CDMTCS
Research
Report
Series

On the Analysis of a (1+1) Self-Adjusting Memetic Algorithm

Michael J. Dinneen
Kuai Wei

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

CDMTCS-424
August 2012

Centre for Discrete Mathematics and
Theoretical Computer Science

On the Analysis of a (1+1) Self-Adjusting Memetic Algorithm

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. This paper defines the *(1+1) self-adjusting memetic algorithm* with a dynamic mutation probability, and proposes a random permutation local search to compare with a traditional random complete local search. Our time complexity analysis proves that these two local searches can outperform each other on different functions. Also memetic algorithms with dynamic mutation probabilities can outperform memetic algorithms with static mutation probabilities, and vice versa.

The success of our experimental results on the Maximum Clique Problem shows the benefit of the self-adjusting strategy combined with the random permutation local search. Also focusing on the Maximum Clique Problem, we propose another time complexity metric (expected time to escape a local optimal) and show how it bounds the expected running time of finding a maximum clique. This metric provides insight and shows some advantages of our approach of using a dynamic mutation probability and a random permutation local search.

1 Introduction

A Memetic Algorithm (MA) [7] 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 [3]. 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 [4].

The experimental studies of MAs have grown rapidly, but theoretical studies have not kept up with the state-of-the-art of MAs. In 2002, Droste, Jansen and Wegener [5] analyzed a (1+1) EA with only a mutation approach but no crossover. The term (1+1) represents that the population size of parents and children are both one. The authors

of [6] show the usefulness of a dynamic mutation approach for EAs and in this paper we claim that dynamic mutation is also crucial in MAs. In 2006, a theoretical analysis of a (1+1) MA from [2] 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.

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. However, these papers 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.

This paper will define a (1+1) Self-Adjusting Memetic Algorithm (SAMA) to analyze the dynamic mutation probability and two different local search approaches. Our study contains both theoretical and experimental results.

The paper is structured as follows. In Section 2, we formalize a (1+1) SAMA and two different local search approaches. In Section 3, we first analyze the running time of the (1+1) SAMA 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) SAMA can drastically outperform each static (1+1) MAs; while on some other functions, a static (1+1) MA can drastically outperform the (1+1) SAMA. In Section 4, experimental results provides a running time comparison among the Dynamic (1+1) EA [6], (1+1) MA [2], and the (1+1) SAMA on the Maximum Clique Problem, followed by a running-time analysis from a new complexity metric based on the expected running time to escape a local optimal point. Our conclusions and future work will be given in Section 5.

2 Algorithm Definitions

The (1+1) Self-Adjusting Memetic Algorithm (SAMA) tries 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 f.

The term "self-adjusting" 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) SAMA contains a mutation operation with a dynamic mutation probability, and a local search operation. A template of the generic algorithm is follows:

Algorithm 1. (1+1) SAMA 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. $p := \texttt{SelfAdjusting}(f(x), f(z), p)$.

6. If $f(z) \geqslant f(x)$ then $x := z$.

7. Stop if meet some stopping criterion; 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 `SelfAdjusting` and `LocalSearch` will be stated in the subsections below.

2.1 Self-adjusting mutation probability

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

The mutation in SAMA is mainly used for making a jump in the search space when SAMA stagnates into a local optimal solution. Meanwhile, the mutation probability $p = 1/n$ means that the expected number of flipped bits is 1. So the self-adjusting 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 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 Local search

As stated before, the local search in step 4 of Algorithm 1 can have many variations. A *Random Complete Local Search* (RCLS) is used quite often in MAs such as the (1+1) MA in [2].

Algorithm 2. Random Complete Local Search (RCLS).

For a given string $x = (x_1, x_2, \cdots, x_n) \in \{0,1\}^n$:

1. $t := 1$.
2. `BestNeighborSet` := $\{\}$, `BestFitFound` := $f(x)$.
3. For $i = 1$ to n, do
4. Let $y :=$ `flip`(x, i).
5. If $f(y) >$ `BestFitFound` then:
6. `BestNeighborSet` := $\{y\}$, `BestFitFound` := $f(y)$.
7. Else if $f(y) =$ `BestFitFound` then:
8. `BestNeighborSet` := `BestNeighborSet` $\bigcup\{y\}$
9. End if.
10. End For.
11. Stop if meet the stopping criterion.
12. Otherwise, $x :=$ randomly choose from `BestNeighborSet`, $t := t + 1$, go to step 2.

Here `flip`(x, i) denotes that the i-th bit in x is flipped. The RCLS performs n fitness evaluations per step. Hence, the stop criterion in step 11 is when $t = O(n)$ or the set `BestNeighborSet` is empty.

