, which were defined in [DJW02] and [Sud06]:

Definition 8. The *Hamming distance* function of two strings x and x' of length n is defined as:

$$\texttt{Hamming}(x, x') = \sum_{i=1}^{n} |x_i - x'_i|.$$

Definition 9. The function `ONEMAX` calculates the number of ones in the string x, and is formalized as:

$$\texttt{ONEMAX}(x) = \sum_{i=1}^{n} x_i.$$

Definition 10. The function `BIN` reads a string as a binary representation of an integer, which is:

$$\texttt{BIN}(x) = \sum_{i=1}^{n} 2^{n-i} x_i.$$

Definition 11. The set of *linear functions* has the form:

$$f(x) = \sum_{i=1}^{n} \omega_i x_i + c,$$

where $c, \omega_i \in \mathcal{R}$.

Definition 12. The function `LEADINGONES` computes the number of consecutive ones in x from left to right:

$$\texttt{LEADINGONES}(x) = \sum_{i=1}^{n} \Pi_{j=1}^{i} x_j.$$

Definition 13. Let $|x|_i$ be the number of bits with value i in x. Let $x \in \{0,1\}^n$ be divided into two parts, $x = x'x''$, with $x' \in \{0,1\}^{n-1}$, $x'' \in \{0,1\}$. Then the function `ZZO` (zero, zeros, one) is defined as:

$$\texttt{ZZO}(x) = \begin{cases} -3n, & \text{if } x' \neq 0^{n-1} \text{ and } x'' = 1, \\ |x'|_0 - 2n, & \text{if } x' \neq 0^{n-1} \text{ and } x'' = 0, \\ |x''|_1 - n, & \text{otherwise (i.e. if } x' = 0^{n-1}). \end{cases}$$

Note that the purpose of the function ZZO is to guide the local searches to reach $0^{n-1}1$. If the start string is not $0^{n-1}1$, local searches on ZZO will first flip the last bit x'' to 0. Then flip all bits in x' to 0^{n-1}. Finally they flip the last bit x'' to 1 and stop. So the optimal search string will end up as $0^{n-1}1$.

3.1 Running time analysis on the (1+1) AMA

In this subsection, we will analyze the expected running time of the (1+1) AMA with two different local search approaches, i.e. RPLS and RCLS.

Lemma 14. *The expected number of steps the mutation approach in Algorithm 4 with probability $p = 1/2$ takes to optimize an arbitrary bit string to a global optimum is $O(2^n)$.*

Proof. Let $x \in \{0,1\}^n$ be an arbitrary bit string that we start to mutate, and x^* be a global optimum bit string of the function f. Let $H(x, x^*)$ denotes the *Hamming distance* between x and x^*, where $0 \leqslant H(x, x^*) \leqslant n$. Then the probability of the mutation to get x^* in one step is $p^{H(x,x^*)} \cdot (1-p)^{n-H(x,x^*)} = \left(\frac{1}{2}\right)^{H(x,x^*)} \cdot \left(1-\frac{1}{2}\right)^{n-H(x,x^*)} = \left(\frac{1}{2}\right)^n$. Thus the expected number of steps until this event happens, i.e. a global optimum has been found, is $O(2^n)$. □

Theorem 15. *The expected running time of the (1+1) AMA-RPLS and the (1+1) AMA-RCLS for an arbitrary fitness function is $O(2^n \cdot n^2 \cdot \log n)$.*

Proof. We prove the theorem in two steps:

1. First, suppose we disable the positive feedback in every generation, i.e. do not decrease the mutation probability when the offspring performs better than its parent. Then in every generation we will take $O(n^2)$ steps for the local search, where both RPLS and RCLS take up to $O(n^2)$ steps as shown in Section 2.2.2. Also, in every $\lceil \log \frac{n}{2} \rceil$ generations we will take one mutation with the mutation probability $p = \frac{1}{2}$. Hence, based on Lemma 14, the upper bound of the (1+1) AMA-RPLS and the (1+1) AMA-RCLS without the positive feedback is $O(2^n \cdot n^2 \cdot \log n)$.

2. Second, if the positive feedback occurs, i.e. we find a better solution and will decrease the mutation probability. Although this will prevent the mutation probability reaching $\frac{1}{2}$, we know that the number of times this positive feedback can occur is at most 2^n before we finding a global optimum. So Theorem 15 still holds when we use the positive feedback to decrease the mutation probability. □

Lemma 16. *The following bounds hold for the expected running time of the RPLS and RCLS on various functions:*

1. *ONEMAX: RPLS runs in $\Theta(n)$ steps, and RCLS runs in $\Theta(n^2)$ steps.*

2. *BIN: RPLS runs in $\Theta(n)$ steps, and RCLS runs in $\Theta(n^2)$ steps.*

3. linear functions*: RPLS runs in $\Theta(n)$ steps, and RCLS runs in $\Theta(n^2)$ steps.*

4. *LEADINGONES: RPLS runs in $\Theta(n^2)$ steps, and RCLS runs in $\Theta(n^2)$ steps.*

Proof. The RPLS will check each bit x_i in x, flip x_i if this change can achieve a higher fitness value. The sequence of i follows the random permutation `Per`. Thus the expected running time of the RPLS is $\Theta(n)$ for `ONEMAX`, `BIN` and *linear functions* because it will find the optimum result when checking all numbers in `Per` once.

For the function `LEADINGONES`, the RPLS will flip at least one bit of x when it checks all numbers in `Per` once, which needs n steps, so the RPLS will flip all bits of x to one before it checks n times of all numbers in `Per`, thus the upper bound is proved.

Now we prove the lower bound of the RPLS on the function `LEADINGONES`. Let $\lambda := \sum_{i=1}^{n} \overline{x_i}$ and $\mathcal{I}$ be an integer array such that $|\mathcal{I}| = \lambda$, $\forall i \in \mathcal{I} \rightarrow x_i = 0$, and $\mathcal{I}_a < \mathcal{I}_b \Leftrightarrow a < b$ where $\mathcal{I}_a$ and $\mathcal{I}_b$ are the a-th and the b-th integers in $\mathcal{I}$ respectively. So we have: for each iteration when searching along the permutation array, let $\mathcal{I}_i$ be the first bit we flip to one in x, then the next bit we need to flip is $\mathcal{I}_{i+1}$. The probability that the index of the number $\mathcal{I}_{i+1}$ in the permutation array is larger than the index of the number $\mathcal{I}_i$ in the permutation array is $1/2$, i.e. the probability that the RPLS will flip at least two bits in one iteration of searching the permutation array is $1/2$. Hence the probability that the RPLS will flip at least t bits in one iteration of searching the permutation array is $1/2^t$. Thus, when the RPLS searches one iteration of the permutation array, a constant number of bits is expected to be flipped to one. Also we know the probability that the random initial bit string has more than $(1/4)n$ bits of zeros is exponentially close to one, thus the expected number of iterations to check the permutation array is $\Omega(n)$, and each iteration checking the permutation array needs n steps of fitness evaluation, thus the lower bound is proved. Therefore, the RPLS runs in $\Theta(n^2)$ on the function `LEADINGONES`.

The RCLS is different from the RPLS because it will search all n bits in x, and then choose one bit in x to flip. So it takes n steps to flip one bit, therefore, the RCLS will take n^2 steps to flip all bits in the string x if required. And the expected running time of the RCLS is $\Theta(n^2)$ for `ONEMAX`, `BIN`, *linear functions* and `LEADINGONES`. □

Theorem 17. *The following bounds hold for the expected running time of the (1+1) AMA-RPLS and the (1+1) AMA-RCLS on the following functions:*

1. *ONEMAX: (1+1) AMA-RPLS takes $\Theta(n)$ time, and (1+1) AMA-RCLS takes $\Theta(n^2)$ time.*
2. *BIN: (1+1) AMA-RPLS takes $\Theta(n)$ time, and (1+1) AMA-RCLS takes $\Theta(n^2)$ time.*
3. linear functions*: (1+1) AMA-RPLS takes $\Theta(n)$ time, and (1+1) AMA-RCLS takes $\Theta(n^2)$ time.*
4. *LEADINGONES: (1+1) AMA-RPLS takes $\Theta(n^2)$ time, and (1+1) AMA-RCLS takes $\Theta(n^2)$ time.*

Proof. Recall that the local search approach in the (1+1) AMA takes up to $O(n^2)$ steps (see Section 2.2.2, and for all functions that can be optimized by the local search approach within $O(n^2)$ steps, they can be optimized by the corresponding (1+1) AMA with

Table 1: A comparison of the algorithms.

	(1+1) EA	Dynamic (1+1) EA	(1+1) AMA-RPLS	(1+1) AMA-RCLS
arbitrary functions	$O(n^n)$	$O(4^n \log n)$	$O(2^n n^2 \log n)$	$O(2^n n^2 \log n)$
ONEMAX	$\Theta(n \log n)$	$\Theta(n \log^2 n)$	$\Theta(n)$	$\Theta(n^2)$
BIN	$\Theta(n \log n)$	$\Theta(n \log^2 n)$	$\Theta(n)$	$\Theta(n^2)$
linear functions	$\Theta(n \log n)$	$O(n^2 \log n)$	$\Theta(n)$	$\Theta(n^2)$
LEADINGONES	$\Theta(n^2)$	$\Theta(n^2 \log n)$	$\Theta(n^2)$	$\Theta(n^2)$

Results of the (1+1) EA and the Dynamic (1+1) EA can be found in [DJW02] and [JW06] respectively.

that local search approach by the same lower bound and upper bound. Therefore, based on the Lemma 16, the Theorem 17 is proved. □

A summary of the expected running time of different algorithms on ONEMAX, BIN, *linear functions*, and LEADINGONES is shown in Table 1. Results of the (1+1) EA and the Dynamic (1+1) EA can be found in [DJW02] and [JW06], respectively.

