A New Genetic Algorithm Based on Primal-Dual Chromosomes for Royal Road Functions

Shengxiang Yang

Department of Mathematics and Computer Science
University of Leicester
University Road, Leicester LE1 7RH, UK
s.yang@mcs.le.ac.uk   http://www.mcs.le.ac.uk/~syang

Abstract

Genetic algorithms (GAs) have been broadly studied by a huge amount of researchers and there have been many variations developed based on Holland's simple genetic algorithm (SGA). Inspired by the phenomenon of diploid genotype and dominance mechanism that broadly exist in nature, in this paper we propose a new genetic algorithm — primal-dual genetic algorithm (PDGA). PDGA operates on a pair of chromosomes that are primal-dual to each other through the primal-dual mapping, which works in the sense of Hamming distance in genotype. The primal-dual mapping improves the exploration capacity of PDGA and thus its searching efficiency in the search space. We compare the performance of PDGA over SGA based on the Royal Road functions, which are especially designed for testing GA's performance. The experiment results show that PDGA outperforms SGA for different performance measures.

Keywords: Genetic algorithm, primal-dual mapping, diploidy, dominance, building blocks, fitness landscape, search space, hitchhiking, Royal Road functions.

1 Introduction

During the last thirty years, there has been a growing interest in developing problem solving systems based on natural principles, such as evolution and heredity. These evolution-based systems, commonly called *Evolutionary Algorithms* (EAs) [16], maintain a population of candidate solutions to a given problem which are evaluated according to a problem-specific fitness function that defines the environment for the evolution. New population is created by selecting relatively fit members of the present population and recombining them through genetic operations using various genetic operators. Evolutionary algorithms can be briefly classified into four main categories: *Evolution Strategies* (ES) [40, 44], *Evolutionary Programming* (EP) [12], *Genetic Algorithm* (GA) [24], and *Genetic Programming* (GP) [29]. Of these four categories, GA is the best known and has been widely studied by a huge amount of researchers.

Genetic algorithms (GAs) were first proposed by Holland [24], usually called Holland's canonical or simple genetic algorithm (SGA), and then developed by Holland and his colleagues and students in the 1960s and the 1970s. Due to GA's characteristics of easy-to-use, great robustness and good parallel processing capacity, it has been used in a great number of scientific and engineering problems and models, such as numerical function optimization [9], combinatorial optimization (e.g., the knapsack problem [23], the job-shop scheduling problem [8, 15], and the traveling salesman problem [17]), automatic programming [3], machine learning [20], economics

[1], automatic control [28], and signal processing [32]. Recently, multiobjective genetic algorithms for multiobjective optimization problems have become a very hot research area among GA's community [49], such as the vector evaluated genetic algorithm (VEGA) [42], the multiobjective optimization genetic algorithm (MOGA) [13], the niched pareto genetic algorithm (NPGA) [26], and the nondominated sorting genetic algorithm (NSGA) [45].

Based on Holland's SGA, there have been many improvements, variations and extensions developed, involving GA's macro-structure and micro-structure. GA's macro-structure variations include hybrid GAs [9], Master-slave GAs [9], and parallel GAs (e.g., massively parallel GAs [38], parallel island GAs [47], and parallel hybrid GAs [22]). In the aspect of GA's micro-structure, variations appear on every aspect of GA's process, including chromosome representation schemes (e.g., binary coding [20], real coding [2, 21] versus problem-specific coding [34], haploid versus diploid [4, 7, 19]), selection strategies (e.g., the elitist model [9], the ranking model [5], the stochastic universal sampling model [6], and several scaling strategies [4, 20, 34]), mating policies (e.g., sharing function [18], seduction [41], incest prevention technique [11]), crossover operators (e.g., one-point, two-point and uniform crossover [20]), and mutation operators (e.g., bit flip mutation and inverse mutation [20]).

Most GAs studied so far are haploidy-based, i.e., they operates on a set of single-stranded chromosomes. Haploid genotype is the simplest genotype found in nature. GA's researchers have also studied diploid genotype (pairs of chromosomes) and dominance mechanism (an important genotype-to-phenotype mapping mechanism) for a long history [4, 7, 19, 31, 39]. In Section 2 of this paper, we will further discuss diploid genotype and dominance mechanism.

In this paper, inspired by the phenomenon of diploid genotype and dominance mechanism that broadly exist in nature, we propose a new genetic algorithm based on primal-dual chromosomes, called primal-dual genetic algorithm (PDGA in brief). PDGA operates on a pair of chromosomes which are primal-dual to each other in the sense of Hamming distance in genotype. Through the primal-dual mapping of operated pair of chromosomes, the exploration capacity of PDGA in the search space is improved and thus the searching efficiency of PDGA as well as its convergence speed is improved. In the present study, we provide a comparison of the performance of PDGA over SGA based on the Royal Road functions [14, 35, 36, 37]. These functions are especially designed for testing the performance of GAs and thus can act as good test problems. The experiment results show that PDGA outperforms SGA on the Royal Road functions for different performance criteria, such as function evaluations to optimum, capacity of achieving optimum, efficiency of function evaluation, and dynamic performance of schema recombination.

The rest of this paper is organized as follows. Section 2 first briefly reviews the conventional simple genetic algorithm (SGA), then presents in detail the proposed PDGA including the concept of primal-dual chromosomes, the framework of PDGA, the motivation and property of PDGA. Section 3 briefly reviews the building block hypothesis and schema theorem and describes the Royal Road functions that are used to test and analyze the performance of PDGA over SGA. Section 4 presents the computer experiment results with several performance criteria. Based on the experiment results of section 4, Section 5 gives some analyses and discussions on the inside mechanism of PDGA which leads to PDGA's outperforming SGA. Finally Section 6 concludes this paper and points out some potential directions for our future research.

2 Primal-Dual Genetic Algorithm

In this section, we describe the proposed PDGA in detail. For the sake of comparison, before introducing PDGA, we first briefly describe the conventional simple genetic algorithms. Then we give out the concept of primal-dual chromosomes and the framework of PDGA. Finally we explain the motivation of PDGA and discuss its haploidy nature instead of diploidy.

```
Procedure SGA:

begin
  parameterize(N, P_c, P_m, t_max);
  t := 0;
  initializePopulation(P(0));
  for each individual x in P(0) do        {evaluate P(0)}
    evaluate x;
  endfor;
  repeat
    P' := selectForReproduction(P(t));
    recombine(P');
    mutate(P');
    for each individual x in P' do        {evaluate P'}
      evaluate x;
    endfor;
    P(t + 1) := selectForSurvival(P(t), P');
    t := t + 1;
  until terminated = true;        {e.g., t > t_max}
end;
```

Figure 1: Pseudocode for SGA.

2.1 Simple Genetic Algorithm

The simple genetic algorithm, as one kind of generation-based evolutionary algorithms, maintains a population of candidate solutions or haploid chromosomes to a given problem which are evaluated according to a problem-specific fitness function that defines the environment for the evolution. New population is created by selecting relatively fit members of the present population and recombining them through crossover and mutation operations. The pseudocode of SGA is shown in Figure 1.

When applying SGA to solve a specific problem, such as an optimization problem, we must first select a problem-specific chromosome coding scheme (binary coding or real coding) and corresponding selection, crossover and mutation operators, and set values for such parameters as the population size N, the crossover probability P_c, the mutation probability P_m, and the maximum allowable generation number t_{max}. Then we can run the specific SGA to solve the given problem.

First, an initial population is created randomly, usually with no fitness or structural bias and the running generation counter t is initialized to 0. Then, in the main loop, a temporary population P' is selected from the current population $P(t)$ to generate the mating pool using the chosen selection strategy, which is called *selection for reproduction*. Afterwards, the recombination and mutation genetic operators are applied to some or all individuals (or members) of the temporary population. Usually, the main loop is repeated until $t > t_{max}$ or some other termination condition such as the time limit or convergence criterion is satisfied. The newly created individuals are evaluated by calculating their fitnesses. Before a new generation is processed, the new population $P(t+1)$ is selected from the old population $P(t)$ and the temporary population P', which is called *selection for survival*. And now, SGA can continue by building a new temporary population.

```
Procedure PDGA:

begin
  parameterize(N, P_c, P_m, t_max);
  t := 0;
  initializePopulation(P(0));
  for each individual x in P(0) do        {evaluate P(0)}
    evaluate x and x̄;
    if f(x̄) > f(x) then x := x̄;          {replace}
  endfor;
  repeat
    P' := selectForReproduction(P(t));
    recombine(P');
    mutate(P');
    for each individual x in P' do        {evaluate P'}
      evaluate x and x̄;
      if f(x̄) > f(x) then x := x̄;        {replace}
    endfor;
    P(t + 1) := selectForSurvival(P(t), P');
    t := t + 1;
  until terminated = true;        {e.g., t > t_max}
end;
```

Figure 2: Pseudocode for PDGA.

2.2 Primal-Dual Genetic Algorithm

Inspired by the phenomenon of diploidy genotype and dominance mechanism that broadly exist in nature, we propose a new primal-dual genetic algorithm (PDGA). PDGA operates on a pair of chromosomes which are primal-dual to each other in the sense of Hamming distance in genotype.

Here we first define the concept of primal-dual chromosomes. We only consider binary bit string representation of genotype and define a pair of chromosomes to be *primal-dual* to each other if their Hamming distance (the number of locations at which corresponding bits differ) is the maximum (equal to their length). That is, for a chromosome $x = (x_1, x_2, ..., x_L) \in I = \{0,1\}^L$ of fixed length L, its dual chromosome is defined as $\bar{x} = (\bar{x}_1, \bar{x}_2, ..., \bar{x}_L) \in I$ where $\bar{x}_i = 1 - x_i$ $(i = 1..L)$. For example, given a chromosome $x = 110000$ of fixed length 6, its dual chromosome is $\bar{x} = 001111$. Given this definition we say that x is mapped to $\bar{x}$ by the primal-dual mapping or Hamming distance mapping, vice versa.

PDGA is quite simple relative to other genetic algorithm variants. With above definition of primal-dual chromosomes, we can now give out the framework of PDGA in the form of pseudocode in Figure 2, where $f(x)$ denotes the fitness of an individual x. Comparing Figure 1 and Figure 2, we can see that PDGA differs from SGA only in the evaluation of chromosomes in the population. The genetic operations including selection, crossover and mutation are all the same for both PDGA and SGA.

2.3 Exploration and Exploitation

An important concept that helps understand GA's dynamic behavior is that of *fitness landscape*, which was originally defined by the biologist Swell Wright [48] in the context of population genetics. Fitness landscape has been proven to be very powerful in the analyses of behaviors of combinatorial optimization algorithms and evolutionary algorithms. A fitness landscape is a representation of the space of all possible genotypes along with their fitnesses. Viewing the search space (i.e., the set of all candidate solutions) as a landscape, the plot of fitness values can form "hills," "peaks" and "valleys", which are analogous to those of physical landscapes; the height of a point in the search space reflects the fitness of the solution associated with that point. An optimization algorithm can be thought of as navigating through the search space in order to find the highest peak in the fitness landscape. Several properties of a fitness landscape are known to have important influence on the performance of optimization algorithms, such as the number of local optima (peaks) in the landscape, the distribution of peaks in the search space, and the landscape ruggedness (i.e., the correlation between neighboring points in the search space) [27, 33].