In this paper, we propose a *Random Permutation Local Search* (RPLS). 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 3. 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 array `Per` of length n.
2. $t := 1$, $i := 1$, `WorseNeighbor` := 0.
3. $y :=$ `flip`$(x,$ `Per`$[i])$.

4. If $f(y) > f(x)$ then $x := y$ and $\texttt{WorseNeighbor} := 0$.

5. $t := t + 1$.

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

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

8. Stop if meet the stopping criterion. 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 Per.

Example 4. Suppose 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 Per in a cyclic fashion. That is, after checking the fourth bit in x, the PRLS will restart checking the third bit in x.

Note that the RPLS uses only one fitness evaluation per step, so the RPLS will stop when $t = O(n^2)$ or there is no neighbor solution which has a better fitness value, i.e. $\texttt{WorseNeighbor} = n$.

Therefore, the expected running time for the local search in step 4 of Algorithm 1 is $O(n^2)$. Dirk Sudholt [2] 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 a upper bound of $O(n^2)$ for the above two local searches.

3 Algorithm Analysis and Comparisons

In this section, we will analyze the expected running time of the (1+1) SAMA with the above two local search approaches (RCLS and RPLS of Section 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) SAMA can drastically outperform each static (1+1) MAs; while on some other functions, a static (1+1) MA can drastically outperform the (1+1) SAMA.

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 [5] and [2]:

Definition 5. 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 6. 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 7. 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 8. 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 9. 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 10. 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 (zeros, zeros, ones) is defined as:

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

Note that 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) SAMA

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

Lemma 11. *The expected number of steps the mutation with probability $p = 1/2$ takes to optimize an arbitrary bit string to a global optimum is 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 2^n. □

Theorem 12. *The expected running time of the (1+1) SAMA-RPLS and the (1+1) SAMA-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. 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 11, the upper bound of the (1+1) SAMA-RPLS and the (1+1) SAMA-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 12 still holds when we use the positive feedback to decrease the mutation probability. □

Lemma 13. *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 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`. And if `Per` $= (n, n-1, \ldots, 1)$, the RPLS needs exactly n^2 steps to find the optimum solution. 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 14. *The following bounds hold for the expected running time of the (1+1) SAMA-RPLS and the (1+1) SAMA-RCLS on the following functions:*

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

Proof. Recall that the local search approach in the (1+1) SAMA takes up to $O(n^2)$ steps (see Section 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) SAMA with that local search approach by the same lower bound and upper bound. Therefore, based on the Lemma 13, the Theorem 14 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 [5] and [6], respectively.

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 15. 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

$$T_{\texttt{RPLS}}(x) = \begin{cases} \texttt{ZZO}(x), & \text{if } x'' = 0. \\ n + 3i + 4k, & \text{else if } x' = 1^i0^j1^k0^{n_1-j-k}, \\ & \quad \text{for } i \geqslant 0, j \geqslant 1, k \geqslant 0. \\ -\texttt{ONEMAX}(x'') - 4n, & \text{otherwise.} \end{cases}$$

where ZZO is defined in Definition 10. We can see the global optimum string is 01^{n-1}. We say x is on the path if $x' = 1^i0^j1^k0^{n_1-j-k}$ for $i \geqslant 0, j \geqslant 1, k \geqslant 0$.

Theorem 16. *Both the RPLS and the RCLS will stop on the function T_{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. 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. 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. Thus the probability of the RCLS to find the global optimum is exponentially large.

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 $x' = 0^j 1^k 0^{n_1-j-k}$, 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''. So the probability that a random permutation can let the RPLS successfully find the global optimum solution is

$$\texttt{Prob}_{\texttt{success}} = \frac{\sum_{i=2}^{n_1} 2^{\min(n_1-i,i)}}{n_1!},$$

which is factorially small. Also if the initial bit string is on the path (the probability of this happens is exponentially small), the RPLS also has the chances to reach the global optimum. Thus the probability of the RPLS to reach the global optimum 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 17. 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

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

where ZZO is defined in Definition 10. We can see the global optimum string is $1^i 0 1^{n_1-i-1}$ for $i \geqslant 2$. We say x is on the path if $x' = 1^i 0^j 1^k 0^{n_1-j-k}$ for $i \geqslant 0, j \geqslant 1, k \geqslant 0$.

Theorem 18. *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 16. 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 16 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 self-adjusting mutation probability. We compare the expected running time of the (1+1) SAMA against the static (1+1) MAs.