From Table 1, we see that the (1+1) AMA-RPLS and (1+1) AMA-RCLS are more efficient than the (1+1) EA on *arbitrary functions*. Meanwhile, the (1+1) AMA-RPLS is more efficient than the other three counterparts on functions ONEMAX, BIN and *linear functions*. On the function LEADINGONES, the (1+1) EA, (1+1) AMA-RPLS and (1+1) AMA-RCLS are more efficient than the Dynamic (1+1) EA.

3.2 RPLS and RCLS can drastically outperform each other

In this subsection, we show that there exist functions causing that the probability of the RPLS to get trapped is exponentially close to 1, while the probability of the RCLS to obtain a global optimum within a polynomial running time is exponentially close to 1, and vice versa.

Definition 18. Let $x \in \{0,1\}^n$ be divided into two parts, $x = x'x''$, with $x' \in \{0,1\}^{n_1}$, $x'' \in \{0,1\}$ for $n_1 = n-1$. The function $T_{\texttt{RPLS}}$ (trap for RPLS) is defined by

$$T_{\texttt{RPLS}}(x) = \begin{cases} \texttt{ZZO}(x), & \text{if } x'' = 0. \\ n + 3i + 4k, & \text{if } x'' = 1, \text{ and} \\ & x' = 1^i 0^j 1^k 0^{n_1-i-j-k}, \\ & \text{for } i \geqslant 0, j \geqslant 1, k \geqslant 0. \\ -\texttt{ONEMAX}(x'') - 4n, & \text{otherwise.} \end{cases}$$

where ZZO is defined in Definition 13, and it will end up with string $x' = 0^{n_1}$, $x'' = 1$. We can see the global optimum string is 01^{n-1}. We say x is on the path if $x' = 1^i 0^j 1^k 0^{n_1-i-j-k}$ for $i \geqslant 0, j \geqslant 1, k \geqslant 0$.

Recall that RCLS will search all neighbor solutions and randomly choose one of the best fitness performance neighbor and restart the search. But the RPLS will search

neighbor solutions according to a permutation array, and move the first improved neighbor. Below is an example of the two local searches working on the the fitness function $T_{\texttt{RPLS}}$.

Example 19. Two different result on RPLS and RCLS.
Let $n = 5$, let the start string $x = 00101$, let the random permutation array for RPLS be 41352. Hence the fitness $T_{\texttt{RPLS}}(x) = 9$ as $n = 5$, $i = 0$ and $k = 1$.

- The RCLS would search:

  1. by flipping bit 1, will get $T_{\texttt{RPLS}}(10101) = 12$;
  2. by flipping bit 2, will get $T_{\texttt{RPLS}}(01101) = 13$;
  3. by flipping bit 4, will get $T_{\texttt{RPLS}}(00111) = 13$.

  So RCLS will randomly choose bit 2 or bit 4 to flip, and restart the search. Finally it will end up with the global optimum 01111.

- The RPLS would search according to the permutation array 41352:

  1. by flipping bit 4, will get $T_{\texttt{RPLS}}(00111) = 13$, which is improved. Then move to this string and restart the search with permutation array 13524;
  2. by flipping bit 1, will get $T_{\texttt{RPLS}}(10111) = 16$, which is improved. Then move to this string and restart the search with permutation array 35241;
  3. tries all neighbors and no better neighbor exists.

  So RPLS will end up with string 10111.

Theorem 20. *Both the RPLS and the RCLS will stop on the function $T_{\texttt{RPLS}}$ in $O(n^2)$ steps. Meanwhile, the probability that the RCLS will find the global optimum solution is exponentially close to 1, but the probability that the RPLS find the global optimum solution is exponentially small.*

Proof. Part 1. Prove both RCLS and RPLS will stop in $O(n^2)$ steps. Firstly, if the start solution x is not on the path, both RCLS and RPLS will search along the function value $\texttt{ZZO}(x)$ until x is on the path. Note if the start string x is not on the path and $x'' = 1$, it will also flip the last bit x'' and then search along the function $\texttt{ZZO}(x)$. During this time, both algorithms will flip no more than $3n$ bits in x for the function ZZO, which needs at most $3n$ steps for the RPLS and $3n^2$ steps for the RCLS.

Secondly, once x reaches the path, both algorithms will climb along the path by flipping the bits in x' from zero to one until only one of zero exists in x'. This needs $O(n^2)$ steps. So the two algorithms will stop on the function in $O(n^2)$ steps.

Part 2. Prove the probability that the RCLS will find the global optimum solution is exponentially close to 1. Once x reaches the path, by flipping each bit in x' will have the result of either (a) increasing the fitness by three, (b) increasing the fitness by four, or (c) decreasing the fitness. So the RCLS will randomly choose one bit from the set of

bits that can increase the fitness by four, and flip that bit. So if the initial bit string in not on the path, the RCLS will end up with the global optimum $x' = 01^{n_1-1}$; and because $x'' = 1$, $x = 01^{n-1}$. If the initial bit string is on the path, and the first bit on the initial string is one, it will block the RCLS to find the global optimum solution, but the probability of this happens is exponentially small due to the fraction of the path is exponentially small compare to the solution space. Thus the probability of the RCLS to find the global optimum is exponentially large.

Part 3. Prove the probability that the RPLS find the global optimum solution is exponentially small. The RPLS will search each bit in x according to the sequence in the permutation, and flip the first bit that can increase the fitness value. So once it is on the path, and the RPLS reaches the number 1 in the permutation, it will check the first bit in x; and at this time, if x is not the global optimum 01^{n-1}, the RPLS will flip the first bit in x which can increase the fitness by three. After that, the RPLS can not find the global optimum in the end. Since the RPLS searches all numbers in the permutation string sequentially, the success permutation must have the property that (a) number one is the last number in the permutation; and (b) after searching the i-th number in the permutation, suppose we have the solution $x' = 0^j1^k0^{n_1-j-k}$, then the $(i+1)$-th number in the permutation must be either number j, $j+k+1$, or number n to point to the last bit x''. Otherwise, when the RPLS reaches the number 1 in the permutation, the solution will not be 01^{n-1}, and the RPLS will find out that by flipping the first bit of x can increase the fitness value, then will execute the flip, so can not find the global optimum.

Therefore, if the initial bit string is not on the path, we know that when RPLS finishes searching the function ZZO, it will get the string $x = 0^{n-1}1$. Then compare to all possible permutations, only a exponentially small fraction of permutation arrays can guide the RPLS to find the global optimum. So we say that in this case, the probability that a random permutation can let the RPLS successfully find the global optimum solution is exponentially small.

Also if the initial bit string is on the path, even if the RPLS also has the chances to reach the global optimum, we say this probability is exponentially small. This is because the probability of the initial bit being on the path is exponentially small (due to the fraction of the path compare to the solution space is exponentially small). □

Next, we show the opposite implication where we build the function like $T_{\texttt{RPLS}}$ but add a global fitness value $5n$.

Definition 21. Let $x \in \{0,1\}^n$ be divided into two parts, $x = x'x''$, with $x' \in$

$\{0,1\}^{n_1}$, $x'' \in \{0,1\}$ for $n_1 = n-1$. The function $T_{\texttt{RCLS}}$ (trap for RCLS) is defined by

$$T_{\texttt{RCLS}}(x) = \begin{cases} \texttt{ZZO}(x), & \text{if } x'' = 0. \\ 5n, & \text{if } x'' = 1, \text{ and} \\ & x' = 1^i 01^{n_1-i-1}, \\ & \text{for } i \geqslant 2. \\ n + 3i + 4k, & \text{if } x'' = 1, \text{ and} \\ & x' = 1^i 0^j 1^k 0^{n_1-i-j-k}, \\ & \text{for } i \geqslant 0, j \geqslant 1, k \geqslant 0. \\ -\texttt{ONEMAX}(x'') - 4n, & \text{otherwise.} \end{cases}$$

where ZZO is defined in Definition 13, and it will end up with string $x' = 0^{n_1}$, $x'' = 1$. We can see the global optimum string is $1^i 01^{n_1-i-1}$ for $i \geqslant 2$. We say x is on the path if $x' = 1^i 0^j 1^k 0^{n_1-i-j-k}$ for $i \geqslant 0, j \geqslant 1, k \geqslant 0$.

Theorem 22. *Both the RPLS and the RCLS will stop on the function $T_{\texttt{RCLS}}$ in $O(n^2)$ steps. Meanwhile, the probability that the RCLS will find the global optimum solution is exponentially small, but the probability that the RPLS find the global optimum solution is exponentially close to 1.*

Proof. Similarly to the proof for Theorem 20. The differences are:

1. The RCLS will have a probability, which is exponentially close to 1, to end up with string $x = 01^{n-1}$, but it is only a local optimal solution for $T_{\texttt{RCLS}}$.