For an optimization algorithm to work efficiently, it should use the following two techniques: *exploration* and *exploitation*. Exploration is used to investigate new, useful areas in the search space and exploitation is used to make use of knowledge acquired by exploration to reach better positions on the search space. In Figure 3 we illustrate both exploration (by dashed and dash-dotted arrows) and exploitation (by solid arrows) in a fitness landscape for different methods including the GA and two class of broadly used heuristic algorithms *random search* and *hillclimbing* (also known as *neighborhood search* or *local search* [30]).

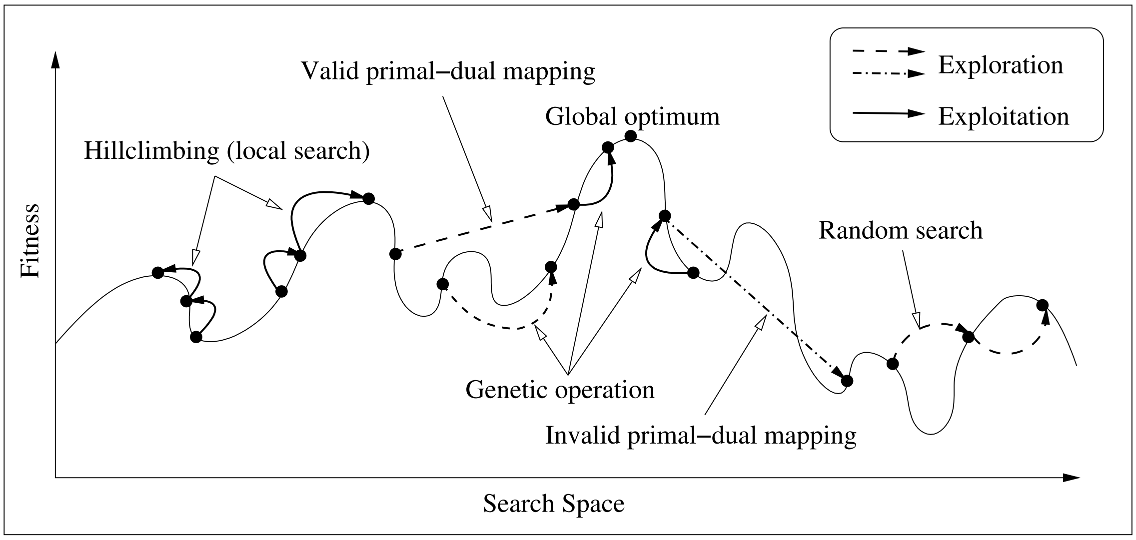


Figure 3: Exploration and exploitation in the fitness landscape.

Random search method starts from a randomly created point (called current best point) in the search space and then makes a random "jump". If the new point jumped at is better, it becomes the current best point, otherwise the algorithm keeps jumping randomly until a better point is achieved. And from the new obtained current best point, the algorithm will make new random jumps. This process continues until the termination condition is satisfied, e.g., the number of evaluations reaches the prescribed maximum limit. Hillclimbing also starts from a random point (called current point) and then select a new point from the neighborhood of the current point. If the new point has fitness higher than the current point, it becomes the new current point. Otherwise some other neighbor is selected and tested. The method stops if no further improvement is possible (i.e., local optimum is achieved). Random search is a typical strategy that explores the search space while neglecting the exploitation of the promising regions of the space; while hillclimbing is a strategy that exploits the best solution for possible improvement and neglects exploration of the search space.

As a class of general purpose (domain independent) meta-heuristic search methods, GAs strike a good balance between exploration and exploitation of the search space. GAs achieve this balance through selection, reproduction and replacement mechanism, genetic operators and multi-point search via a population of solutions. GAs are more robust than hillclimbing and pure random search methods because GAs are probabilistic algorithms that combine elements of directed search via recombination and stochastic search via mutation. The recombination and mutation operators are aimed to explore (or exploit) the search space by jumping to new regions (or local regions), as illustrated in Figure 3. GAs also perform multidirectional search by maintaining a population of potential solutions and encourage information gathering and exchange between these directions.

The motivation of PDGA is to improve GA's exploration capacity in the search space through the primal-dual mapping between the operated pair of chromosomes. As shown in Figure 3, the primal-dual mapping can be thought of as long distance jumping (maximal distance jumping in the sense of Hamming distance) that explores the fitness landscape. There are two types of primal-dual mapping: one that jumps upward the fitness landscape from a low fitness point (primal chromosome) to a high fitness point (dual chromosome), as illustrated by the dashed arrow in Figure 3, and the other that jumps downward the fitness landscape from a high fitness point to a low fitness point, as illustrated by the dash-dotted arrow in Figure 3. The former primal-dual mapping replaces the primal chromosome with its dual peer and thus is called *valid primal-dual mapping*, the latter mapping does nothing and is called *invalid primal-dual mapping*. It is the valid primal-dual mappings that are expected to improve the efficiency of PDGA's searching in the search space.

2.4 Haploidy over Diploidy

Dominance in nature is usually associated with genetic material presented using diploid chromosomes. In the general diploid form a genotype carries two sets of chromosomes (called *homologous* chromosomes), each containing information for the same functions. Each position in the genotype has two or multiple allele values. In nature each allele might represents a different phenotypic characteristic. When building a body the genes from one set compete with those in the other set. Genes that are dominant are expressed in the phenotype of an organism and those that are less likely to be expressed are recessive. Here a genetic operator, called *dominance mechanism* by geneticists, is required for determining which allele value for a gene will be expressed when the allele values do not agree [20]. In nature the donimance mechanism is determined by and also evolve with the environment.

There have been numerous explanations and theories proposed for diploidy and dominance. The most sensable theories state that diploidy provides a mechanism for remembering previously useful alleles and allele combinations and dominance provides an operator to shield those remembered alleles from harmful selection in a currently hostile environment. The redundant memory of diploidy permits multiple solutions to the same problem to be carried along with only one particular solution expressed [20].

Diploidy and dominance have long been the research topics of the GA community. Bagley [4] first used a variable dominance map that was coded as part of the chromosome. Hollstien [25] proposed two simple evolving dominance mechanisms for function optimization: two-locus dominance scheme and single-locus triallelic dominance scheme. Hollstien's single-locus triallelic dominance scheme drew alleles from the 3-alphabet $\{0, 1, 2\}$ where both 2 and 1 map to "1," but 2 dominates 0 and 0 dominates 1. Holland [24] later studied Hollstien's triallelic scheme and introduced the clearer alphabet $\{0, 1_0, 1\}$ instead of Hollstien's $\{0, 1, 2\}$. Brindle in her dissertation [7] studied six dominance schemes for a number of function optimization problems. Goldberg and Smith [19] compared the performance of a haploid GA, a diploid GA with fixed dominance map where 1's doinate 0's, and a diploid GA with Holstien-Holland triallelic dominance map on a blind, nonstationary knapsack problem and concluded that Holstien-Holland triallelic dominance is better than either fixed dominance or haploid structure. Ng and Wong [39] proposed a diploid representation with simple dominance change for non-stationary function optimization. More recently Lewis *et. al.* [31] have compared various diploid GAs with or without mechanisms for dominance change on non-stationary problems and concluded that some form of dominance change is essential as a diploid encoding is not flexible enough to respond to change on these problems.

From above description of PDGA, it seems that PDGA is diploidy-based since it works on a pair of primal-dual chromosomes. However, further thinking will make it a haploidy-like genetic algorithm since in fact we needn't explicitly keep track of each chromosome's dual chromosome. That is, the dual chromosome $\bar{x}$ of a primal chromosome x can be looked as the shadow of x and only shows its body through the primal-dual mapping when x is evaluated. Of course, if $\bar{x}$ proves to be better, it will embody itself and throw x into its shadow and in this case we say $\bar{x}$ dominates x. That is, each chromosome in PDGA carries intrinsically with itself some redundant information or memory that can be used as soon as needed. In this sense, we can call PDGA a *pseudo-diploid* or *implicitly diploid* genetic algorithm.

Here we must also note that PDGA is different from those genetic algorithms [4, 7, 19, 31, 39] that are based on diploidy and dominance in the following three aspects. First, in those GAs dominance is gene oriented and thus needs special dominant scheme while in PDGA dominance is phenotype oriented and thus needs no special dominant scheme. Second, in those GAs the chromosomes undergoing dominance operation are chosen randomly while in PDGA dominance operates on the primal-dual pair of chromosomes. Finally, in those GAs genetic operations perform on the pair of chromosomes while in PDGA only the winner or dominant of the primal-dual pair goes through genetic operations.

3 Building Blocks and Royal Road Functions

In this section, we first briefly review Holland's schema theorem and building block hypothesis and then we describe in detail the Royal Road functions, which are chosen as the test problems of the experiments carried out and reported later on in this paper to compare the performance of PDGA over SGA.

3.1 Schema Theorem and Building Block Hypothesis

To formally analyze the behavior of genetic algorithms, Holland [24] first proposed the notation of *schema* to describe a subset of all binary vectors of fixed length that have similarities at certain positions. A schema is typically specified by a vector over the alphabet {0, 1, *}, where the "*" denotes a "wildcard" or "don't care" bit that matches both 0 and 1. For example, the schema $S = 1****0$ represents the set of all 6-bit strings that begin with 1 and end with 0. If a bit string x fits this template (e.g., $x = 100010$ or $x = 101010$), it is said to be an *instance* of S or, equivalently, $x \in S$. Given a schema S, its *order* $o(S)$ is simply defined as the number of fixed or defined positions within S and its *defining length* $l(S)$ is defined as the maximum distance between fixed positions within S or, equivalently, the distance between the outermost defined bits of S. For example, given $S = 01**1*$, $o(S) = 3$ or, equivalently, S is said to be of order 3 because there are 3 fixed positions in S and $l(S) = 4$ for the first fixed position is 1 and the last is 5, $l(S) = 5 - 1 = 4$.

Given the concept of schema, Holland worked out the *schema theorem* for GAs using the fitness proportionate selection, one-point crossover and bit flip mutation operators. The schema theorem states that short, low-order, better than average schemas (also called *building blocks*) receive an exponentially increasing number of trials in the subsequent generations. An immediate result of the schema theorem is that GAs explore the search space by short, low-order schemas that are subsequently used for information exchange during crossover. The building block hypothesis states that a GA seeks near-optimal performance through the juxtaposition of short, low-order, high-fit schemas into increasingly fit, higher-order schemas.