3.3 (1+1) SAMA 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 [6]. 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 [6] 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) SAMA 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 19. 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

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

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) SAMA has to rely on both the mutation and the local search to climb along the path.

Theorem 20. *The expected running time of the (1+1) SAMA on 2-PTJ is bounded by $O(n^4 \log n)$.*

Proof. First, based on Lemma 13, 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) SAMA 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) SAMA 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 [6]), thus the expected running time for the (1+1) SAMA on the 2-PTJ is bounded by $O(n^4 \log n)$. □

Theorem 21. *The expected running time of each static (1+1) MA on 2-PTJ is exponentially large.*

Proof. Since Jansen and Wegener [6] 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) SAMA 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 PLS 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 20 and 21, we can see that for the function 2-PTJ, the (1+1) SAMA 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) SAMA

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 [6] 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) SAMA on the function 2-PTW.

Definition 22. 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 $1^i 0^{n_1 - i}$ 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 23. *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. The proof is similar to the proof in Theorem 20. 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 24. *The probability that the (1+1) SAMA needs an exponential number of steps on* `2-PWT` *is exponentially close to 1.*

Proof. 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 [6]). □

4 Experimental Results and Analysis on Maximum Clique Problem

The Maximum Clique Problem is a NP-hard problem [8]. In this section, we will formalize a fitness function $f_{\texttt{MCP}}$ for the Maximum Clique Problem, and then test it by using the Dynamic (1+1) EA, (1+1) MA, (1+1) SAMA_RCLS and (1+1) SAMA_RPLS respectively.

For a given graph $G = (\{v_1, v_2, ..., v_n\}, E)$, a bit string $x = (x_1, x_2, ..., x_n) \in \{0,1\}^n$ defines a Maximum Clique potential solution 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 25. 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 minimum number of missing edges such that x can represent a clique if these edges are added into the graph G.

Example 26. For a given graph G in Figure 1, $f_{\texttt{MCP}}(1101) = 3$ 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 $(1, 3)$.

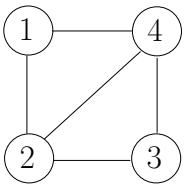

Figure 1: A graph G.

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

Table 2 reports the maximum clique results on some DIMACS instances [1]. Column 1 depicts the graph names with their best known clique size in the parentheses. We test the Dynamic (1+1) EA [5] in column 2, the (1+1) MA [2] in column 3, the (1+1) SAMA_RCLS in column 4 and the (1+1) SAMA_RPLS in column 5. Each algorithm is run on each graph 10 times where each run is limited with one minute of running time. 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 (algorithm 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.

From Table 2, we claim the following:

1. Four of the algorithms have found the global optimum in some small order graphs such as `C125.9` and `keller4`. This denotes that each algorithm is able to find the global optimum if it has enough computing time.

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

3. The three Memetic Algorithms outperform the Dynamic Evolutionary Algorithm on most graphs. This denotes that the local search approaches of MAs are useful.

4. Each iteration of the Dynamic (1+1) EA uses the least running time while each iteration of the (1+1) SAMA_RCLS uses the most running time, i.e. the Dynamic (1+1) EA has the largest value in the "Gen" sub-column for each graph, while the (1+1) SAMA_RCLS has the smallest value. This denotes that the local search approaches, especially the RCLS, are very time consuming.

5. The (1+1) SAMA_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 not require the global optimal solutions, but they have strict requirements on the running time.

Now we analyze the effects of the dynamic mutation probability and two local searches.

4.1 Time complexity analysis for the RCLS and the RPLS

To optimize a given string x on the function $f_{\texttt{MCP}}(x)$, the local search RCLS (Algorithm 2) needs n steps to evaluate the results of flipping each bit in x and then decide to flip a bit in x that increases the fitness the most. So if x does not represent a clique, the RCLS will take $O(n^2)$ steps to flip some bits in x from one to zero to get a clique; and later on take $O(n^2)$ steps to find a local optimal clique or global optimal clique. Therefore, the running time of the RCLS in the (1+1) SAMA is $O(n^2)$ per iteration.

The local search RPLS (Algorithm 3), however, will flip a bit in x as soon as this change can increase the fitness function $f_{\texttt{MCP}}(x)$, and then check the next bit according to the random permutation. So the RPLS will take $O(n)$ steps to find a clique, and then take $O(n)$ steps to reach a local or global optimal clique. Thus, the running time of the RPLS in the (1+1) SAMA is $O(n)$ per iteration.