2. For the RPLS, we can use the same method as in the proof for the Theorem 20 to prove that the probability that the RPLS does not end up with the string 01^{n-1} or 101^{n-2} is exponentially close to 1; that is the probability the RPLS finds the global optimum solution is exponentially close to 1.

□

After analyzing the relationship between the two local search approaches, next we will analyze the influence of the adaptive mutation probability. We compare the expected running time of the (1+1) AMA against the static (1+1) MAs.

3.3 (1+1) AMA can drastically outperform each static (1+1) MA

Here we need to cite two important functions PTJ (path to jump), and PWT (path with trap) in [JW06]. Then we define two new functions 2-PTJ and 2-PTW, where each two points x_a, x_b on the path have at least *Hamming distance* two, i.e. $\texttt{Hamming}(x_a, x_b) \geqslant 2$, for $x_a, x_b \in 1^{2i}0^{n-2i}$ and $a \neq b$.

The PTJ is defined as:

$$\texttt{PTJ}(x) = \begin{cases} n + i, & \text{if } x = 1^i 0^{n-i}, \\ 3n, & \text{if } x \in T, \\ n - \texttt{ONEMAX}(x), & \text{otherwise,} \end{cases}$$

where T is the global optimum containing all points x with $\texttt{ONEMAX}(x) \in [(3/4)n, (7/8)n]$, and $H(x, b) \geqslant n/16$ for all $b = 1^i 0^{n-i}$, $0 \leqslant i \leqslant n$.

Jansen and Wegener [JW06] have proved the running time on the function PTJ is bounded by $O(n^2 \log n)$ for the Dynamic (1+1) EA, but is exponential for each static (1+1) EA. Now we introduce a function 2-PTJ. We will prove that the expected running time for the (1+1) AMA on 2-PTJ is bounded by $O(n^4 \log n)$; but the expected running time for each static (1+1) MA on 2-PTJ is exponential.

Definition 23. Let $x \in \{0,1\}^n$ be divided into two parts, $x = x'x''$, with $x' \in \{0,1\}^{n_1}$, $x'' \in \{0,1\}$ for $n_1 = n - 1$. The function 2-PTJ is defined by

$$\texttt{2-PTJ}(x) = \begin{cases} \texttt{ZZO}(x), & \text{if } x'' = 0. \\ 2n, & \text{else if } x' = 1^{n_1}. \\ n + 2i, & \text{else if } x' = 1^{2i}0^{n_1-2i}, \\ & \quad \text{for } 2i < n_1. \\ 3n, & \text{else if } x' \in T. \\ -5n, & \text{else if } \texttt{ONEMAX}(x') \\ & > (3/4)n_1. \\ -4n - \texttt{ONEMAX}(x''), & \text{otherwise.} \end{cases}$$

where T is the global optimum containing all points x' with $\texttt{ONEMAX}(x') \in [(3/4)n_1, (7/8)n_1]$, and $H(x', b) \geqslant n_1/16$ for all $b = 1^i 0^{n_1-i}$, $0 \leqslant i \leqslant n_1$.

Note that in the fitness function 2-PTJ, each point on the path is a local optimal point, and this property has disabled the local search from climbing to 1^n directly. So the (1+1) AMA has to rely on both the mutation and the local search to climb along the path.

Theorem 24. *The expected running time of the (1+1) AMA on 2-PTJ is bounded by $O(n^4 \log n)$.*

Proof. First, the probability of the initial bit string with less than $(3/4)n$ ones is exponentially close to 1. Then the initial bit string will be in T, be on the path or fall into the function ZZO. Also, based on Lemma 16, the expected time to finish searching the fitness $\texttt{ZZO}(x)$ to reach the path or the global optimum is bounded by $O(n^2)$ because the local search will directly get on the path.

Next, we prove the expected time of the (1+1) AMA on 2-PTJ to reach $x' = 1^{n_1}$ or T from the path is $O(n^4 \log n)$. To climb from $x' = 1^{2i}0^{n_1-2i}$ to $x' = 1^{2i+2}0^{n_1-2i-2}$, we can let the mutation to flip one bit from zero to one, and the local search will definitely flip the other bit. So the probability to climb on the path is $\texttt{Prob}_{\texttt{climb}} = p(1-p)^{n-1} = \Omega\big(pe^{-p(n-1)}\big)$, and when $p = 1/n$, we have $\texttt{Prob}_{\texttt{climb}} = 1/en$. Hence, the expected time to climb is $O(n)$ when $p = 1/n$. Furthermore, we will reach 1^n before this climb happens n times. Every mutation followed by $O(n^2)$ steps of local search, and every $\lceil \log n \rceil$ number of mutations will have at least one mutation with $p = 1/n$. So the (1+1) AMA is expected to reach 1^n or T in $O(n^4 \log n)$ steps.

The probability of the dynamic mutation reaching $x' \in T$ from $x' = 1^{n_1}$ is bounded by $O(1)$ (proof can be found in [JW06]), thus the expected running time for the (1+1)

AMA on the 2-PTJ is bounded by $O(n^4 \log n)$. □

Theorem 25. *The expected running time of each static (1+1) MA on 2-PTJ is exponentially large.*

Proof. Since Jansen and Wegener [JW06] proved the expected running time of each static (1+1) EA on PTJ is exponential, we can see that the expected running time of each static (1+1) AMA without the local search on 2-PTJ is exponential.

Now we take the local search into account. For any point $a \in T$, the local search can help only if the mutation can flip to any point b with $\texttt{Hamming}(a, b) = 1$, then the local search can find this point a (the RPLS has a probability of $1/n$ to find the point a according to the sequence of the permutation, but we simply assume it can find). And for each point $a \in T$, we can have at most n points that the hamming distance to a is one. So with the help of the local search, we can increase an exponentially small probability by at most a factor of n, which is still exponentially small. So the expected running time to get T is exponentially large. □

Based on Theorems 24 and 25, we can see that for the function 2-PTJ, the (1+1) AMA is drastically faster than each static (1+1) MA. Next we will prove the opposite way.

3.4 Static (1+1) MA can drastically outperform (1+1) AMA

The function PTW is defined as:

$$\texttt{PTW}(x) = \begin{cases} 3n, & \text{if } x = 1^n, \\ n + i, & \text{if } x = 1^i 0^{n-i},\ i \neq n, \\ 2n, & \text{if } x \in T, \\ n - \texttt{ONEMAX}(x), & \text{otherwise,} \end{cases}$$

where T is the trap containing all x with k ones for $(1/4)n \leqslant k \leqslant (3/4)n$, such that the *Hamming distance* to some path $1^i 0^{n-i}$ is in the interval $[n/12, n/6]$, and the *Hamming distance* to each path point is at least $n/24$.

Jansen and Wegener [JW06] also proved the function PWT is polynomially solvable by the static (1+1) EA, but is exponential for the Dynamic (1+1) EA. Now we introduce a function 2-PTW, and show that the static (1+1) MA with mutation probability $p = 1/n$ can drastically outperform the (1+1) AMA on the function 2-PTW.

Definition 26. Let $x \in \{0,1\}^n$ be divided into two parts, $x = x'x''$, with $x' \in \{0,1\}^{n_1}$, $x'' \in \{0,1\}$ for $n_1 = n - 1$. The 2-PTW is defined as:

$$\texttt{2-PTW}(x) = \begin{cases} \texttt{ZZO}(x), & \text{if } x'' = 0. \\ 3n, & \text{else if } x' = 1^{n_1}. \\ n + 2i, & \text{else if } x' = 1^{2i} 0^{n_1 - 2i}, \\ & \quad \text{for } 2i < n_1. \\ 2n, & \text{else if } x' \in T. \\ -4n - \texttt{ONEMAX}(x''), & \text{otherwise.} \end{cases}$$

where T is the trap containing all x' with k ones for $(1/4)n_1 \leqslant k \leqslant (3/4)n_1$, such that the *Hamming distance* to some path $1^i0^{n_1-i}$ is in the interval $[n_1/12, n_1/6]$, and the *Hamming distance* to each path point is at least $n_1/24$.