For a string of length L, there are in total 2^L possible schemas. Thus a population of size N contains somewhere between 2^L to $N * 2^L$ schemas, depending on the population diversity. Of these many schemas, those schemas that are usefully processed by the GA are of the order N^3 [20]. That is, while the GA is explicitly processing a population of N individuals during each generation, it is in fact implicitly processing a much larger number of useful schemas (of the order N^3) and increasing their representation at an exponential rate. This is called GA's *implicit parallelism.*

Holland's schema theorem and building block hypothesis are the theoretical foundations of GAs and have been shown to have far reaching influence on GA's research. However, the schema theorem and the building block hypothesis do not state how crossover, the major source of the search power of GA, works to recombine highly fit schemas from short, low-order schemas.

3.2 Royal Road Functions

To investigate schema processing and recombination in detail, Mitchell, Forrest and Holland designed a class of fitness landscapes, called Royal Road functions, to capture the essence of building blocks in an idealized form [14, 35, 37]. For example, the Royal Road functions R_1 and R_2 are shown in Figure 4 and Figure 5 respectively.

Royal Road functions R_1 and R_2 are defined using a list of schemas. Each schema s_i is given a coefficient c_i which is equal to its order (i.e., $c_i = o(s_i)$). From Figure 4 and Figure 5, we can see that for R_1, $c_i = 8$ for all s_i $(i = 1..8)$ while for R_2, $c_i = 8$ for s_i $(i = 1..8)$, $c_i = 8$ for s_i $(i = 9..12)$, and $c_i = 8$ for s_i $(i = 13, 14)$. The fitness of a bit string x for both $R_1(x)$ and $R_2(x)$ is computed by summing the coefficients c_i corresponding to each of the given schema s_i of which x is an instance. That is, $R_1(x)$ and $R_2(x)$ of a bit string x are defined as follows:

$$R_1(x) = \sum_{i=1}^{i=8} c_i \delta_i(x) \text{ and } R_2(x) = \sum_{i=1}^{i=14} c_i \delta_i(x)$$

where $\delta_i(x) = \{1, \text{if } x \in s_i; 0, \text{otherwise}\}$. For R_1, if a string x is an instance of exactly two of the order-8 schema, e.g., $x = 111111110..011111111$, $R_1(x) = 16$ since x is an instance of s_1 and s_8. However, for R_2 a string's fitness depends not only on the number of 8-bit schemas to which the string belongs, but also on their positions in the string. For example, the string $x = 111111110..011111111$ has a fitness $R_2(x) = 16$ since x is an instance of s_1 and s_8, while the string $y = 11111111111111110..0$ has a fitness $R_2(y) = 32$ since y is an instance of s_1, s_8 and s_9. Similarly, the optimal solutions for R_1 and R_2 are given as follows: $R_1(s_{opt}) = R_1(111..1) = 64$ and $R_2(s_{opt}) = R_2(111..1) = 192$.

According to the building block hypothesis, two fitness landscape features are particularly relevant for the GA: one is the presence of short, low-order, highly fit schemas; the other is the presence of intermediate "stepping stones", i.e., intermediate-order higher-fitness schemas that result from combinations of the lower-order schemas, and that in turn can combine to form even higher-order, higher-fitness schemas. The Royal Road functions contain tailor-made building blocks or schemas of short order and short defining length that contribute to an individual's fitness. These schemas are hierarchically structured with pre-defined corresponding fitness values and build up the fitness landscape into a staircase-like shape with several levels. For example, in R_2, schemas s_1 to s_8 are the lowest level (level 0) schemas. Level 1 schemas s_9 to s_{12} comprise of a combination of adjacent level 0 schemas and in turn they are the stepping stones for level 2

$$
\begin{aligned}
s_1 &= 11111111**;\ c_1 = 8\\
s_2 &= ********11111111**;\ c_2 = 8\\
s_3 &= ****************11111111**;\ c_3 = 8\\
s_4 &= ************************11111111********************************;\ c_4 = 8\\
s_5 &= ********************************11111111************************;\ c_5 = 8\\
s_6 &= **11111111****************;\ c_6 = 8\\
s_7 &= **11111111********;\ c_7 = 8\\
s_8 &= **11111111;\ c_8 = 8\\
s_{opt} &= 1111111111111111111111111111111111111111111111111111111111111111
\end{aligned}
$$

Figure 4: Royal Road Function R_1. Here $c_i = order(s_i)$ and $R_1(s_{opt}) = 64$.

$$
\begin{aligned}
s_1 &= 11111111**;\ c_1 = 8\\
s_2 &= ********11111111**;\ c_2 = 8\\
s_3 &= ****************11111111**;\ c_3 = 8\\
s_4 &= ************************11111111********************************;\ c_4 = 8\\
s_5 &= ********************************11111111************************;\ c_5 = 8\\
s_6 &= **11111111****************;\ c_6 = 8\\
s_7 &= **11111111********;\ c_7 = 8\\
s_8 &= **11111111;\ c_8 = 8\\
s_9 &= 1111111111111111**;\ c_9 = 16\\
s_{10} &= ****************1111111111111111********************************;\ c_{10} = 16\\
s_{11} &= ********************************1111111111111111****************;\ c_{11} = 16\\
s_{12} &= **1111111111111111;\ c_{12} = 16\\
s_{13} &= 11111111111111111111111111111111********************************;\ c_{13} = 32\\
s_{14} &= ********************************11111111111111111111111111111111;\ c_{14} = 32\\
s_{opt} &= 1111111111111111111111111111111111111111111111111111111111111111
\end{aligned}
$$

Figure 5: Royal Road Function R_2. $R_2(s_{opt}) = 192$.

schemas s_{13} and s_{14}. And the highest level schema s_{opt} is the solution of R_2. This stepping-stone structure lays out a "royal road" for the GA to follow to the global optimum.

As stated in [14], the Royal Road functions provide an ideal laboratory for studying GA's behavior for several reasons. First, the landscape can be altered, e.g., by adding or removing stepping stones, by changing the size of level 0 schemas, and by increasing or decreasing the height of the intermediate stepping stones. Second, all possible fitness values including the global optimum are known in advance, which makes it easy to compare GA's performance on different functions. Finally, explicitly building all desired schemas into the functions (thus knowing them in advance) makes it easy to study GA's dynamic performance of schema processing and recombination. For these reasons, we use the Royal Road functions R_1 and R_2 as our test problems to compare the performance of PDGA over SGA in this paper.

For Royal Road functions, with PDGA we can evaluate the primal-dual chromosome pair at the same time with only a few extra instructions than when we evaluate one single chromosome with SGA. For example the pseudocode of procedure $R_1(x)$ that evaluates a bit string $x = (x_1, x_2, ..., x_L)$ of fixed length L is shown in Figure 6. From Figure 6 we can see that only lines 11, 12, 17 to 22 are extra for PDGA than SGA. Here we gain an almost free lunch in the sense of function evaluations. And for this reason we count the evaluations of the primal-dual pair as one evaluation in our following experiments.

```
Procedure R_1(x):

begin
  onesCount := 0;
  dualFitness := 0, primalFitness := 0;
  for i := 1 to L do
    if x_i = 1 then
      onesCount := onesCount + 1;
    endif;
    if (i MOD schemaOrder) = 0 then    {schemaOrder = 8}
      if onesCount = schemaOrder then
        primalFitness := primalFitness + schemaOrder;
      else if onesCount = 0 then
        dualFitness := dualFitness + schemaOrder;
      endif;
      onesCount := 0;    {clear the counter}
    endif;
  endfor;
  if dualFitness > primalFitness then    {replace}
    primalFitness := dualFitness;
    for i := 0 to L do
      x_i := x̄_i;    {x̄_i = 1 − x_i}
    endfor;
  endif;
end;
```

Figure 6: Pseudocode for procedure $R_1(x)$.

4 Computer Experiment Study

To compare the performance of PDGA over SGA, in this section we will give out the results of our computer experiment study. In all our experiments, we use the sigma truncation scaling [20] selection method together with the elitist model [9], one-point crossover and bit flip mutation operators for both PDGA and SGA. Here the genetic operators used are quite simple and typical for GAs. With the sigma truncation scaling selection method, an individual x's expected number of offspring is $1 + (f(x) - \bar{f})/(2\sigma)$, where $f(x)$ is x's fitness and $\bar{f}$ is the mean fitness of the population, and σ is the standard deviation. The elitist model retains the best individual in present generation into next generation.

4.1 Performance Measures

Here the performance measures have twofold functions: one is to test the absolute performance of PDGA itself as a genetic algorithm; the other is to compare the relative performance of PDGA over SGA. With this consideration, we use the following performance measures: function evaluations to optimum, generations to optimum, percentage of achieving optimum over a number of runs against generations, percentage of optimal members in the population for a run against generations. We will also test PDGA and SGA with respect to the performance of efficiency of function evaluation and schema recombination.

4.2 Experiments on Parameter Setting

It is well known that the parameter setting, including the population size N, the crossover probability P_c, the mutation probability P_m, is very important for GAs to work well. And it is very difficult to optimize the parameter setting for GAs.

In order to investigate the effects of parameter setting on the performance of PDGA over SGA, in our primary experiments, we consider different combinations of parameters N (128, 1024), P_c (0.6, 0.7, 0.8) and P_m (0.005, 0.01, 0.02) using the performance measures of function evaluations to optimum and generations to optimum. We carried out 200 runs of SGA and PDGA on R_1 and R_2 respectively using the same 200 random seeds for each parameter combination and recorded the function evaluations and generations required to obtain R_1's and R_2's optimal solutions. Here for each run only those chromosomes changed by crossover and mutation operations were evaluated and counted into the total number of function evaluations. The statistic results over 200 runs of PDGA and SGA on R_1 and R_2 with respect to mean, median and best (or least) evaluations to optimum are given in Table 1 and Table 2 respectively. And the statistic results over 200 runs of PDGA and SGA on R_1 and R_2 with respect to mean, median and best (or least) generations to optimum are given in Table 3 and Table 4 respectively. The results are shown more intuitively in Figure 7 and Figure 8. In Table 1 to Table 4 the numbers in parentheses are the standard errors.

From Table 1 to Table 4, Figure 7 and Figure 8 we can see that PDGA outperforms SGA on R_1 and R_2 with respect to both mean and best function evaluations to optimum and mean and best generations to optimum for most parameter settings. The best improvement of PDGA over SGA with respect to mean evaluations to optimum reaches about 25% with parameter setting 9 on R_1 and about 28% with parameter setting 18 on R_2.

From Table 1 to Table 4, Figure 7 and Figure 8 we can also see that the parameter setting does matter. Generally speaking the effect of parameter setting is similar for both R_1 and R_2. The setting of P_c obviously doesn't have much effect and shows little difference between 0.6, 0.7 and 0.8. However the setting of N and P_m has great effect and correlation exists between N and P_m. Generally speaking, greater population size 1024 is much better than smaller population size 128 (except on R_2 when $P_m = 0.02$). When N is fixed, the effect of setting P_m may be slightly different for SGA and PDGA.