To compare the RCLS with the RPLS, the disadvantage of the RCLS is that it is expensive in terms of the time complexity. But the RCLS has its advantage that it evaluates all neighbors of x to do one flip, thus if the fitness function can give a correct hint to distinguish all neighbors of x, the RCLS can directly approach the global optimum without trapping into any local optimum. Here, we claim that no polynomial fitness function can directly guide the RCLS to the global optimum, otherwise the maximum clique problem can be solved by the RCLS polynomially.

The experimental results between the (1+1) SAMA_RPLS and the (1+1) SAMA_RCLS in Table 2 shows that the RPLS is more efficient than the RCLS in all test graphs, except for the graph `gen200_p0.9_44`.

4.2 Ability to avoid stagnation

Now we analyze how these tested algorithms are coping with the stagnation when they are trapped into a local optimal clique.

Definition 27. 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 number of ones in x.

So $\texttt{BLOCKONES}(x)$ is the minimal number of ones in x, 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 28. *If the four tested algorithms are stagnated at a local optimal solution x with $t = \textit{BLOCKONES}(x)$, the expected running time to skip out of this local optimal solution and find a larger clique is bounded by*

1. $O(n^{2t+1} \log n)$ *for the Dynamic (1+1) EA;*
2. $O(n^{2t+2})$ *for the (1+1) MA;*
3. $O(n^{2t+2} \log n)$ *for the (1+1) SAMA_RCLS; and*
4. $O(n^{2t+1} \log n)$ *for the (1+1) SAMA_RPLS.*

Proof. Part 1. Since $t = \texttt{BLOCKONES}(x)$, there exist a larger clique y such that x can flip t bits to get a sub-clique of y. So the Dynamic (1+1) EA needs 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 a larger clique. So the probability of the Dynamic (1+1) EA to find this larger clique in one step 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 $p = 1/n$, $\texttt{Prob}_{\texttt{success}} = \Omega\big(n^{-(2t+1)}\big)$; and because the dynamic mutation probability has a $p = 1/n$ in every $\lceil \log n \rceil$ mutations, the first upper bound is proved.

Part 2. Note that the (1+1) MA, the (1+1) SAMA_RCLS, and the (1+1) SAMA_RPLS both have a local search approach, so these algorithms can flip t bits from one to zero and flip t bits from zero to one; then we get a new clique with the same clique size as x. But note that this new clique is a sub-clique of y, so the local search will flip at least one more bit to get a larger clique. 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 in each iteration, and RCLS needs $O(n^2)$ steps in each iteration; also the (1+1) SAMA_RPLS and the the (1+1) SAMA_RCLS have at least one mutation with probability $p = 1/n$ in every $\lceil \log n \rceil$ iterations, the rest upper bounds are proved. □

Definition 29. For a given graph G, function $\texttt{MAXBLOCKONES}$ (G) is formalized as: $\texttt{MAXBLOCKONES}(G) = \max\{\texttt{BLOCKONES}(x) \mid x \text{ is a clique in G}\}$.

Theorem 30. *For a given graph G, let $t = \textit{MAXBLOCKONES}(G)$. The expected running time of the four tested algorithms to find the maximum clique of G is bounded by*

1. $O(n^{2t+2} \log n)$ *for the Dynamic (1+1) EA;*
2. $O(n^{2t+3})$ *for the (1+1) MA;*
3. $O(n^{2t+3} \log n)$ *for the (1+1) SAMA_RCLS; and*
4. $O(n^{2t+2} \log n)$ *for the (1+1) SAMA_RPLS.*

Proof. Part 1. We prove the upper bound of 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 $O(n^2 \log n)$; and (b) if the start string x represents a clique, then the expected running time of finding the maximum clique is bounded by $O(n^{2t+2} \log n)$.

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 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 $p = 1/n$; and we have one mutation with $p = 1/n$ in every $\lceil \log n \rceil$ mutations. So we will expect to flip at least one bit in x from one to zero in $O(n \log n)$ steps; and x has at most n bits of ones, so we will expect to find a clique in $O(n^2 \log n)$ steps.

Step (b): Based on Lemma 28, the Dynamic (1+1) EA can skip out of a local optimal clique and find a larger clique by $O(n^{2t+1} \log n)$ steps; and the maximum clique of G will be obtained before n times of this skip performed.

So the Dynamic (1+1) EA is expected to find the maximum clique of G in $O(n^{2t+2} \log n)$ steps.