Theorem 27. *The success probability of the (1+1) MA with mutation probability* $1/n$ *on* `2-PTW` *within* $O(n^4)$ *steps is exponentially close to 1.*

Proof Sketch. The proof is similar to the proof in Theorem 24. Firstly, the expected running time for the (1+1) MA to get on the path is $O(n^2)$. Secondly, once the (1+1) MA reaches the path, the expected running time for the (1+1) MA to reach $x' = 1^{n_1}$ on `2-PTW` is bounded by $O(n^4)$. This is because (a) the expected number of mutations to climb from $x' = 1^{2i}0^{n_1-2i}$ to $x' = 1^{2i+2}0^{n_1-2i-2}$ is $O(n)$, (b) each mutation followed $O(n^2)$ steps of local search, and (c) the algorithm will reach $x' = 1^{n_1}$ before n number of this climbing happens. Within this $O(n^4)$ time, we claim the probability of reaching $x' \in T$ before reaching $x' = 1^n$ is exponentially small. This is because to reach $x' \in T$ from any point on the path must flip at least $n_1/24$ number of bits. With the helps from the local search, the mutation still needs to flip $n_1/24 - 1$ number of bits, and this probability is exponentially small. Since the (1+1) MA is expected to reach $x' = 1^{n_1}$ within $O(n^4)$ steps, the probability of reaching $x' \in T$ before reaching $x' = 1^{n_1}$ is exponentially small. □

Theorem 28. *The probability that the (1+1) AMA needs an exponential number of steps on* `2-PWT` *is exponentially close to 1.*

Proof Sketch. First assume that we have $x' \in T$ for the current search point. Then the probability of reaching $x' = 1^{n_1}$ is exponentially small because we need the mutation to flip at least $n_1/4-1$ number of bits, and the success probability is $p^{n_1/4-1}$ is exponentially small for all mutation probabilities $p \leqslant 1/2$. Secondly, the probability of reaching $x' \in T$ before reaching $x' = 1^{n_1}$ is exponentially close to 1 (proof can be found in [JW06]). □

4 Case Study: Clique Problem

4.1 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 *Maximum 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 $f_{\texttt{MCP}}$ for the Maximum 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 Maximum Clique potential solution (an induced subgraph) where $x_i = 1$ represents that 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. $\{(v_i, v_j) \mid x_i = x_j = 1 \text{ and } i \neq$

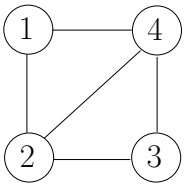
$j\} \subseteq E$.

Definition 29. The fitness function $f_{\texttt{MCP}}$ is defined as follows:

$$f_{\texttt{MCP}}(x) = \begin{cases} \texttt{ONEMAX}(x), & \text{if x represents a clique,} \\ -\texttt{LackEdges}(x), & \text{otherwise,} \end{cases}$$

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

Example 30. For a given graph G displayed below, $f_{\texttt{MCP}}(1101) = 3$ because $x = (1101)$ is a clique consists of vertices 1, 2 and 4. $f_{\texttt{MCP}}(1111) = -1$ because we need to add at least one edge $(1, 3)$.



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

4.2 New Metric on Stagnation Analysis

Now we analyze how these algorithms are coping with the stagnation when they are trapped into a local optimal clique. We first define a new metric to measure the difficulty of escaping out of a local optimal clique. Then we show how can this new metric dominates the time complexity of each analyzed algorithm to find a maximum clique, so as to show the usefulness of the new metric.

Definition 31. For a given clique $x = (x_1, x_2, \ldots, x_n) \in \{0, 1\}^n$ in graph G, function $\texttt{BLOCKONES}(x)$ is formalized as:

$$\texttt{BLOCKONES}(x) = \min_{(y_1, y_2, \ldots, y_n) \in Clique_{>x}} \left(\sum_{i=1}^{n} x_i \overline{y_i} \right),$$

where $Clique_{>x}$ is the set of all cliques in G with clique size greater than the clique size of x, i.e. $f_{\texttt{MCP}}(y) > f_{\texttt{MCP}}(x)$ for all $y \in Clique_{>x}$. Note, the complement of y_i is $\overline{y_i} = 1 - y_i$.

So $\texttt{BLOCKONES}(x)$ is the minimal number, such that at least $\texttt{BLOCKONES}(x)$ number of ones in x are blocking x to find a larger clique in G. So we have $\texttt{BLOCKONES}(x) = 0$ if the clique of x is a subset of a larger clique. Also $0 < \texttt{BLOCKONES}(x) < n/2$ if x is a local optimal clique.

Lemma 32. *If the analyzed algorithms are stagnated at a local optimal solution x, let $t := \texttt{BLOCKONES}(x)$, the expected running time to skip out of this local optimal solution and find a larger clique is bounded by*

1. *$O(n^{2t+1})$ for the (1+1) EA,*

2. $O(n^{2t+1}\log n)$ *for the Dynamic (1+1) EA,*

3. $O(n^{2t+2})$ *for the (1+1) MA,*

4. $O(n^{2t+2}\log n)$ *for the (1+1) AMA_RCLS, and*

5. $O(n^{2t+1}\log n)$ *for the (1+1) AMA_RPLS.*

Proof. Since $t := \texttt{BLOCKONES}(x)$, there must exist a larger clique y such that $f_{\texttt{MCP}}(y) = f_{\texttt{MCP}}(x) + 1$ and $\sum_{i=1}^{n} \overline{y_i} x_i = t$.

Part 1. The (1+1) EA and the Dynamic (1+1) EA both need to flip those blocking t bits in x from one to zero, and also flip another $t+1$ bits in x from zero to one to find this larger clique y. So the probability of the (1+1) EA and the Dynamic (1+1) EA to find this larger clique in one mutation is: $\texttt{Prob}_{\texttt{success}} = p^{2t+1}(1-p)^{n-2t-1} = \Omega\big(p^{2t+1}e^{-p(n-2t-1)}\big)$, where p is the mutation probability. So if $p = 1/n$, $\texttt{Prob}_{\texttt{success}} = \Omega\big(n^{-(2t+1)}\big)$, the (1+1) EA is expected to skip out of the local optimal x and find a larger clique in $O(n^{2t+1})$ steps. And for the Dynamic (1+1) EA, because the dynamic mutation probability has a $p = 1/n$ in every $\lceil \log n \rceil$ mutations, the upper bound of the Dynamic (1+1) EA is proved.

Part 2. Note that the (1+1) MA, the (1+1) AMA_RCLS, and the (1+1) AMA_RPLS both have a local search approach, so these algorithms can flip t bits from one to zero and flip another t bits from zero to one to get a new clique with the same clique size as x, and this new clique is also a sub-clique of y. Then the local search will flip at least one more bit to get a larger clique (note this larger clique may not be y). So the probability that the local search will find a larger clique after this mutation is one. And the probability of this mutation happens is: $\texttt{Prob}_{\texttt{success}} = p^{2t}(1-p)^{n-2t} = \Omega\big(p^{2t}e^{-p(n-2t)}\big)$.

So if $p = 1/n$, $\texttt{Prob}_{\texttt{success}} = \Omega(n^{-2t})$. And since the local search RPLS needs $O(n)$ steps after each mutation (see Algorithm 6), and RCLS needs $O(n^2)$ steps after each mutation (see Algorithm 5); also the (1+1) AMA_RPLS and the the (1+1) AMA_RCLS have at least one mutation with probability $p = 1/n$ in every $\lceil \log n \rceil$ mutations, the rest upper bounds are proved. □

We claim Lemma 32 is important because it indicates that if any of the analyzed algorithm has trapped into a local optimal solution for more than this number of times evaluating the fitness function, the probability that the algorithm can find a larger clique in the future is exponentially small. Note that due to Definition 31, $\texttt{BLOCKONES}(x) \leq \texttt{ONEMAX}(x)$. Thus each running algorithm knows an upper bound of t in Lemma 32. So Lemma 32 can indicate each running algorithm to stop with an overwhelming probability that a maximum clique has been found.

For example, if the (1+1) AMA_RPLS has found a clique of five nodes, then we know $\texttt{BLOCKONES}(x) \leq 5$. So according to Lemma 32, if we could not find a larger clique in the next $O(n^{2\cdot 5+1}\log n)$ fitness evaluations, we can claim this 5-clique is a maximum clique, and the probability of our claim to be false is exponentially small.

Definition 33. For a given graph G, the function MAXBLOCKONES (G) is formalized as:

$\texttt{MAXBLOCKONES}(G) = \max\{\texttt{BLOCKONES}(x) \mid x \text{ is a clique in G}\}$.

Theorem 34. *For a given graph G, let $t :=$ MAXBLOCKONES(G). The expected running time of the analyzed algorithms to find a maximum clique of G is bounded by*

1. $O(n^{2t+2})$ *for the (1+1) EA,*
2. $O(n^{2t+2} \log n)$ *for the Dynamic (1+1) EA,*
3. $O(n^{2t+3})$ *for the (1+1) MA,*
4. $O(n^{2t+3} \log n)$ *for the (1+1) AMA_RCLS, and*
5. $O(n^{2t+2} \log n)$ *for the (1+1) AMA_RPLS.*

Proof. Part 1. We prove the upper bounds of the (1+1) EA and the Dynamic (1+1) EA in two steps: (a) if the start string x does not represent a clique, then the expected running time of finding a clique is bounded by $O(n^2)$ and $O(n^2 \log n)$ respectively; and (b) if the start string x represents a clique, then the expected running time of finding a maximum clique is bounded by $O(n^{2t+2})$ and $O(n^{2t+2} \log n)$ respectively.

Step (a): if x does not represent a clique, the function $f_{\texttt{MCP}}$ will guide the algorithm to flip many ones to zeros to find a clique. The probability of the (1+1) EA and the Dynamic (1+1) EA to flip at least one bit in x from one to zero is $\texttt{Prob}_{\texttt{success}} = \Omega(p^1(1-p)^{n-1})$. And $\texttt{Prob}_{\texttt{success}} = \Omega(1/n)$ when $p = 1/n$. Note we have one mutation with $p = 1/n$ in every $\lceil \log n \rceil$ mutations for the Dynamic (1+1) EA. So we will expect to flip at least one bit in x from one to zero in $O(n)$ mutations on the (1+1) EA and $O(n \log n)$ mutations on the Dynamic (1+1) EA. Also, x has at most n bits of ones, so we will expect to find a clique in $O(n^2)$ mutations on the (1+1) EA and $O(n^2 \log n)$ mutations on the Dynamic (1+1) EA.