When $N = 1024$, setting P_m to 0.005 is much better than to 0.01 or 0.02 for both SGA and PDGA on both R_1 and R_2. For example, fixing $P_c = 0.7$ with respect to mean function evaluations to optimum, on R_1 with $P_m = 0.005$ SGA gains about 16% improvement over $P_m = 0.01$ and about 59% over $P_m = 0.02$ and PDGA gains about 18% improvement over $P_m = 0.01$ and about 57% over $P_m = 0.02$, while on R_2 with $P_m = 0.005$ SGA gains about 38% improvement over $P_m = 0.01$ and about 92% over $P_m = 0.02$ and PDGA gains about 30% improvement over $P_m = 0.01$ and about 91% over $P_m = 0.02$. While when $N = 128$, on R_1 setting P_m to 0.01 is the best for both SGA and PDGA and on R_2 setting P_m to 0.01 is the best for SGA but setting P_m to 0.005 is the best for PDGA. For example, fixing $P_c = 0.7$ with respect to mean function evaluations to optimum, on R_1 with $P_m = 0.01$ SGA gains about 28% improvement over $P_m = 0.005$ and about 79% over $P_m = 0.02$ and PDGA gains about 36% improvement over $P_m = 0.01$ and about 77% over $P_m = 0.02$, while on R_2 with $P_m = 0.01$ SGA gains about 7% improvement over $P_m = 0.005$ and about 79% over $P_m = 0.02$ and with $P_m = 0.005$ PDGA gains about 7% improvement over $P_m = 0.01$ and about 81% over $P_m = 0.02$.

Another interesting result we can see from our experimental tables and figures is that both SGA and PDGA perform much better on R_1 than on R_2. This result was also observed and reported by Forrest and Mitchell in [14]. This result is opposite from what was expected that GA would perform better on R_2 than on R_1 because in R_2 there is a very clear path via crossover from pairs of the eight order-8 bottom schemas (s_1 – s_8) to the four intermediate order-16 schemas

Table 1: Statistics of running PDGA and SGA on R_1 with respect to evaluations to optimum.

Parameter Setting				SGA			PDGA		
No.	N	P_c	P_m	Mean(StdErr)	Median	Best	Mean(StdErr)	Median	Best
1	128	0.6	0.005	62439(4335)	43500	5063	57273(3010)	44558	6818
2	128	0.6	0.01	44328(2331)	34887	3094	37707(2122)	28906	3834
3	128	0.6	0.02	203417(28625)	83795	8133	210281(24972)	86854	4834
4	128	0.7	0.005	58598(4005)	42903	8091	58301(3939)	38869	4164
5	128	0.7	0.01	41927(2499)	29922	2846	37167(2310)	29445	3883
6	128	0.7	0.02	196284(20432)	85383	4587	164347(20127)	75300	5487
7	128	0.8	0.005	63944(3624)	49869	8272	55604(3222)	45380	2926
8	128	0.8	0.01	41134(2220)	31776	3733	38391(2909)	27299	2754
9	128	0.8	0.02	164875(19121)	71462	9105	123666(14773)	58861	4961
10	1024	0.6	0.005	21054(807)	17550	12233	17548(471)	16529	11968
11	1024	0.6	0.01	25641(816)	22654	13212	22581(528)	21270	12946
12	1024	0.6	0.02	55531(2919)	46181	21173	49221(1716)	43516	21463
13	1024	0.7	0.005	20234(543)	18485	11414	18328(406)	17254	12772
14	1024	0.7	0.01	24072(444)	22860	15424	22246(340)	21258	13555
15	1024	0.7	0.02	47036(1281)	41808	17593	42714(1257)	38513	18248
16	1024	0.8	0.005	21239(729)	18786	12735	18306(336)	17698	10900
17	1024	0.8	0.01	23992(373)	23199	15659	22811(405)	21868	11724
18	1024	0.8	0.02	44040(1199)	40813	17624	40948(1134)	36949	20126

Table 2: Statistics of running PDGA and SGA on R_2 with respect to evaluations to optimum.

Parameter Setting				SGA			PDGA		
No.	N	P_c	P_m	Mean(StdErr)	Median	Best	Mean(StdErr)	Median	Best
1	128	0.6	0.005	84491(6252)	60954	4343	67761(5494)	44491	2544
2	128	0.6	0.01	86583(11694)	47717	5291	76711(6422)	45221	3635
3	128	0.6	0.02	365759(43165)	138923	10805	471597(58060)	159367	9607
4	128	0.7	0.005	76212(5030)	57177	5072	73837(6965)	46326	4602
5	128	0.7	0.01	70939(6159)	45167	5831	79641(6978)	48161	3182
6	128	0.7	0.02	332978(35330)	140991	7305	389978(47715)	136044	11456
7	128	0.8	0.005	85816(10783)	53175	5223	71700(5477)	45047	3018
8	128	0.8	0.01	72881(6184)	43320	4505	69481(5014)	41624	2878
9	128	0.8	0.02	292604(40655)	120687	7322	276523(31078)	134745	5168
10	1024	0.6	0.005	30539(1594)	20456	10581	26236(1330)	17815	9199
11	1024	0.6	0.01	48487(2761)	33633	11825	39295(2492)	26419	11733
12	1024	0.6	0.02	348073(38911)	126787	18128	329780(33468)	144649	15376
13	1024	0.7	0.005	30178(1719)	19559	11577	27566(1656)	17194	10157
14	1024	0.7	0.01	48701(2946)	31890	12754	39277(2939)	22236	11577
15	1024	0.7	0.02	369536(39997)	188213	17373	316985(36096)	121280	17414
16	1024	0.8	0.005	26965(1288)	18988	10990	25085(1397)	17468	9918
17	1024	0.8	0.01	45463(2688)	29968	13438	34698(1852)	22869	11300
18	1024	0.8	0.02	329686(36216)	138917	15823	235858(25614)	72413	14419

Table 3: Statistics of running PDGA and SGA on R_1 with respect to generations to optimum.

Parameter Setting				SGA			PDGA		
No.	N	P_c	P_m	Mean(StdErr)	Median	Best	Mean(StdErr)	Median	Best
1	128	0.6	0.005	689(47)	481	55	631(33)	492	74
2	128	0.6	0.01	438(23)	345	30	373(21)	286	37
3	128	0.6	0.02	1785(251)	736	71	1846(219)	761	42
4	128	0.7	0.005	586(40)	428	81	583(39)	389	41
5	128	0.7	0.01	389(23)	277	26	344(21)	273	36
6	128	0.7	0.02	1671(174)	728	38	1399(171)	641	46
7	128	0.8	0.005	585(33)	456	76	509(29)	416	26
8	128	0.8	0.01	359(19)	277	32	335(25)	238	24
9	128	0.8	0.02	1363(158)	590	75	1022(122)	487	41
10	1024	0.6	0.005	28(1)	23	16	23(0)	22	16
11	1024	0.6	0.01	30(1)	27	16	27(0)	26	15
12	1024	0.6	0.02	60(3)	50	23	53(1)	47	23
13	1024	0.7	0.005	24(0)	22	14	22(0)	21	15
14	1024	0.7	0.01	27(0)	26	17	25(0)	24	15
15	1024	0.7	0.02	49(1)	44	18	44(1)	40	19
16	1024	0.8	0.005	23(0)	21	14	20(0)	20	12
17	1024	0.8	0.01	25(0)	25	16	24(0)	23	12
18	1024	0.8	0.02	44(1)	42	18	41(1)	38	20

Table 4: Statistics of running PDGA and SGA on R_2 with respect to generations to optimum.

Parameter Setting				SGA			PDGA		
No.	N	P_c	P_m	Mean(StdErr)	Median	Best	Mean(StdErr)	Median	Best
1	128	0.6	0.005	932(69)	674	47	747(60)	490	27
2	128	0.6	0.01	857(115)	472	52	759(63)	448	35
3	128	0.6	0.02	3211(379)	1218	94	4141(390)	1400	84
4	128	0.7	0.005	763(50)	573	49	739(69)	463	46
5	128	0.7	0.01	658(57)	417	54	739(64)	447	29
6	128	0.7	0.02	2836(301)	1200	62	3322(239)	1159	97
7	128	0.8	0.005	786(98)	487	47	656(50)	413	27
8	128	0.8	0.01	637(54)	379	39	607(43)	363	25
9	128	0.8	0.02	2420(72)	998	60	2287(257)	1114	42
10	1024	0.6	0.005	41(2)	27	14	35(1)	24	12
11	1024	0.6	0.01	59(3)	41	14	47(3)	32	14
12	1024	0.6	0.02	381(42)	138	19	361(36)	158	16
13	1024	0.7	0.005	36(2)	24	14	33(2)	21	12
14	1024	0.7	0.01	55(3)	36	14	44(3)	26	13
15	1024	0.7	0.02	392(42)	200	18	336(38)	129	18
16	1024	0.8	0.005	30(1)	21	12	28(1)	19	11
17	1024	0.8	0.01	49(2)	32	14	37(2)	24	12
18	1024	0.8	0.02	340(37)	143	16	243(26)	74	14

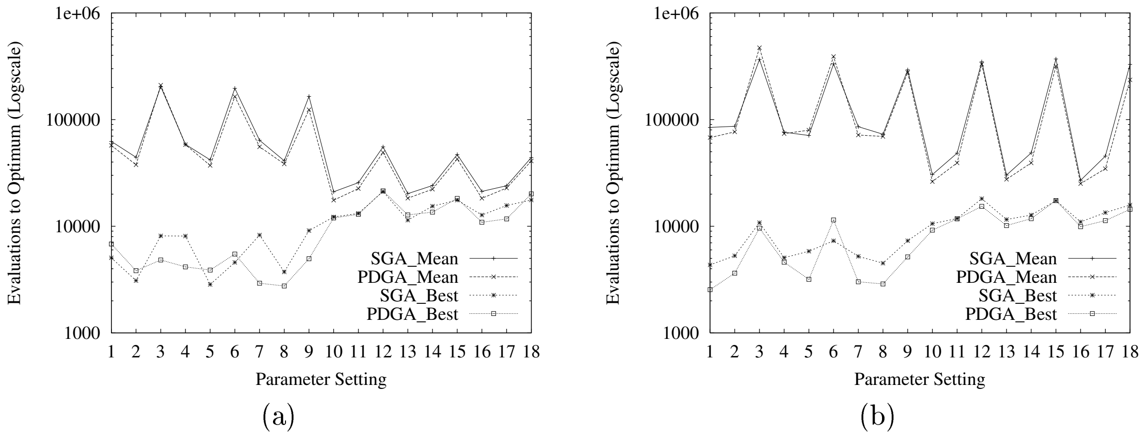


Figure 7: Effects of different parameter settings on the performance of PDGA vs. SGA with respect to mean and best function evaluations to optimum on (a) R_1 and (b) R_2.