Part 2. The proof is similar to Part 1. We have shown 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 Section 4.1). Also, from Lemma 28, 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, the maximum clique of G will be obtained before n times of this skip performed. So the upper bounds of the (1+1) MA, the (1+1) SAMA_RCLS and the (1+1) SAMA_RPLS are proved. □

Corollary 31. *For all graphs G with $t = \texttt{MAXBLOCKONES}(G)$, we have*

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

2. *If $t = \omega(1)$ and $t = o(n)$, the four tested algorithms are expected to find the maximum clique of G sub-exponentially.*

Theorem 30 and Corollary 31 show that if any graph is expected to take at least exponential time to find the maximum clique, it must contain a local optimal clique x with $\texttt{BLOCKONES}(x) = \Theta(n)$.

Theorem 32. *If we use a static mutation probability $p = 1/n$, for a local optimal clique x with $\texttt{BLOCKONES}(x) = \Theta(n)$, both the (1+1) MA with RPLS and the (1+1) MA with RCLS are expected to skip out of x and find a larger clique exponentially faster than the (1+1) EA.*

Proof. 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 x to a larger clique by: (a) directly mutating to a larger clique; or (b) mutating multiple times to different cliques with the same clique size as x, and then mutating to a larger clique.

So if we can show that for any clique y with $f_{\texttt{MCP}}(y) \geqslant f_{\texttt{MCP}}(x)$, the probabilities that the (1+1) MA with RPLS and the (1+1) MA with RCLS jump from x to y are

exponentially larger than the probability that (1+1) EA jumps from x to y. Then the probabilities that the (1+1) MA with RPLS and the (1+1) MA with RCLS escape from x and find a larger clique is exponential larger than the (1+1) EA. Even though the RPLS and the RCLS need $O(n)$ and $O(n^2)$ steps in their local searches respectively, the theorem still holds.

Now we show that the expected running time for the (1+1) MA with RPLS and the (1+1) MA with RCLS to jump from x to y are exponentially faster than the (1+1) EA. To jump from x to y, let t be the number of bits that we need to flip from one to zero, and u be the number of bits that we need to flip from zero to one, where $u \geqslant t$ since y is at least the same clique size as x.

Let $\texttt{Prob}_{\texttt{EA}}$, $\texttt{Prob}_{\texttt{MA-RPLS}}$ and $\texttt{Prob}_{\texttt{MA-RCLS}}$ be the probabilities that the three algorithms jump from x to y, respectively. Let $K^u_{\texttt{RPLS}}$ and $K^u_{\texttt{RCLS}}$ be the probabilities of the two different local searches, starting from a string that the mutation has only flipped those t bits from one to zero, choose to flip those u bits to get y, respectively. We have

$$\texttt{Prob}_{\texttt{EA}} = p^t p^u (1-p)^{n-t-u},$$

$$\texttt{Prob}_{\texttt{MA-RPLS}} = \Omega\big(p^t (1-p)^{n-t-u} K^u_{\texttt{RPLS}}\big),$$
$$\texttt{Prob}_{\texttt{MA-RCLS}} = \Omega\big(p^t (1-p)^{n-t-u} K^u_{\texttt{RCLS}}\big),$$

Note that for the two MA based algorithms, they only require the mutation to flip those t bits from one to zero, and require another $(n-u-t)$ bits not to flip. The mutations in MAs do not need to care for those u bits that should flip from zero to one to find y because the local search $K^u_{\texttt{RPLS}}$ and $K^u_{\texttt{RCLS}}$ will check those bits after the mutation.

Furthermore, if we want the RPLS to initially flip those u bits (any other flips may lead to a worse local optimum), the random permutation can have the indices of the u bits positioned ahead of the other $(n-u)$ bits. So the probability is:

$$K^u_{\texttt{RPLS}} \geqslant \frac{u!(n-u)!}{n!} \geqslant \left(\frac{1}{n-u+1}\right)^u,$$

and for the RCLS, the probability to firstly choose those u bits is:

$$K^u_{\texttt{RCLS}} \geqslant \frac{u}{n}\frac{u-1}{n-1}\cdots\frac{1}{n-u+1} \geqslant \left(\frac{1}{n-u+1}\right)^u.$$

Therefore,

$$\frac{\texttt{Prob}_{\texttt{MA-RPLS}}}{\texttt{Prob}_{\texttt{EA}}} \geqslant \frac{1}{(n-u+1)^u}\frac{1}{p^u} \text{ and}$$
$$\frac{\texttt{Prob}_{\texttt{MA-RCLS}}}{\texttt{Prob}_{\texttt{EA}}} \geqslant \frac{1}{(n-u+1)^u}\frac{1}{p^u}.$$

Since $p = 1/n$, $t = \Theta(n)$ and $u \geqslant t$, these ratios are exponentially large. □

4.3 When the dynamic mutation probability can help

To observe the effect of the dynamic mutation probability, we compare the results between the (1+1) SAMA_RCLS suite and the (1+1) MA suite. Note the (1+1) MA used the RCLS local search as well. From Table 2, we found that the (1+1) SAMA_RCLS outperforms the (1+1) MA on all test graphs with less than 300 nodes, such as brock200_2 and p_hat300-1. However, for graphs with more than 300 nodes, the (1+1) MA performs better.