Step (b): Based on Lemma 32, the (1+1) EA and the Dynamic (1+1) EA can skip out of a local optimal clique and find a larger clique by $O(n^{2t+1})$ and $O(n^{2t+1} \log n)$ mutations respectively. Also, a maximum clique of G will be obtained before this skip is performed n times.

So the (1+1) EA and the Dynamic (1+1) EA is expected to find a maximum clique of G in $O(n^{2t+2})$ and $O(n^{2t+2} \log n)$ mutations (i.e. fitness evaluations) respectively.

Part 2. We prove the upper bounds of the (1+1) MA, the (1+1) AMA_RCLS and the (1+1) AMA_RPLS. The proof is similar to Part 1. We have stated that if the start string x does not represent a clique, the RPLS and the RCLS are expected to find a clique in $O(n)$ and $O(n^2)$ steps respectively (in Algorithm 6 and Algorithm 5). Also, from Lemma 32, the upper bounds for the three memetic based algorithms to skip out of a local optimal clique and find a larger clique is known. Since each time this skip will increase the clique size by at least one, a maximum clique of G will be obtained before this skip is performed n times. So the upper bounds of the (1+1) MA, the (1+1) AMA_RCLS and the (1+1) AMA_RPLS are proved. □

Corollary 35. *For a graph G, let $t :=$ MAXBLOCKONES(G). We have*

1. *If $t = \Theta(1)$, the analyzed algorithms are expected to find a maximum clique of G in a polynomial time.*

2. *If $t = \omega(1)$ and $t = o(n)$, the analyzed algorithms are expected to find a maximum clique of G in a sub-exponential time.*

3. *If any analyzed algorithm has took an exponential time to find a maximum clique on a graph G, this algorithm must have been trapped into at least one local optimal clique x with $\textit{BLOCKONES}(x) = \Theta(n)$.*

Theorem 34 and Corollary 35 show that

1. For all bipartite graphs and planar graphs, the analyzed algorithms are expected to find a maximum clique within a polynomial time. This is because the maximum clique size is three for planar graphs, and two for bipartite graphs.

2. For all sparse random graphs in the $G(n, c/n)$ model (e.g. edges between n nodes are connected with probability c/n, where $c > 0$ is a constant [Wit12]), the analyzed algorithms are expected to find a maximum clique within a polynomial time. Note a graph in this model is built by inserting all possible edges independently with probability c/n. Thus the expected vertex degree is $c(n-1)/n$ which is $\Theta(1)$. Hence it falls into the category of $t = \Theta(1)$ in Corollary 35.

4.3 Ability to avoid stagnation

From Section 4.2, we see that the ability of jumping out of a local optimal clique x with a large $\texttt{BLOCKONES}(x)$ is very important because it is the most time consuming part and dominates the time complexity of finding a maximum clique. This section we will analyze this ability and show that for any local optimal clique x with a very large $\texttt{BLOCKONES}(x)$, our proposed (1+1) AMA_RPLS algorithm is expected to take much less running time than the well-known (1+1) EA with a mutation probability $p = 1/n$ to jump out of x and find a better clique. First we define some formulas that will be used in this section.

Definition 36. To measure the probabilities of jumping from a local optimal clique x to another clique y with $f_{\texttt{MCP}}(y) \geq f_{\texttt{MCP}}(x)$. Let $U_1 := \{i \mid x_i = 1, y_i = 0 \text{ and } 1 \leq i \leq n\}$, and $U_2 := \{i \mid x_i = 0, y_i = 1 \text{ and } 1 \leq i \leq n\}$, then we have (a) $|U_2| \geq |U_1|$, and (b) $|U_1| \geq \texttt{BLOCKONES}(x)$ if $|U_2| > |U_1|$ (due to Definition 31).

Hence to jump from x to y, we need to flip all bits in U_1 from one to zero, and flip all bits in U_2 from zero to one. Now we define $\texttt{Prob}^{\texttt{EA}}_{(x\to y)}$, $K^U_{\texttt{RPLS}}$, $\texttt{Prob}^{+}_{(x\to y)}$ and $\texttt{Prob}^{-}_{(x\to y)}$ as below:

1. Let $\texttt{Prob}^{\texttt{EA}}_{(x\to y)}$ be the probability of the (1+1) EA jumping from x to y using one mutation. Then we have:

$$\texttt{Prob}^{\texttt{EA}}_{(x\to y)} = p^{|U_1|}p^{|U_2|}(1-p)^{n-|U_1|-|U_2|}.$$

2. For a set of bits U, let $K^U_{\texttt{RPLS}}$ be the probability that the RPLS will first check all bits in U according to the permutation array. So the permutation array will have all bits in U prior to the other $(n - |U|)$ bits. Then the probability of getting this type of permutation array is:

$$K^U_{\texttt{RPLS}} = \frac{|U|!(n-|U|)!}{n!} \geq \frac{1}{(n-|U|+1)^{|U|}}$$

3. Let $\texttt{Prob}^+_{(x\to y)}$ be the probability of the (1+1) AMA_RPLS jumping from x to y using one mutation with restriction that

   (a) The mutation flips all bits in U_1 from one to zero, and keeps $n - |U_1| - |U_2|$ bits not been flipped. Note none of these $n - |U_1| - |U_2|$ bits is in U_1 or U_2. Thus the mutation reaches a sub-clique of both x and y.

   (b) The local search RPLS first checks all bits in U_2 and flips them if they are not ones.

   Then we have:

$$\texttt{Prob}^+_{(x\to y)} = p^{|U_1|}(1-p)^{n-|U_1|-|U_2|}K^{U_2}_{\texttt{RPLS}},$$

   where $K^{U_2}_{\texttt{RPLS}}$ is the probability that the RPLS will first check all bits in U_2 according to the permutation array.

4. Let $\texttt{Prob}^-_{(x\to y)}$ be the probability of the (1+1) AMA_RPLS jumping from x to y using one mutation with restriction that

   (a) The mutation flips all bits in U_2 from zero to one, and keeps $n - |U_1| - |U_2|$ bits not been flipped. Note none of these $n - |U_1| - |U_2|$ bits is in U_1 or U_2 Thus the mutation reaches a bit string which does not represent a clique.

   (b) The local search RPLS first checks all bits in U_1 and flips them if they are not zeros, i.e. the RPLS reaches y by the $-\texttt{LackEdges}$ part in the fitness function (see Definition 29).

   Then we have:

$$\texttt{Prob}^-_{(x\to y)} = p^{|U_2|}(1-p)^{n-|U_1|-|U_2|}K^{U_1}_{\texttt{RPLS}},$$

   where $K^{U_1}_{\texttt{RPLS}}$ is the probability that the RPLS will first check all bits in U_1 according to the permutation array.

Theorem 37. *Let x and y be two cliques with $f_{MCP}(y) \geq f_{MCP}(x)$. Let $U_1 := \{i \mid x_i = 1, y_i = 0$ and $1 \leq i \leq n\}$. If $|U_1| = \Theta(n)$, then the expected running time of the (1+1) AMA_RPLS directly jumping from x to y is exponentially faster than the expected running time of the (1+1) EA with a mutation probability $p = 1/n$ directly jumping from x to y.*

Note "directly jumping" denotes the algorithm only use one mutation to reach the destination y. This is to distinguish with the algorithm using multiple mutations where each mutation jumps to another clique and finally reaches the destination y.

Proof. Let $U_2 := \{i \mid x_i = 0, y_i = 1 \text{ and } 1 \leq i \leq n\}$. Since $f_{\texttt{MCP}}(y) \geq f_{\texttt{MCP}}(x)$ and $|U_1| = \Theta(n)$, we have $|U_2| \geq |U_1| = \Theta(n)$.

Then the success probability of the (1+1) EA with $p = 1/n$ to find y in one mutation is: $\texttt{Prob}^{\texttt{EA}}_{x\to y} = p^{|U_1|}p^{|U_2|}(1-p)^{n-|U_1|-|U_2|} = (\frac{1}{n})^{|U_1|+|U_2|}(1-\frac{1}{n})^{n-|U_1|-|U_2|} = (\frac{1}{n})^{\Theta(n)}$.

The success probability of the (1+1) AMA_RPLS with $p = 1/2$ to find y in one mutation is: $\texttt{Prob}^{\texttt{AMA\_RPLS}}_{x\to y} > p^{|U_1|}p^{|U_2|}(1-p)^{n-|U_1|-|U_2|} = (\frac{1}{2})^{|U_1|+|U_2|}(1-\frac{1}{2})^{n-|U_1|-|U_2|} = (\frac{1}{2})^{n}$.

Thus the success probability of the (1+1) AMA_RPLS with $p = 1/2$ directly mutating from x to y is exponentially larger than the success probability of the (1+1) EA with $p = 1/n$ directly mutating from x to y.