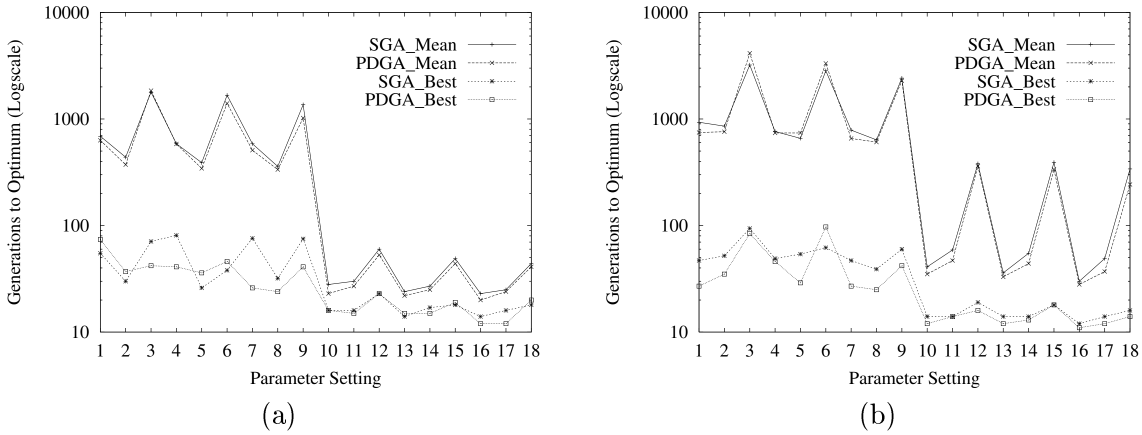


Figure 8: Effects of different parameter settings on the performance of PDGA vs. SGA with respect to mean and best generations to optimum on (a) R_1 and (b) R_2.

(s_9 – s_{12}), then to the two higher order-32 schemas (s_{13} and s_{14}), and finally to the optimum (s_{opt}). The presence of this stronger royal road was expected to speed up GA's searching for the optimum. The reason to the result opposite to the expectation lies in the phenomenon of "spurious correlation" or "hitchhiking". Hitchhiking was first discovered by Schaffer *et. al.* [43] and then discussed by Forrest and Mitchell [14], Vekaria and Clack [46] among others. Hitchhiking results from undesirable (less fit) schemas being sampled at rates that are not justified by their static fitnesses. These undesirable schemas have been coupled or hitchhiked along with desirable more fit schemas during recombination, thereby producing above average individuals, which later get sampled at a higher rate during selection. This may produce more instances of these undesirable schemas together with desirable schemas. Hitchhiking seriously limits GA's implicit parallelism and causes premature convergence. We will present further experiment results on hitchhiking later on in this section.

For our further experiment study, according to our primary experiment results, we fix the parameters as follows: $N = 128$, $P_c = 0.7$, $P_m = 0.01$. We set N to 128 instead of 1024 for the sake of better analyses and comparisons between PDGA and SGA because when $N = 1024$ they both converge quite fast with respect to generations to optimum, see Table 3, Table 4 and Figure 8. For example, when $N = 1024$, $P_c = 0.7$ and $P_m = 0.01$, SGA and PDGA converge to

Table 5: Comparison of PDGA vs. SGA with respect to the performance of percentage of achieving optimal solutions over 1000 runs against generations on R_1.

Gen	SGA	PDGA	Gen	SGA	PDGA	Gen	SGA	PDGA	Gen	SGA	PDGA
25	0.1	0.1	275	41.4	46.5	525	76.8	80.1	775	89.1	91.2
50	0.2	0.3	300	47.4	52	550	78.6	81.9	800	89.4	91.5
75	1	1.8	325	52.6	56.8	575	79.8	82.8	825	89.7	91.9
100	2.5	5.5	350	56.5	61.3	600	81.8	83.9	850	90.3	92.3
125	6.5	10.2	375	60.07	64.9	625	83	85.2	875	90.9	92.7
150	11.4	17.3	400	64.7	68.3	650	84.2	87.2	900	91.8	93
175	17.4	24.3	425	67.6	71.4	675	85	88.5	925	91.9	93.4
200	24.6	29.9	450	70.4	74.8	700	86.2	89.4	950	92.1	94
225	30.1	36.4	475	72.5	77.3	725	87.6	90.3	975	92.7	94.4
250	36.5	42	500	74.5	78.6	750	88	91.24	1000	93	94.8

Table 6: Comparison of PDGA vs. SGA with respect to the performance of percentage of achieving optimal solutions against generaions over 1000 runs on R_2.

Gen	SGA	PDGA	Gen	SGA	PDGA	Gen	SGA	PDGA	Gen	SGA	PDGA
25	0.1	0	275	29.3	31.9	525	56.7	59.5	775	71.8	73.6
50	0.1	0.2	300	32.6	35.5	550	59	61.2	800	72.2	74.1
75	0.5	0.9	325	35.1	38.9	575	60.9	62.7	825	72.9	74.9
100	1.4	2.9	350	38.2	42.6	600	62.7	64.7	850	74.3	75.8
125	3.6	6.2	375	40.9	45.5	625	64.1	66.4	875	75.7	76.4
150	6.4	9.2	400	43.9	48.1	650	65.6	67.8	900	76.9	77.1
175	11	12.8	425	47.1	50.5	675	67.2	69.1	925	77.9	77.9
200	16.2	16.9	450	49.3	53.4	700	68	70.5	950	78.6	78.4
225	20	22.6	475	52.1	55.3	725	68.7	71.7	975	79.4	78.9
250	24.5	26.6	500	55.1	57.6	750	70.3	72.7	1000	79.9	79.3

optimum on R_1 with 27 and 25 generations on the average respectively and on R_2 with 55 and 44 generations on the average respectively.

4.3 Experiments on Capacity of Achieving Optimum

Capacity of achieving optimum is obviously an important measure for evaluating an optimization algorithm. To compare the performance of achieving optimum against generations of PDGA over SGA, we carried out 1000 runs of PDGA and SGA on R_1 and R_2 under the same 1000 random seeds and with t_{max} set to 1000 for each run. For each run the information about whether the optimum is achieved and the percentage of optima (if any) in the population by current generation is reported every 25 generations. The statistic results with respect to the percentage of achieving optimum over 1000 runs against generations are given in Table 5 and Table 6 respectively. The statistic results with respect to the mean percentage of optimal individuals in the population over 1000 runs against generations on R_1 and R_2 are given in Table 7 and table 8 respectively. Figure 9 and Figure 10 show the results more intuitively.

From Figure 9 and Figure 10, we can see that PDGA outperforms SGA on both R_1 and R_2 for both performance measures of achieving optimum. For example, with 500 generations, out of the 1000 runs PDGA achieved R_1's optimum for 786 runs (78.6%) and R_2's optimum for

Table 7: Comparison of PDGA vs. SGA with respect to the performance of mean percentage of optimal individuals in the population against generaions over 1000 runs on R_1.

Gen	SGA	PDGA	Gen	SGA	PDGA	Gen	SGA	PDGA	Gen	SGA	PDGA
25	0.02	0.02	275	10.80	12.27	525	20.92	21.88	775	24.23	24.97
50	0.03	0.04	300	12.19	13.51	550	21.69	22.21	800	24.86	24.81
75	0.16	0.38	325	13.90	14.93	575	21.46	22.41	825	24.62	25.26
100	0.41	1.06	350	15.42	16.45	600	22.48	22.88	850	24.42	25.55
125	1.32	2.24	375	16.09	17.49	625	22.73	23.20	875	24.53	25.05
150	2.51	3.65	400	17.22	18.38	650	23.00	23.75	900	24.71	25.31
175	4.26	5.86	425	18.50	19.30	675	23.48	24.17	925	25.20	25.58
200	6.02	7.45	450	18.80	19.95	700	23.87	24.18	950	25.45	25.45
225	7.59	9.23	475	19.57	20.54	725	24.21	24.88	975	25.11	25.68
250	9.47	11.11	500	20.57	21.55	750	23.85	24.90	1000	25.43	25.77

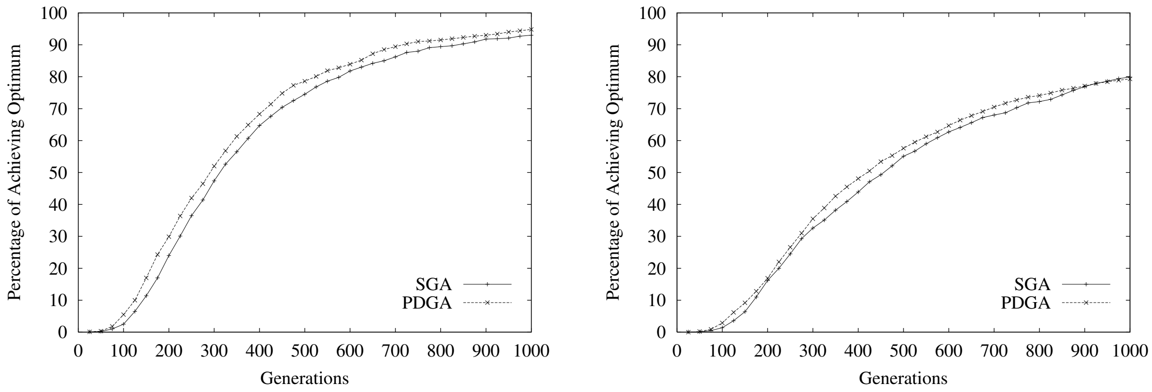
Table 8: Comparison of PDGA vs. SGA with respect to the performance of mean percentage of optimal individuals in the poputlation against generaions over 1000 runs on R_2.

Gen	SGA	PDGA	Gen	SGA	PDGA	Gen	SGA	PDGA	Gen	SGA	PDGA
25	0.02	0	275	9.55	9.87	525	19.1	19.97	775	24.05	24.87
50	0.02	0.03	300	10.97	11.33	550	19.76	20.74	800	24.73	25.27
75	0.09	0.21	325	11.71	12.62	575	20.47	21.49	825	24.89	25.06
100	0.34	0.73	350	12.34	13.54	600	21.28	21.54	850	25.04	25.88
125	0.97	1.77	375	13.49	15.38	625	21.44	22.27	875	25.68	26.12
150	2.01	2.75	400	14.79	16.15	650	22.06	22.94	900	25.86	25.91
175	3.13	3.82	425	15.58	16.76	675	22.43	23.20	925	26.57	26.18
200	5.06	5.25	450	16.51	17.69	700	23.06	23.86	950	26.82	26.69
225	6.31	6.90	475	17.34	18.77	725	23.20	24.06	975	26.77	26.59
250	7.85	8.77	500	18.48	19.20	750	23.26	24.91	1000	27.49	26.71

576 runs (57.6%) while SGA achieved R_1's optimum for 745 runs (74.5%) and R_2's optimum for 551 runs (55.1%). And with 500 generations, the percentage of optimal individuals in the population averaged over 1000 runs for PDGA is 21.55% on R_1 and 19.20% on R_2 while for SGA it is 20.57% on R_1 and 18.48% on R_2. From Figure 9 and Figure 10, we can also see that both PDGA and SGA perform better on R_1 than on R_2. For example, with 1000 generations, both PDGA and SGA achieve R_1's optimum with a percentage around 94% (93.0% for SGA and 94.8% for PDGA) while achieve R_2's optimum with a percentage around 80% (79.9% for SGA and 79.3% for PDGA).