Furthermore, Figure 2 records the details of the best run for the four algorithms on the first test graph '`brock200_2`'. Since all algorithms stagnate after 20 seconds, we only display the records from 0 to 20 seconds. From Figure 2 we can see that the (1+1) MA outperforms the (1+1) SAMA_RCLS from two second to four second. But after that, the (1+1) SAMA_RCLS outperforms the (1+1) MA.

Here we claim that the dynamic mutation probability in MAs is

1. Not helpful at the beginning of the search. This is because MAs combine a small mutation probability and a local search can easily find a larger clique at the beginning of the search. During this time, a larger mutation probability will flip many bits to one which costs the local search a lot of computation time to repair the solution to get a clique.

2. Helpful when the algorithm is stagnated at a local optimal clique for a relatively long time. This can be shown in the theorem below.

Theorem 33. *If any of the tested algorithms is trapped into a local optimal clique x with $\mathit{BLOCKONES}(x) = \Theta(n)$, let $\mathit{Prob}_{\Theta(1)}$ and $\mathit{Prob}_{\Theta(1/n)}$ be the probabilities that this algorithm skip out of the local optimal x with the mutation probabilities $p = \Theta(1)$ and $p = \Theta(1/n)$ respectively, we have $\mathit{Prob}_{\Theta(1)}/\mathit{Prob}_{\Theta(1/n)}$ is exponentially large.*

Proof. Let $t = \texttt{BLOCKONES}(x)$ with $t = \Theta(n)$. Firstly, when the mutation probability $p = \Theta(1)$, let $\lambda = p$ for a constant $0 < \lambda \leqslant 1/2$, the success skipping probability is bounded by directly flipping $2t + 1$ bits. This contains t bits of x from one to zero, and another $t + 1$ bits of x from zero to one, so the local search approach does not need to help. And the probability of this is: $\texttt{Prob}_{\Theta(1)} \geqslant p^{2t+1}(1-p)^{n-2t-1} = \lambda^{2t+1}(1-\lambda)^{n-2t-1} \geqslant (1-\lambda)^n$.

Secondly, when the mutation probability $p = \Theta(1/n)$, we know that any algorithm wants to skip out of the local optimal clique must flip at least t bits of x from one to zero, so the probability of skipping is bounded by: $\texttt{Prob}_{\Theta(1/n)} \leqslant p^t = (1/n)^t$.

Since $t = \Theta(n)$ and $0 < \lambda \leqslant 1/2$, we have

$$\frac{\texttt{Prob}_{\Theta(1)}}{\texttt{Prob}_{\Theta(1/n)}} \geqslant \frac{n^t}{(1-\lambda)^n},$$

is exponentially large. □

5 Conclusions and Future Work

We have defined the (1+1) SAMA with a self-adjusting mutation probability and two different local searches (RCLS and RPLS). The time complexity analysis proved that two local searches can outperform each other on different functions. Also on some functions, the (1+1) SAMA can drastically outperform each static (1+1) MAs; while on some other functions, a static (1+1) MA can drastically outperform the (1+1) SAMA.

The experimental results on the Maximum Clique Problem have shown the success of the self-adjusting strategy and the new random permutation local search. We then proposed a metric of analyzing the expected running time of escaping a local optimal solution. We have shown some advantages of using a dynamic mutation probability and random permutation local search.

A next step to investigate in the future is to compare the random permutation local search with other local searches on the clique problems with different fitness functions. Also, we would like to try our proposed complexity metric and new local search techniques on other optimization problems.

Acknowledgements

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

References

[1] 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. American Mathematical Society, 1996.

[2] D. Sudholt. On the Analysis of the (1+1) Memetic Algorithm. Proceedings of the 8th annual conference on Genetic and evolutionary computation. 493–500, 2006.

[3] E.K. Burke and D.J.L. Silva. The design of memetic algorithms for scheduling and timetabling problems. In N. Krasnogor, W. Hart, and J. Smith (eds.), Recent Advances in Memetic Algorithms, Studies in Fuzziness and Soft Computing, vol. 166, 289–312, 2004.