Recall that the (1+1) AMA_RPLS has a dynamic mutation approach and a local search, so it will have at least one mutation with $p = 1/2$ in every $\log n$ mutations, and every mutation is followed by $O(n)$ steps of local search. But an exponential large number divided by a polynomial large number is still exponential. Thus the expected running time of the (1+1) AMA_RPLS directly jumping from x to y is still exponentially faster than the expected running time of the (1+1) EA directly jumping from x to y. □

Lemma 38. *In Definition 36, if $|U_1| = \omega(1)$, then the probabilities $\texttt{Prob}^{+}_{(x\to y)}$ and $\texttt{Prob}^{-}_{(x\to y)}$ with mutation $p = \Theta(|U_1|/n)$ are super-polynomially larger than the same probabilities with mutation $p = 1/n$ respectively. I.e. both ratios of $\frac{p_1^{|U_1|}(1-p_1)^{n-|U_1|-|U_2|}K_{RPLS}^{U_2}}{p_2^{|U_1|}(1-p_2)^{n-|U_1|-|U_2|}K_{RPLS}^{U_2}}$ and $\frac{p_1^{|U_2|}(1-p_1)^{n-|U_1|-|U_2|}K_{RPLS}^{U_1}}{p_2^{|U_2|}(1-p_2)^{n-|U_1|-|U_2|}K_{RPLS}^{U_1}}$ are super-polynomially large when $p_1 = \Theta(|U_1|/n)$ and $p_2 = 1/n$.*

Proof. Let $p_1 = \Theta(|U_1|/n)$ and $p_2 = 1/n$. Then for the probability $\texttt{Prob}^{+}_{(x\to y)}$, the ratio of p_1 and p_2 is:

$$\frac{p_1^{|U_1|}(1-p_1)^{n-|U_1|-|U_2|}}{p_2^{|U_1|}(1-p_2)^{n-|U_1|-|U_2|}} = \left(\frac{p_1}{p_2}\right)^{|U_1|}\left(\frac{1-p_1}{1-p_2}\right)^{n-|U_1|-|U_2|},$$

We have $\left(\frac{1-p_1}{1-p_2}\right)^{n-|U_1|-|U_2|} > (1-p_1)^{n-|U_1|-|U_2|} > (1-p_1)^n$ and $(1-p_1)^n = \Theta(e^{-|U_1|})$, since $p_1 = \Theta(|U_1|/n)$. And because $p_1 = \Theta(|U_1|/n)$, $p_2 = 1/n$, we have $\left(\frac{p_1}{p_2}\right)^{|U_1|} = (\omega(1))^{|U_1|}$. Thus this ratio is dominated by $\left(\frac{p_1}{p_2}\right)^{|U_1|}$. Also, because $|U_1| = \omega(1)$, this ratio is super-polynomially large.

Because $|U_2| \geq |U_1|$, the proof for the probability $\texttt{Prob}^{-}_{(x\to y)}$ is the same as above. □

For the following theorem, recall that "directly jumping" denotes the algorithm only use one mutation to reach the destination y.

Theorem 39. *Let x and y be two cliques with $f_{\texttt{MCP}}(y) \geq f_{\texttt{MCP}}(x)$. Let $U_1 := \{i \mid x_i = 1, y_i = 0 \text{ and } 1 \leq i \leq n\}$. If $|U_1| = \omega(1)$ and $|U_1| = o(n)$, then the expected running time of the (1+1) AMA_RPLS directly jumping from x to y is super-polynomially faster than the expected running time of the (1+1) EA with a mutation probability $p = 1/n$ directly jumping from x to y.*

Proof. Let $U_2 := \{i \mid x_i = 0, y_i = 1 \text{ and } 1 \leq i \leq n\}$. Then we have $|U_2| \geq |U_1| = \omega(1)$ since $f_{\texttt{MCP}}(y) \geq f_{\texttt{MCP}}(x)$ and $|U_1| = \omega(1)$. According to Definition 36, the ratio of $\texttt{Prob}^{+}_{(x\to y)}$ and $\texttt{Prob}^{\texttt{EA}}_{(x\to y)}$ is:

$$\frac{\texttt{Prob}^{+}_{(x\to y)}}{\texttt{Prob}^{\texttt{EA}}_{(x\to y)}} = \frac{1}{(n-|U_2|+1)^{|U_2|}}\frac{1}{p^{|U_2|}} > 1.$$

We have this ratio is greater than one when $p = 1/n$. Note this is the probability ratio of the (1+1) AMA_RPLS with mutation probability $p = 1/n$ and the (1+1) EA with mutation probability $p = 1/n$.

In the (1+1) AMA_RPLS, the dynamic mutation approach obtains a mutation probability between $|U_1|/n$ and $2|U_1|/n$ in every $\log n$ mutations unless it finds a larger clique earlier. And according to Lemma 38, since $|U_1| = \omega(1)$, the ratio of the $\texttt{Prob}^{+}_{(x\to y)}$ with a mutation probability between $|U_1|/n$ and $2|U_1|/n$ and the $\texttt{Prob}^{+}_{(x\to y)}$ with a mutation probability $p = 1/n$ is super-polynomially large.

So the probability of the (1+1) AMA_RPLS directly jumping from x to y is super-polynomially larger than the probability of the (1+1) EA directly jumping from x to y. □

Theorem 40. *For a local optimal clique x with $t :=$ BLOCKONES (x), we have if $t = \Theta(n)$, the (1+1) AMA_RPLS is expected to skip out of x and find a larger clique super-polynomially faster than the (1+1) EA with a mutation probability $p = 1/n$.*

Proof. To prove the theorem, we just need to show that if both algorithms are trapped into x, then for any clique y such that y is the first clique the algorithms have found with $f_{\texttt{MCP}}(y) > f_{\texttt{MCP}}(x)$ (escape from x), the probability of the (1+1) AMA_RPLS jumping from x to y is super-polynomially larger than the probability of the (1+1) EA with $p = 1/n$ jumping from x to y.

Moreover, since the (1+1) EA only accepts a new solution if its fitness is greater or equal to the current solution, it can only escape from clique x to clique y by two ways: (a) directly mutating from x to y; or (b) mutating multiple times to different cliques with the same clique size as x, and then mutating to the clique y.

Proof of case (a), by directly mutating from x to y.
Let $U_1 := \{i \mid x_i = 1, y_i = 0 \text{ and } 1 \leq i \leq n\}$. Since $f_{\texttt{MCP}}(y) > f_{\texttt{MCP}}(x)$, we have $|U_1| \geq t = \Theta(n)$.

Because $|U_1| = \Theta(n)$, from Theorem 37 we know the probability ratio of the (1+1) AMA_RPLS directly jumping from x to y and the (1+1) EA directly jumping from x to y is exponentially large.

Proof of case (b), by mutating multiple times to different cliques with the same clique size as x, and then mutating to the clique y.

Let $\texttt{Path}_\lambda$ be an arbitrary $(\lambda+1)$-length path $x^0 \to x^1 \to x^2 \cdots x^\lambda \to y$, where $x^0 = x$, $\lambda \geq 0$ and each x^i is a bit string with $f_{\texttt{MCP}}(x) = f_{\texttt{MCP}}(x^i)$. We proof this part by proving that the probability of the (1+1) AMA_RPLS jumping along this $\texttt{Path}_\lambda$ to reach y is super-polynomially larger than the probability of the (1+1) EA jumping along this $\texttt{Path}_\lambda$ to reach y when $t = \Theta(n)$. The proof contains four steps.

Step 1. For any jump from x^i to x^{i+1} $(0 \leq i \leq \lambda-1)$ in $\texttt{Path}_\lambda$, let x^i_j denote the j-th bit of the bit string x^i, and let $\mathcal{U}_i = \{j \mid x^i_j = 1, x^{i+1}_j = 0 \text{ and } 1 \leq j \leq n\}$. So we will flip $|\mathcal{U}_i|$

number of bits from one to zero and flip another $|\mathcal{U}_i|$ number of bits from zero to one. So according to Definition 36, we know $\texttt{Prob}^{-}_{(x^i \to x^{i+1})} = \texttt{Prob}^{+}_{(x^i \to x^{i+1})} > \texttt{Prob}^{\texttt{EA}}_{(x^i \to x^{i+1})}$. Thus we have:

$$\begin{aligned} \frac{\texttt{Prob}^{\texttt{AMA\_RPLS}}_{(x^i \to x^{i+1})}}{\texttt{Prob}^{\texttt{EA}}_{(x^i \to x^{i+1})}} &> \frac{\texttt{Prob}^{+}_{(x^i \to x^{i+1})} + \texttt{Prob}^{-}_{(x^i \to x^{i+1})}}{\texttt{Prob}^{\texttt{EA}}_{(x^i \to x^{i+1})}} \\ &= \frac{2}{(n - |\mathcal{U}_i| + 1)^{|\mathcal{U}_i|}} \frac{1}{p^{|\mathcal{U}_i|}} > 2. \end{aligned} \tag{1}$$

Furthermore, this ratio is exponentially large if $|\mathcal{U}_i| = \Theta(n)$ (in Theorem 37), or this ratio is super-polynomially large if $|\mathcal{U}_i| = \omega(1)$ and $|\mathcal{U}_i| = o(n)$ (in Theorem 39).

Step 2. Also according to Definition 36, for the last jump from x^λ to y, we have:

$$\frac{\texttt{Prob}^{\texttt{AMA\_RPLS}}_{(x^\lambda \to y)}}{\texttt{Prob}^{\texttt{EA}}_{(x^\lambda \to y)}} > \frac{\texttt{Prob}^{+}_{(x^\lambda \to y)}}{\texttt{Prob}^{\texttt{EA}}_{(x^\lambda \to y)}} > 1,$$

and this ratio is exponentially large if this jump needs to flip $\Theta(n)$ bits (in Theorem 37), or this ratio is super-polynomially large if the number of bits flipped is between $\omega(1)$ and $o(n)$ (in Theorem 39).