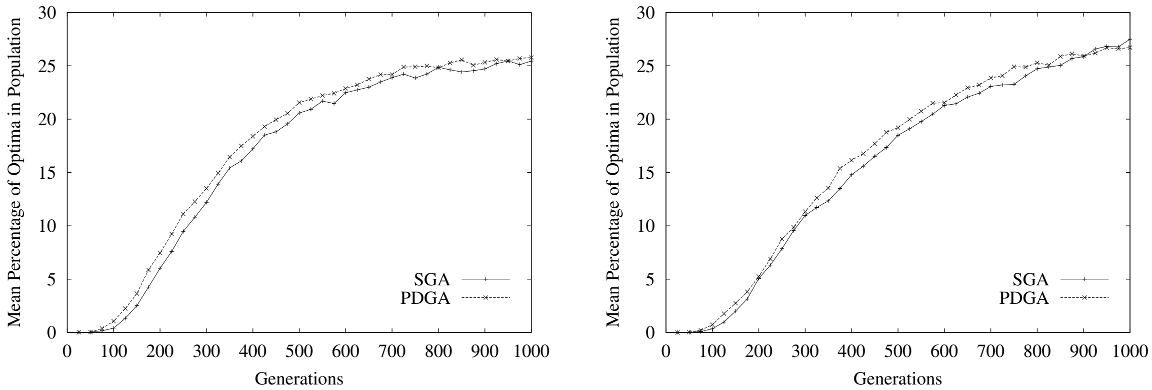
4.4 Experiments on Efficiency of Evaluation

Efficiency of function evaluation is another important measure that is often used among GA's community to compare the dynamic performance of GAs. To compare the efficiency of function evaluation, we carried out 1000 runs of PDGA over SGA on R_1 and R_2 respectively with the same 1000 random seeds for each combination of algorithm and function. We record the best fitness found every 100 evaluations for each run. Here, as with our parameter setting experiments, only those chromosomes that were changed by crossover and mutation operations were evaluated and counted. The statistic results over 1000 runs are shown in Figure 11. From Figure 11 we can



(a) (b)

Figure 9: Comparisons of PDGA vs. SGA with respect to the performance of percentage of achieving optimum over 1000 runs against generations on (a) R_1 and (b) R_2.



(a) (b)

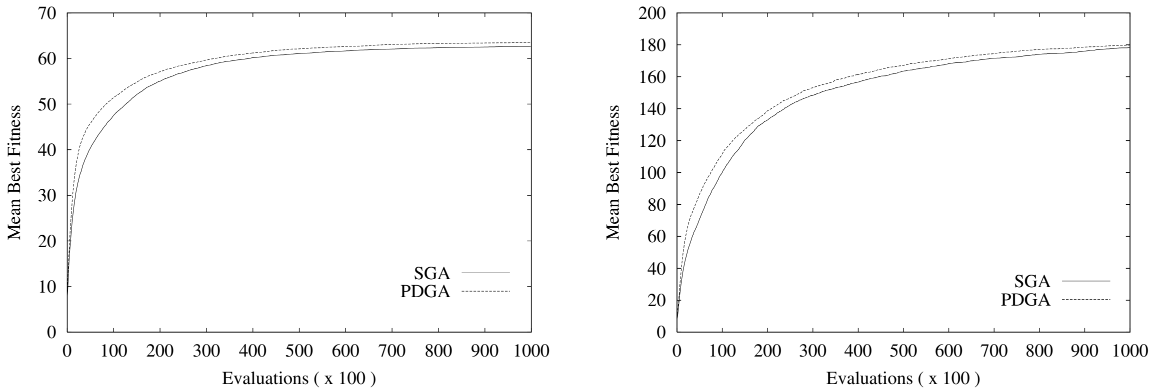
Figure 10: Comparisons of PDGA vs. SGA with respect to the performance of mean percentage of optimal individuals in the population against generaions over 1000 runs on (a) R_1 and (b) R_2.

see again that PDGA overruns SGA quite well, especially during the early generations.

4.5 Experiments on Schema Recombination

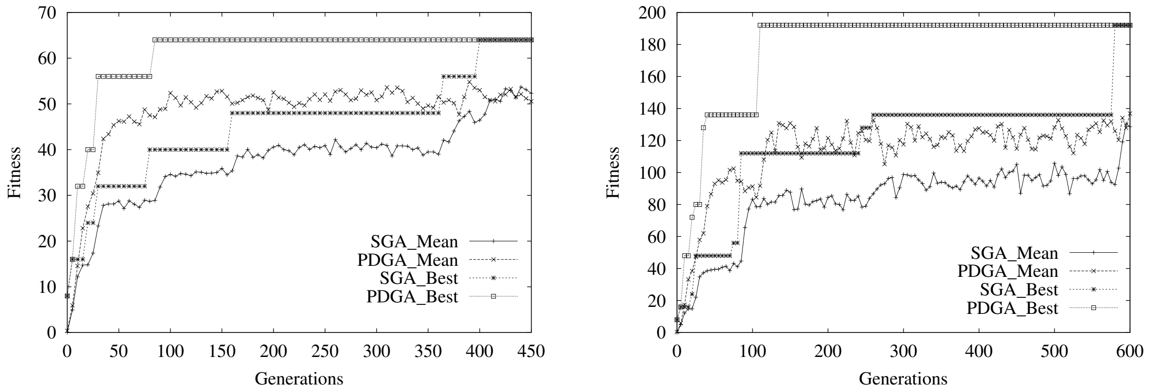
In our previous experiments we have seen that both PDGA and SGA perform better on R_1 than on R_2. To unveil the reason for this result and further compare the dynamic searching process of PDGA and SGA, we carried out experiments with respect to the dynamic behavior of schema processing and recombination. We give out the results of a typical run of PDGA and SGA on R_1 and R_2 respectively with respect to mean and best fitness achieved against generations in Figure 12. For the typical runs, the random seeds used are the same. That is, they started from the same initial population. The data are plotted every 5 generations.

From Figure 12 we can see that PDGA overruns SGA quite well especially during the early generations. PDGA obtained R_1's near-optimal fitness 56 (only next to optimum 64) within only 35 generations and R_2's near-optimal fitness 136 (only next to optimum 192) within only 45 generations while SGA obtained R_1's near-optimal fitness 56 within 365 generations and R_2's near-optimal fitness 136 within 245 generations. From Figure 12 we can also see that both PDGA and SGA perform better on R_1 than on R_2. PDGA achieves R_1's optimum at generation



(a) (b)

Figure 11: Comparisons of PDGA vs. SGA with respect to mean best fitness over 1000 runs against evaluations on (a) R_1 and (b) R_2.



(a) (b)

Figure 12: Mean and best fitness against generations for a typical run of PDGA vs. SGA on (a) R_1 and (b) R_2.

85 with 9252 evaluations while achieves R_2's optimum at generation 108 with 11784 evaluations. And SGA achieves R_1's optimum at generation 400 with 43264 evaluations while achieves R_2's optimum at generation 578 with 62360 evaluations.

For the above typical runs, we also traced the evolution of each schema and recorded the density of each schema (the percentage of individuals in the population that are instances of each schema) against time (generations). The density of each schema is sampled every 5 generations. Figure 13 and Figure 14 show the corresponding schema recombination process on R_1 (of the same typical run as in Figure 12(a)) and on R_2 (of the same typical run as in Figure 12(b)) respectively. From Figure 13 and Figure 14 we can see that both PDGA and SGA really process useful schemas according to the schema theorem: once a schema is found in the population, its density in the population rises very fast (exponentially) to about 90% except for some occasions (e.g., schemas s_6 and s_7 with SGA in Figure 13) due to the phenomenon of hitchhiking.

From Figure 13 and Figure 14 we can also see that both PDGA and SGA suffer from hitchhiking more heavily on R_2 than on R_1. This result can be observed from the fact that the schema density curves (e.g., s_5 and s_8) oscillate much more heavily in Figure 14 than in Figure 13. This heavy oscillation happens because the introduction of intermediate schemas s_9 to s_{14} in R_2 gives the hitchhikers (those zeros that live near and co-evolute with the higher-fit schemas)

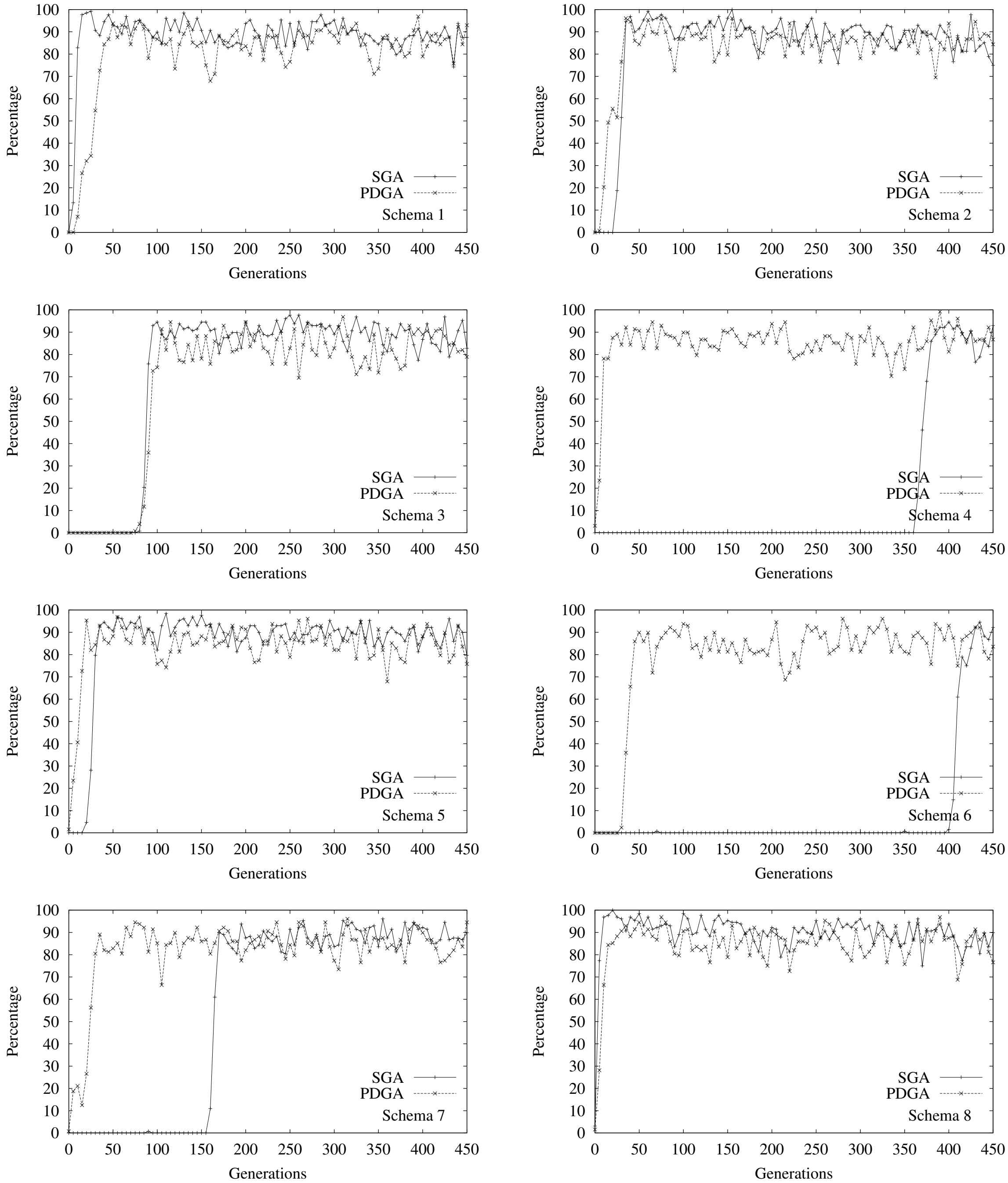


Figure 13: Percentage of individuals in the population that are instances of the given schema against generations of a typical run of PDGA vs. SGA on R_1.