[4] F. Neri, C. Cotta, and P. Moscato (Eds). Handbook of Memetic Algorithms. Studies in Computational Intelligence, Volume 379, 2011.

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

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

[7] Wikipedia contributors. 'Memetic algorithm', Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/wiki/Memetic_algorithm, (accessed on 12 July 2012).

[8] Wikipedia contributors. 'Clique problem', Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/wiki/Clique_problem, (accessed on 23 July 2012).

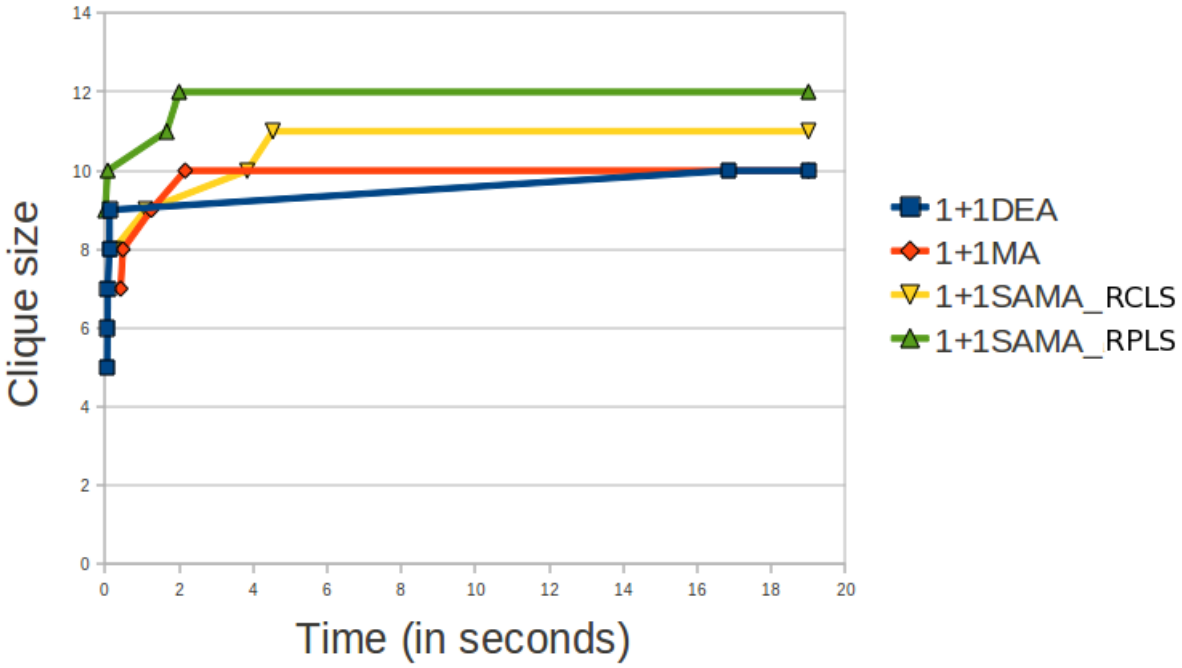

Figure 2: Running times for test case `Brock200_2`.

Table 1: A comparison of the algorithms.

	(1+1) EA	Dynamic (1+1) EA	(1+1) SAMA-RPLS	(1+1) SAMA-RCLS
Arbitrary functions	$\Theta(n^n)$	$O(4^n \log n)$	$O(2^n n^2 \log n)$	$O(2^n n^2 \log n)$
`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	$\Theta(n \log n)$	$\Theta(n^2 \log n)$	$\Theta(n)$	$\Theta(n^2)$
`LEADINGONES`	$\Theta(n^2)$	$\Theta(n^2 \log n)$	$\Theta(n^2)$	$\Theta(n^2)$

Table 2: A comparison of the algorithms on the Maximum Clique Problem for 1 minute of CPU time.