Step 3. Let $\texttt{Prob}^{\texttt{AMA\_RPLS}}_{\texttt{success}}$ and $\texttt{Prob}^{\texttt{EA}}_{\texttt{success}}$ be the probabilities of the (1+1) AMA_RPLS and the (1+1) EA with $p = 1/n$ jumping along this itinerary to reach y, respectively. We have:

$$\frac{\texttt{Prob}^{\texttt{AMA\_RPLS}}_{\texttt{success}}}{\texttt{Prob}^{\texttt{EA}}_{\texttt{success}}} = \prod_{i=0}^{\lambda-1} \frac{\texttt{Prob}^{\texttt{AMA\_RPLS}}_{(x^i \to x^{i+1})}}{\texttt{Prob}^{\texttt{EA}}_{(x^i \to x^{i+1})}} \frac{\texttt{Prob}^{\texttt{AMA\_RPLS}}_{(x^\lambda \to y)}}{\texttt{Prob}^{\texttt{EA}}_{(x^\lambda \to y)}}.$$

Step 4. Recall that $t = \texttt{BLOCKONES}(x)$, and $f_{\texttt{MCP}}(y) > f_{\texttt{MCP}}(x)$, thus to jump along this itinerary from x to y, we will finally flip at least t bits from one to zero. And since $t = \Theta(n)$, to achieve our final goal of finding y, we have three ways:

1. at least one jump (either belonging to Step 1 or Step 2) needs to flip $\Theta(t)$ bits, and the ratio of this jump is exponentially large (Theorem 37), while the ratios of other jumps are all greater than one, thus the overall ratio in Step 3 is exponentially large; or

2. have at least $\omega(1)$ number of jumps, which are belonging to Step 1, where each jump needs to flip $\omega(1)$ bits. Since the ratio of each jump with flipping $\omega(1)$ bits is super-polynomially large, thus the overall ratio in Step 3 is super-polynomially large; or

3. have $\Omega(t)$ number of jumps, which are belonging to Step 1, where each jump only needs to flip $\Theta(1)$ bits. Since the ratio of these small jumps is greater than two (in Step 1), the overall ratio in Step 3 is still exponentially large.

□

4.4 Experimental Results

In this section we will test our analyzed algorithms on the Maximum Clique Problem. To avoid only comparing algorithms that are analyzed in theory, we also bring a state-of-art Spacing Memetic Algorithm (SMA) [PHK11] into comparison.

In short, the SMA first keeps the minimum distance between each two individuals above a threshold, and then try to maximize the average distance among the population individuals. Also, it uses an elitist selection approach based on both distance and fitness. Thus, it follows the principle "diversity without quality sacrifices".

Table 2 reports maximum clique results on some DIMACS instances [JT96]. Column 1 depicts the graph names with their best known clique size in the parentheses. We test the (1+1) EA [DJW02] in column 2, the Dynamic (1+1) EA [JW06] in column 3, the (1+1) MA [Sud06] in column 4, our (1+1) AMA_RCLS [DW13] in column 5, our (1+1) AMA_RPLS [DW13] in column 6 and the SMA [PHK11] in column 7. Each algorithm is run on each graph 10 times where each run is limited to one minute of running time on a linux machine with 2.5GHz Intel CPU. The sub-column "Best" is the best clique found in 10 runs, and the sub-column "Avg" is the average clique size in 10 runs. The sub-column "Gen" represents the average generations, i.e. average number of iterations.

The "Best" entry is grey-colored if it finds the best known clique of that graph. The "Avg" entry is grey-colored if it has the best average result.

Note we restrict the running time to one minute because it is enough for our proposed (1+1) AMA_RPLS to find a maximum clique in all 10 runs on those small order graphs (such as `brock200_2`, `C125.9`, `gen200_p0.9_55`, etc.), but not all tested algorithms can achieve this within one minute. Meanwhile, even though no algorithm can find a maximum clique on some large order graphs (`C4000.5`, `p_hat1500_3`, etc.) in one minute, our proposed (1+1) AMA_RPLS still outperforms other tested algorithms in terms of the "best" and "Avg" performance.

From Table 2, we claim the following:

1. All tested algorithms have found a global optimum in some small order graphs such as `C125.9` and `keller4`. This means that each algorithm found the global optimum if it has enough time.

2. The (1+1) AMA_RPLS outperforms the other five algorithms in most graphs in terms of getting the best "Avg" results. Meanwhile, the "Best" results of the (1+1) AMA_RPLS are greater or equal to the "Best" results in other three algorithms. This denotes that the (1+1) AMA_RPLS has excellent stability.

3. Apart from the population-based SMA, each iteration of the (1+1) EA uses the least running time while each iteration of the (1+1) AMA_RCLS uses the most running time, i.e. the (1+1) EA has the largest value in the "Gen" sub-column for each graph, while the (1+1) AMA_RCLS has the smallest value. This denotes that the local search approaches, especially the RCLS, are very time consuming.

4. The (1+1) AMA_RPLS is the most efficient algorithm that can quickly detect a clique. This can be observed from some large order graphs such as `C4000.5`. We claim this is important because there are many real-world problems that do

Table 2: A comparison of the algorithms on the Maximum Clique Problem for one minute of CPU time (k=10^3, m=10^6).

	(1+1) EA			Dynamic (1+1) EA			(1+1) MA			(1+1) AMA_RCLS			(1+1) AMA_RPLS			(1+1) SMA		
Metrics	Best	Avg	Gen	Best	Avg	Gen	Best	Avg	Gen	Best	Avg	Gen	Best	Avg	Gen	Best	Avg	Gen
brock200_2 (=12)	11	9.8	15.1m	10	9.5	5.4m	11	10.1	170.0k	11	10.9	1.6k	12	12	59.8k	12	11	2.1k
brock200_4 (=17)	16	14.6	14.5m	15	14	5.5m	16	14.9	139.5k	16	15.8	1.7k	17	16.8	57.3k	16	16	2.2k
brock400_2 (=29)	23	22.3	7.6m	22	20.6	2.2m	24	22.9	37.8k	24	22.7	170.0	25	24	11.7k	23	21.9	21.7
brock400_4 (=33)	22	21.2	7.7m	22	19.5	2.2m	24	22.4	38.3k	24	22.7	169.7	24	23.3	11.9k	22	20.9	6.2
brock800_2 (=24)	18	16.5	4.2m	17	15.1	830.4k	18	16.6	3.4k	17	15.4	11.1	19	18.6	2.3k	16	15	1.0
brock800_4 (=26)	15	14.1	4.3m	14	12.4	1.3m	16	15.1	8.5k	16	13.8	16.1	16	16	3.9k	15	13.5	1.5
C125.9 (≥34)	34	33.5	16.2m	34	33.8	10.8m	34	33.5	143.5k	34	34	13.1k	34	34	115.3k	34	34	3.4k
C250.9 (≥44)	44	41.9	9.5m	42	41.2	5.1m	44	42.2	45.5k	44	42.5	1.3k	44	43.4	30.9k	44	42.5	622.2
C500.9 (≥57)	48	44.8	5.6m	45	41.1	2.5m	49	46.4	15.5k	46	45.1	141.4	49	47.9	9.3k	47	45.4	70.3
C1000.9 (≥68)	52	50.2	3.1m	49	45.4	919.7k	53	50.4	890.8	52	49.2	17.6	59	56.2	1.8k	50	46.4	1.1
C2000.5 (≥16)	10	9.5	1.8m	10	9.1	384.7k	fail	fail	0	fail	fail	0	13	11.5	487.5	fail	fail	0
C2000.9 (≥77)	46	41.3	1.7m	42	39	423.6k	fail	fail	0	fail	fail	0	52	49.5	492.8	fail	fail	0
C4000.5 (≥18)	12	9.8	710.1k	fail	fail	70.7k	fail	fail	0	fail	fail	0	13	11.7	81.7	fail	fail	0
DSJC500.5 (≥13)	12	11.3	6.6m	12	10.1	1.5m	12	11.3	31.9k	11	10.9	59.9	13	12.5	6.7k	12	10.8	2.0
DSJC1000.5 (≥15)	12	11.3	3.4m	11	10.3	492.3k	fail	fail	0	fail	fail	0	13	13	1.1	fail	fail	0
gen200_p0.9_44 (=44)	44	40	11.1m	44	39	6.3m	44	39.9	64.8k	44	40.7	2.6k	44	39.9	47.1k	44	39.4	1.1k
gen200_p0.9_55 (=55)	55	46.2	10.5m	55	44.1	6.2m	55	44.7	58.2k	55	55	2.3k	55	55	36.9k	55	50	879.1
gen400_p0.9_55 (=55)	49	47.6	6.5m	47	44.5	2.7m	51	49.1	20.6k	50	48.4	270.8	52	50.2	11.5k	49	47.9	180.0
gen400_p0.9_65 (=65)	48	46.3	6.6m	45	43.7	2.7m	55	48.7	21.1k	49	46	253.7	55	49.3	12.0k	49	46.4	184.7
gen400_p0.9_75 (=75)	55	49.7	6.4m	56	48.8	2.7m	75	65.3	16.1k	75	64.8	287.0	75	65.7	9.9k	75	55.3	161.9
hamming8-4 (=16)	16	13.2	11.6m	16	13.2	4.4m	16	16	90.7k	16	16	850.8	16	16	38.3k	16	16	885
hamming10-4 (=40)	35	33.5	3.2m	30	27.7	720.7k	32	8 fails	20	32	9 fails	2	40	36.3	1.5k	fail	fail	0
keller4 (=11)	11	10.8	17.0m	11	10.3	7.0m	11	11	209.1k	11	11	3.2k	11	11	94.0k	11	11	4.0k
keller5 (=27)	18	17.0	4.4m	18	16.2	1.4m	19	17.8	10.0k	17	16.8	24.1	19	18.9	4.0k	17	16.0	1.2
keller6 (≥59)	26	22.1	933.9k	fail	fail	1.3m	fail	fail	0	fail	fail	0	28	26.4	120.7	fail	fail	0
MANN_a27 (=126)	124	122.4	3.5m	123	120.6	2.7m	125	124.3	6.8k	124	122.8	354.1	126	125.3	5.7k	125	121.9	680.3
MANN_a45 (=345)	331	330.5	673.1k	331	330.1	502k	331	330.4	196.6	332	330.6	10.5	342	339	142.8	331	330.4	1.5
MANN_a81 (≥1100)	364	360.7	403.3k	fail	fail	135.1k	fail	fail	0	fail	fail	0	365	364.6	65.4	fail	fail	0
p_hat300-1 (=8)	8	7.2	10.6m	8	6.8	3.2m	8	7.6	97.3k	8	8	336.8	8	8	25.0k	8	7.8	468.0
p_hat300-2 (=25)	25	24.1	9.5m	25	23.9	2.6m	25	24.6	54.7k	25	25	320.8	25	25	18.1k	25	24.9	373.6
p_hat300-3 (=36)	36	34.5	8.8m	34	32.1	2.9m	36	35.3	41.7k	36	35.4	413.2	36	35.5	18.0k	36	35.4	409.3
p_hat700-1 (=11)	8	7.7	4.9m	8	7.1	951.2k	9	8.5	12.3k	9	8.3	14.6	11	9.8	3.0k	11	8.3	1.5
p_hat700-2 (≥44)	37	35.0	4.5m	30	26	951.0k	38	37.1	6.9k	37	34.8	22.5	38	37.7	2.8k	35	31.9	1.5
p_hat700-3 (≥62)	43	41.0	4.4m	40	33.8	1.4m	43	42.3	8.2k	42	40.8	37.4	43	42.9	4.3k	42	37.4	1.6
p_hat1500-1 (≥12)	9	8.0	2.3m	8	6.9	376.4k	fail	fail	0	fail	fail	0	11	9.4	581.8	fail	fail	0
p_hat1500-2 (≥65)	52	46.2	2.1m	54	44.4	213.9k	fail	fail	0	fail	fail	0	65	62.2	323.8	fail	fail	0
p_hat1500-3 (≥94)	73	67.8	2.0m	63	57.7	269.3k	fail	fail	0	fail	fail	0	88	85.3	390.8	fail	fail	0