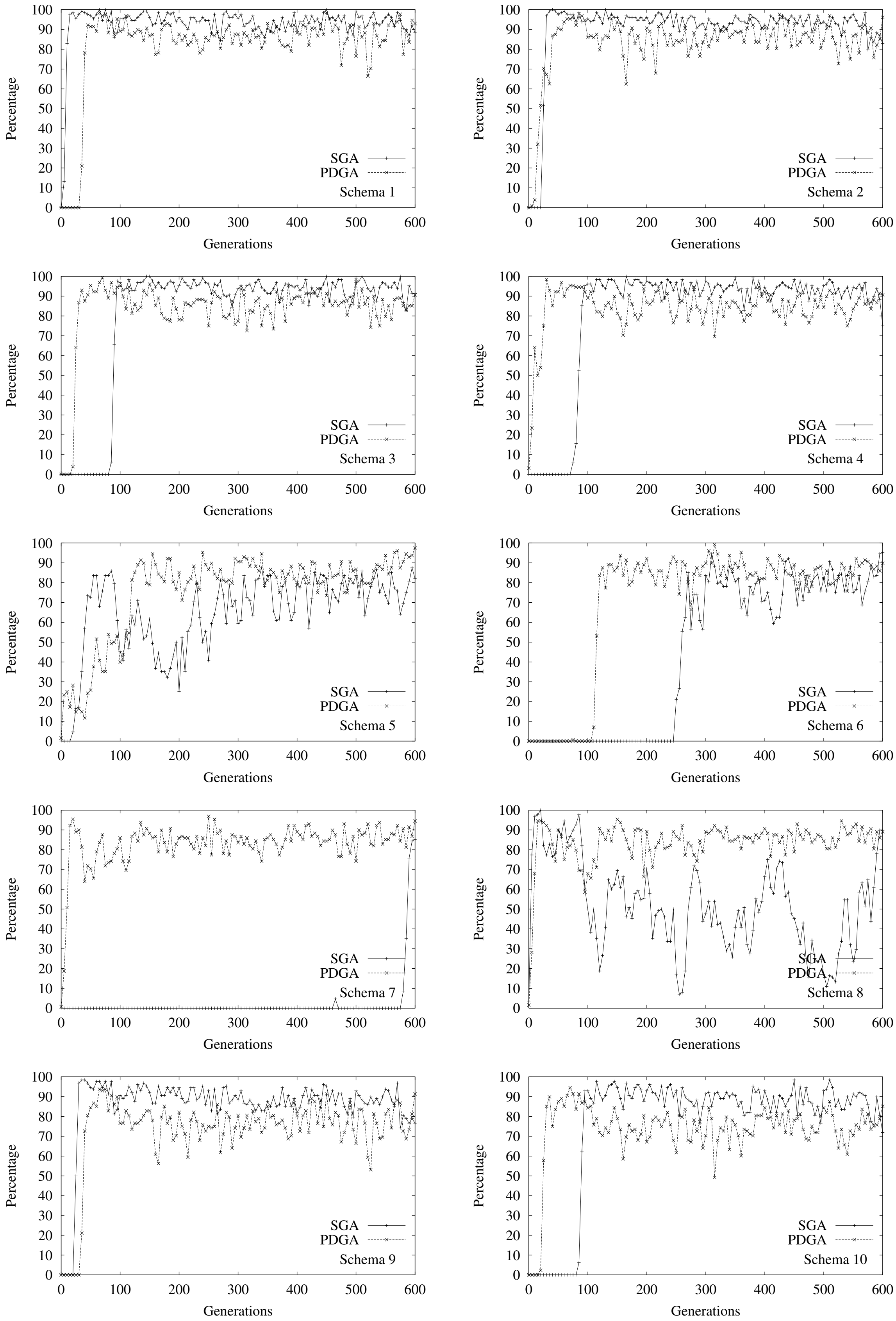


Figure 14: Percentage of the population that is an instance of the given schema plotted against generations for a typical run of SGA and PDGA on R_2.

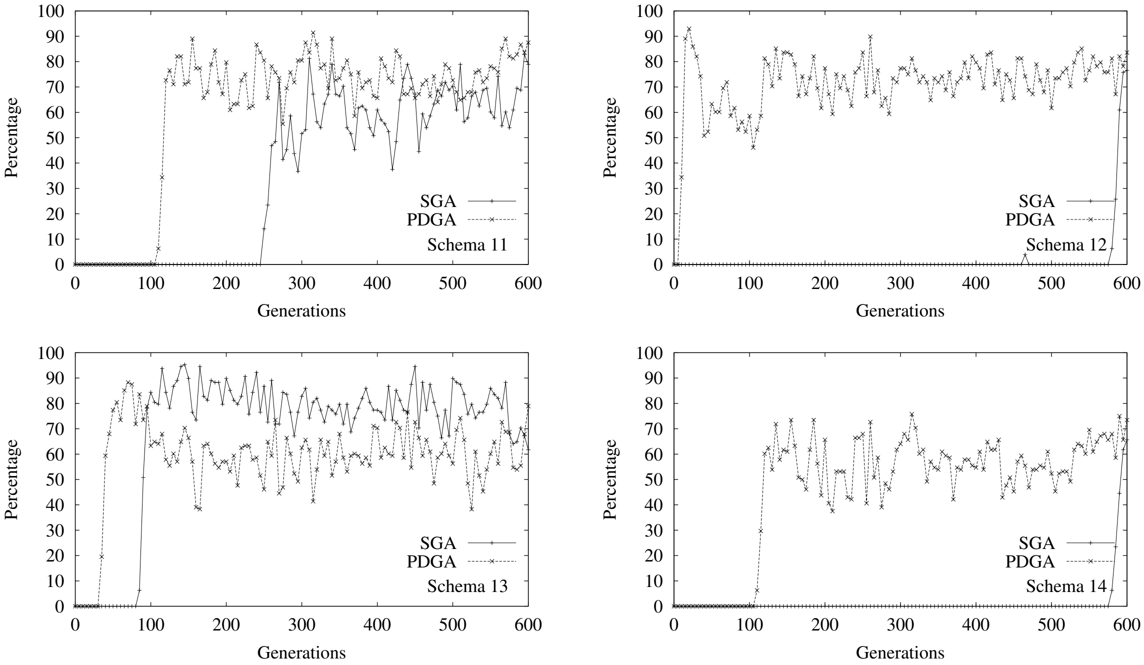


Figure 14: (CONT'D.) Percentage of the population that is an instance of the given schema plotted against generations for a typical run of SGA and PDGA on R_2.

chances to hitchhike with them to a much heavier degree.

For example, for SGA on R_2, with the rapid rise of s_3 and s_4 from around generation 80 to generation 100, s_{10} (which comprise of s_3 and s_4) and s_{13} (which comprise of s_1 through s_4) also rise rapidly. These rises coincide with the major dips in s_5 and s_8 (s_6 and s_7 aren't discovered yet). What happens here is as following: in the first few instances of s_{13}, along with the 32 ones in positions corresponding to the first through fourth blocks of the eight lowest-level building blocks are some zeros in the fifth through eighth blocks. Because an instance of s_{13} has fitness $32 + 2 * 16 + 4 * 8 = 96$ which is much higher than the fitness of an instance of s_5 or s_8 which is only 8, this great fitness difference causes s_{13} to rise very quickly compared to s_5 and s_8, and instances of s_{13} with some zeros (hitchhikers) in the fifth or eighth block tend to push out many of the existing instances of s_5 and s_8 in the population. Similarly, for SGA on R_2, from around generation 245 schema s_6 and thus subsequently s_{11} rise very rapidly together with the major dip in s_8. This happens because some zeros in the eighth block hitchhike with s_{11} and squash out many of the existing instances of s_8 in the population. For SGA on R_2, hitchhiking also happens with s_7 and thus with s_{12} and s_{14} that contain s_7. Schema s_7 is first suppressed by s_8 and then by s_{13}, is fleetingly discovered at around generation 460 (see the blip on the x-axis of s_7) due to the minor dips of s_8 and s_{11}, but then dies out and is depressed due to the rise of s_{11} and only appears (so do s_{12} and s_{14}) not until around generation 578 to obtain the optimum.

Hitchhiking also happens with PDGA on R_2. Schema s_6 is first suppressed by s_5 and s_7, is fleetingly discovered at around generation 75 (see the blip on the x-axis of s_6) due to the dips of s_5 and s_7, but then dies out due to hitchhiking and only appears (so do schemas s_{11} and s_{14}) until around generation 108 to obtain the optimum.

As shown in Figure 13, hitchhiking also happens on R_1 but not as heavily as on R_2. For example, with SGA s_6 appears at around generation 70 and generation 350 but is suppressed by the rapid rise of s_3 and s_4 respectively and re-appears at around generation 400 to obtain the optimum.

Now from above analyses we can see that the effect of hitchhiking causes the relatively slower times for the PDGA and SGA to find the optimum on R_2 than on R_1. The power of crossover to combine lower-level building blocks was hampered since some of the necessary building blocks were either partially or totally suppressed by the quick rise of disjoint building blocks.

Another result we can observe from Figure 13 and Figure 14 is that SGA suffers more heavily from hitchhiking than PDGA. For example, on R_1 for PDGA schema s_3 is first suppressed by s_2 and s_4 but is discovered around generation 80 to obtain the optimum while for SGA schema s_6 is first suppressed by s_5 and then by s_7, is fleetingly discovered at around generation 70 and generation 350 but is suppressed by the rapid rise of s_3 and s_4 respectively and only re-appears not until around generation 400 to obtain the optimum.

5 Discussions and Analyses

From above experiments we can see that PDGA outperforms SGA, especially during the early stage of GA's searching process. In this section, we give out our explanations and analyses to this result. We have said that PDGA is proposed with the aim of improving GA's exploration capacity in the search space through the primal-dual mapping. Here the mapping function has the key role in improving PDGA's performance. We illustrate the effects of the primal-dual mapping in Figure 15, where the attribute(s) axis represents the combination of different attributes (e.g., schemas for Royal Road functions) of a chromosome, and the primal-dual mapping is illustrated by the virtual mapping curve that maps a primal chromosome into its dual chromosome.