	Dynamic (1+1) EA			(1+1) MA			(1+1) SAMA-RCLS			(1+1) SAMA-RPLS		
Metrics	Best	Avg	Gen	Best	Avg	Gen	Best	Avg	Gen	Best	Avg	Gen
brock200_2 (=12)	10	9.5	5.4m	11	10.1	170.0k	11	10.9	1.6k	12	12	59.8.2k
brock200_4 (=17)	15	14	5.5m	16	14.9	139.5k	16	15.8	1.7k	17	16.8	57.3k
brock400_2 (=29)	22	20.6	2.2m	24	22.9	37.8k	24	22.7	170.0	25	24	11.7k
brock400_4 (=33)	22	19.5	2.2m	24	22.4	38.3k	24	22.7	169.7	24	23.3	11.9k
brock800_2 (=24)	17	15.1	830.4k	18	16.6	3.4k	17	15.4	11.1	19	18.6	2.3k
brock800_4 (=26)	14	12.4	1.3m	16	15.1	8.5k	16	13.8	16.1	16	16	3.9k
C125.9 (⩾34)	34	33.8	10.8m	34	33.5	143.5k	34	34	13.1k	34	34	115.3k
C250.9 (⩾44)	42	41.2	5.1m	44	42.2	45.5k	44	42.5	1.3k	44	43.4	30.9k
C500.9 (⩾57)	45	41.1	2.5m	49	46.4	15.5k	46	45.1	141.4	49	47.9	9.3k
C1000.9 (⩾68)	49	45.4	919.7k	53	50.4	890.8	52	49.2	17.6	59	56.2	1.8k
C2000.5 (⩾16)	10	9.1	384.7k	fail	fail	0	fail	fail	0	13	11.5	487.5
C2000.9 (⩾77)	42	39	423.6k	fail	fail	0	fail	fail	0	52	49.5	492.8
C4000.5 (⩾18)	fail	fail	70.7k	fail	fail	0	fail	fail	0	13	11.7	81.7
DSJC500.5 (⩾13)	12	10.1	1.5m	12	11.3	31.9k	11	10.9	59.9	13	12.5	6.7k
DSJC1000.5 (⩾15)	11	10.3	492.3k	fail	fail	0	fail	fail	0	13	13	1.1
gen200_p0.9_44 (=44)	44	39	6.3m	44	39.9	64.8k	44	40.7	2.6k	44	39.9	47.1k
gen200_p0.9_55 (=55)	55	44.1	6.2m	55	44.7	58.2k	55	55	2.3k	55	55	36.9k
gen400_p0.9_55 (=55)	47	44.5	2.7m	51	49.1	20.6k	50	48.4	270.8	52	50.2	11.5k
gen400_p0.9_65 (=65)	45	43.7	2.7m	55	48.7	21.1k	49	46	253.7	55	49.3	12.0k
gen400_p0.9_75 (=75)	56	48.8	2.7m	75	65.3	16.1k	75	64.8	287.0	75	65.7	9.9k
hamming8-4 (=16)	16	13.2	4.4m	16	16	90.7k	16	16	850.8	16	16	38.3k
hamming10-4 (=40)	30	27.7	720.7k	32	8 fails	20	32	9 fails	2	40	36.3	1.5k
keller4 (=11)	11	10.3	7.0m	11	11	209.1k	11	11	3.2k	11	11	94.0k
keller5 (=27)	18	16.2	1.4m	19	17.8	10.0k	17	16.8	24.1	19	18.9	4.0k
keller6 (⩾59)	fail	fail	1.3m	fail	fail	0	fail	fail	0	28	26.4	120.7
MANN_a27 (=126)	123	120.6	2.7m	125	124.3	6.8k	124	122.8	354.1	126	125.3	5.7k
MANN_a45 (=345)	331	330.1	502k	331	330.4	196.6	332	330.6	10.5	342	339	142.8
MANN_a81 (⩾1100)	fail	fail	135.1k	fail	fail	0	fail	fail	0	365	364.6	65.4
p_hat300-1 (=8)	8	6.8	3.2m	8	7.6	97.3k	8	8	336.8	8	8	25.0k
p_hat300-2 (=25)	25	23.9	2.6m	25	24.6	54.7k	25	25	320.8	25	25	18.1k
p_hat300-3 (=36)	34	32.1	2.9m	36	35.3	41.7k	36	35.4	413.2	36	35.5	18.0k
p_hat700-1 (=11)	8	7.1	951.2k	9	8.5	12.3k	9	8.3	14.6	11	9.8	3.0k
p_hat700-2 (⩾44)	30	26	951.0k	38	37.1	6.9k	37	34.8	22.5	38	37.7	2.8k
p_hat700-3 (⩾62)	40	33.8	1.4m	43	42.3	8.2k	42	40.8	37.4	43	42.9	4.3k
p_hat1500-1 (⩾12)	8	6.9	376.4k	fail	fail	0	fail	fail	0	11	9.4	581.8
p_hat1500-2 (⩾65)	54	44.4	213.9k	fail	fail	0	fail	fail	0	65	62.2	323.8
p_hat1500-3 (⩾94)	63	57.7	269.3k	fail	fail	0	fail	fail	0	88	85.3	390.8