not require the global optimal solutions, but they have strict requirements on the running time.

5 Conclusions and Future Work

We have defined the (1+1) AMA with an adaptive mutation probability and two different local searches (RCLS and RPLS). The time complexity analysis proved that the (1+1) AMA with RPLS outperforms the other three counterparts on `arbitrary functions`, `ONEMAX`, `BIN`, `linear functions` and `LEADINGONES`.

Besides, we defined two classes of functions ($T_{\texttt{RPLS}}$ and $T_{\texttt{RCLS}}$), and have proved that two local searches can outperform each other on different functions.

Also, we defined one class of function (`2-PTJ`), and have proved that the (1+1) AMA can drastically outperform each static (1+1) MAs; while defined another class of function (`2-PTW`), and have proved that a static (1+1) MA can drastically outperform the (1+1) AMA.

A next step to investigate in the future is to analyze and test the (1+1) AMA on some NP-hard problems. Also, compare the two local searches (RCLS and RPLS) with other local searches such as Tabu Search.

6 Acknowledgements

We wish to thank Ralph Versteegen for helpful discussions that improved the content of this paper.

References

[BS04] 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*, pages 289–312. Springer, 2004.

[Dav91] L. Davis. Bit-climbing, representational bias, and test suite design. In *Proceedings of Fourth International Conference on Genetic Algorithms*, pages 18–23, 1991.

[DJW98] 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*, pages 13–22. Springer-Verlag, London, UK, 1998.

[DJW02] S. Droste, T. Jansen, and I. Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276:51–81, 2002.

[DLW11] 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)*, volume 1, pages 140–144, 2011.

[DNO11] G. Durrett, F. Neumann, and U. M. O'Reilly. Computational complexity analysis of simple genetic programming on two problems modeling isolated program semantics. In *Proceedings of the 11th workshop proceedings on Foundations of Genetic Algorithms. ACM, New York, NY, USA*, pages 69–80, 2011.

[DW13] M. J. Dinneen and K. Wei. On the analysis of a (1+1) adaptive memetic algorithm. In *Proceedings of Memetic Computing, MC2013*, pages 24–31. IEEE, 2013.

[GW03] 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*, pages 415–426. Springer-Verlag, London, UK, 2003.

[JJW05] T. Jansen, K. A. D. Jong, and I. Wegner. On the choice of the offspring population size in evolutionary algorithms. *Evol. Comput.*, 13(4):413–440, 2005.

[JT96] D. Johnson and M. Trick. Cliques, coloring and satisfiability second DIMACS implementation challenge, volume 26 of DIMACS series in Discrete Mathematics and Theoretical Computer Science, 1996.

[JW01] 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*, 5(6):589–599, 2001.

[JW02] T. Jansen and I. Wegener. On the analysis of evolutionary algorithms—a proof that crossover really can help. *Algorithmica*, 34(1):47–66, 2002.

[JW06] T. Jansen and I. Wegener. On the analysis of a dynamic evolutionary algorithm. *Journal of Discrete Algorithms*, 4(1):181–199, 2006.

[KST11] T. Kotzing, D. Sudholt, and M. Theile. How crossover helps in pseudo-boolean optimization. In Natalio Krasnogor, editor, *Proceedings of the 13th annual conference on Genetic and Evolutionary Computation (GECCO '11)*, pages 989–996. ACM, New York, NY, USA, 2011.

[NCM12] F. Neri, C. Cotta, and P. Moscato. *Handbook of Memetic Algorithms*, volume 379. Studies in Computational Intelligence, Springer, 2012.

[NW04] 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), pages 713–724, 2004. LNCS 3102. Springer.

[OHY07a] P. S. Oliveto, J. He, and X. Yao. Evolutionary algorithms and the vertex cover problem. In *Proc. CEC, Singapore*, pages 1870–1877. IEEE, 2007.

[OHY07b] 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*, 4(3):281–293, 2007.

[OHY08] P. S. Oliveto, J. He, and X. Yao. Analysis of population-based evolutionary algorithms for the vertex cover problem. In *Proc. CEC, Hong Kong, China*, pages 1563–1570. IEEE, 2008.

[PHK11] D. C. Porumbel, J. K. Hao, and P. Kuntz. Spacing memetic algorithms. In Natalio Krasnogor, editor, *Proceedings of the 13th annual conference on Genetic and Evolutionary Computation (GECCO '11), ACM, New York, NY, USA*, pages 1061–1068, 2011.

[QYZ11] C. Qian, Y. Yu, and Z. H. Zhou. An analysis on recombination in multi-objective evolutionary optimization. In Natalio Krasnogor, editor, *Proceedings of the 13th annual conference on Genetic and Evolutionary Computation (GECCO '11)*, pages 2051–2058. ACM, New York, NY, USA, 2011.

[Rud98] G. Rudolph. Finite Markov chain results in evolutionary computation: A Tour D'horizon. *Fundamenta Informaticae*, 35(1–4):67–89, 1998.

[Sto06] T. Storch. How randomized search heuristics find maximum clique in planar graphs. In *Proceedings of the 8th annual conference on Genetic and Evolutionary Computation (GECCO '06)*, pages 567–574. ACM, New York, NY, USA, 2006.

[Sud06] D. Sudholt. On the analysis of the (1+1) memetic algorithm. In *Proceedings of the 8th annual conference on Genetic and Evolutionary Computation*, pages 493–500, 2006.

[Sud09] D. Sudholt. The impact of parametrization in memetic evolutionary algorithms. *Theoretical Computer Science*, 410(26):2511–2528, 2009.

[Sud11] D. Sudholt. Hybridizing evolutionary algorithms with variable-depth search to overcome local optima. *Algorithmica*, 59(3):343–368, 2011.

[SZ10] 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)*, volume 6506 of *LNCS*, pages 340–352. Springer, 2010.

[Wit05] 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)*, volume 3804 of *LNCS*, pages 44–56. Springer, 2005.

[Wit06] C. Witt. 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*, pages 651–658, 2006.

[Wit12] C. Witt. Analysis of an iterated local search algorithm for vertex cover in sparse random graphs. *Theoretical Computer Science*, 425(1):417–425, 2012.