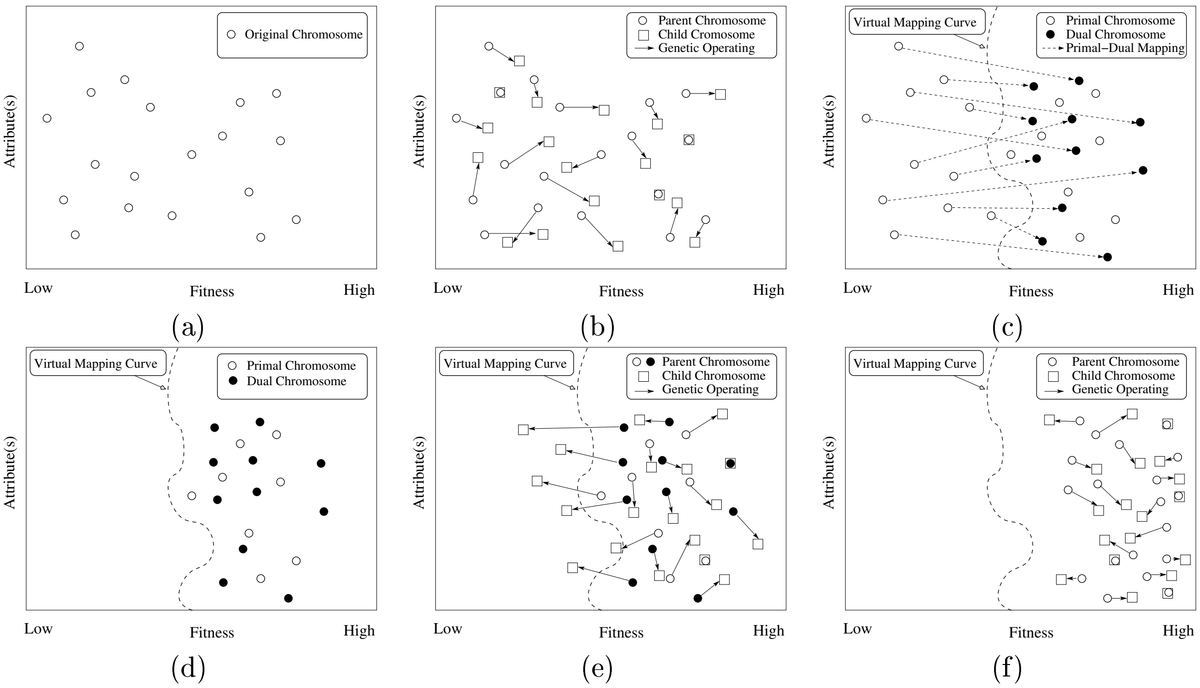


Figure 15: Illustration of the effects of the primal-dual mapping. (a) original population; (b) genetic operating without primal-dual mapping; (c) for population in (a), mapping primal chromosomes with fitness left to the virtual mapping curve into their dual ones; (d) after the primal-dual mapping; (e) genetic operating on the mapped population; (f) after certain generations the primal-dual mapping has little effect.

Figure 15(a) illustrates an original population during the early generation of GA's searching progress, where there are many individuals with low fitness. With SGA the genetic operations perform directly on the population without the primal-dual mapping, as shown in Figure 15(b);

while with PDGA before genetic operations the primal-dual mappings (only valid prima-dual mappings are illustrated) map those chromosomes with fitness left to the virtual mapping curve, i.e., low fitness, into (and replace them with) their dual ones that have high fitnesses and are right to the virtual mapping curve (see Figure 15(c)) and the genetic operations now perform on the mapped better population (see Figure 15(d)), as shown in Figure 15(e).

From Figure 15 (a) to Figure 15(e) we can see that the primal-dual mapping greatly improves GA's performance during the early generations. This happens because during the early stage, the mean fitness of the population is quite low and many primal-dual mappings are valid. However, after certain generations the primal-dual mapping has little effect because the mean fitness of the population has become quite high and new chromosomes created by the genetic operations seldom fall into the left side of the virtual mapping curve, as shown in Figure 15(f). That is, by now most primal-dual mappings are invalid.

In our experiments another result we observed is that the primal-dual mapping also helps moderating the hitchhiking phenomenon though PDGA still suffers from it. This is because the primal-dual mapping improves the diversity of the population via exploration in the search space during the early stage of GA's searching progress.

6 Conclusions and Future Directions

In this paper, inspired by the phenomenon of diploid genotype and dominance mechanisms broadly existing in nature, we propose a new primal-dual genetic algorithm which operates on a pair of chromosomes that are primal-dual to each other in the sense of Hamming distance. We have compared the performance of PDGA over SGA based on the Royal Road functions. The experiment results show that PDGA overperforms SGA on the Royal Road functions for different performance measures. The Royal Road functions are a class of fitness landscapes that are designed to test GA's performance, especially with respect to schema processing and recombination. In this paper, we take the Royal Road functions as the test problems to make our first step of testing PDGA. In the future we will further test PDGA's performance on other classes of fitness landscapes, such as the NK model [27] and the L-SAT problems [10].

PDGA is proposed with the aim of improving GA's searching efficiency in the search space through the primal-dual mapping. Through the primal-dual mapping, PDGA's performance of exploration in the search space is improved and thus its total searching efficiency is improved. Here the mapping function has the key role in PDGA's performance. In this paper we take the Hamilton distance as the primal-dual mapping function, which is a static mapping function. This mapping function works well during the early generations by shortening genetic operations performed on low fitness chromosomes and thus speed up GA's convergence. However, whence the mean fitness of the population becomes quite high, it lose its effect. For the future research on PDGA, we believe that dynamic primal-dual mapping function instead of the static Hamming mapping that can adapt itself with GA's searching progress (e.g., the virtual mapping curve in Figure 15 moves to the right together with the mean fitness of the population) will further improve PDGA's performance.

From the viewpoint of structure, PDGA shares the same framework with SGA, thus there can be many variations for PDGA as well as for SGA. Whatever improvements and variations (for example, the elitist model and the Sigma truncation scaling selection method used in this paper) that work well with SGA should also work well with PDGA. Thus combining advanced operators developed so far for SGAs with PDGA to achieve even better performance is obviously one of the future research directions on PDGA.

References

[1] F. Allen and R. Karjalainen, "Using genetic algorithms to find technical trading rules," *Journal of Financial Economics*, vol. 51, pp. 245-271, 1999.

[2] H. J. Antonisse, "A new interpretation of schema notation that overturns the binary encoding constraint," in *Proc. of the 3rd Int. Conf. on Genetic Algorithms*, J. Schaffer, Ed., 1989, pp. 86-91, San Mateo, CA: Morgan Kaufmann Publishers.

[3] W. Banzhaf, P. Nordin, R. E. Keller, and F. Francone, *Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and Its Applications*, Morgan Kaufmann Publishers, 1998.

[4] J. D. Bagley, *The behavior of adaptive systems witch employ genetic and correlation algorithms*, Doctoral Dissertation, University of Michigan, 1967.

[5] J. E. Baker, "Adaptive selection methods for genetic algorithms," in *Proc. of the 1st Int. Conf. on Genetic Algorithms and Their Applications*, J. J. Grefenstette, Ed., Pittsburgh, PA, July 24-26, 1985, pp. 101-111, Hillsdale, NJ: Lawrence Eribaum Associates.

[6] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm," in *Proc. of the 2nd Int. Conf. on Genetic Algorithms*, J. J. Grefenstette, Ed., Cambridge, MA, 1987, pp. 14-22, Hillsdale, NJ: Lawrence Eribaum Associates.

[7] A. Brindle, *Genetic Algorithms for Function Optimization*, Doctoral Dissertation, University of Alberta, Edmonton, Canada, 1981.

[8] L. Davis, "Jobshop scheduling with genetic algorithms," in *Proc. of the 1st Int. Conf. on Genetic Algorithms and Their Applications*, J. J. Grefenstette, Ed., Pittsburgh, PA, July 24-26, 1985, pp. 136-140, Hillsdale, NJ: Lawrence Eribaum Associates.

[9] K. A. De Jong, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Doctoral Dissertation, University of Michigan, 1975.

[10] K. A. De Jong, M. A. Potter and W. M. Spears, "Using problem generators to explore the effects of epistasis," in *Proc. of the 7th Int. Conf. on Genetic Algorithms*, T. Bäck, Ed., July, 1997, pp. 338-345, Morgan Kaufmann Publishers.

[11] L. J. Esheman and J. D. Schaffer, "Preventing premature convergence in genetic algorithms by preventing incest," in *Proc. of the 4th Int. Conf. on Genetic Algorithms*, R. Belew and L. Booker, Eds., San Diego, USA, 1991, pp. 115-122, San Mateo, CA: Morgan Kaufmann Publishers.

[12] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence Through Simulated Evolution*, John Wiley, Chichester, UK, 1966.

[13] C. M. Fonseca, P. J. Fleming, "Genetic algorithms for multiobjective optimization: formulation, discussion and generalization," in *Proc. of the 5th Int. Conf. on Genetic Algorithms*, S. Forest, Ed., Urbana-Champaign, USA, July 17-21, 1993, pp. 416-423, San Mateo, CA: Morgan Kaufmann Publishers.

[14] S. Forrest and M. Mitchell, "Relative building-block fitness and the building-block hypothesis," in *Foundations of Genetic Algorithms 2*, L. D. Whitley, Ed., San Mateo, California, 1993, Morgan Kaufmann Publishers.

[15] M. Gen, Y. Tsujimura, and E. Kubota, "Solving job-shop scheduling problem using genetic algorithms," in *Proc. of the 16th Int. Conf. on Computers and Industrial Engineering*, M. Gen and T. Kobayashi, Eds., Ashikaga, Japan, 1994, pp. 576-579.

[16] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Design*, John Wiley & Sons, 1997.

[17] D. E. Goldberg and R. Lingle, "Alleles, loci, and the TSP," in *Proc. of the 1st Int. Conf. on Genetic Algorithms and Their Applications*, J. J. Grefenstette, Ed., Pittsburgh, PA, July 24-26, 1985, pp. 154-159, Hillsdale, NJ: Lawrence Eribaum Associates.

[18] D. E. Goldberg and J. Richardson, "Genetic algorithms with sharing for multimodal function optimization," in *Proc. of the 2nd Int. Conf. on Genetic Algorithms*, J. J. Grefenstette, Ed., Cambridge, MA, 1987, pp. 41-49, Hillsdale, NJ: Lawrence Eribaum Associates.

[19] D. E. Goldberg and R. E. Smith, "Nonstationary function optimization using genetic algorithms with dominance and diploidy," in *Proc. of the 2nd Int. Conf. on Genetic Algorithms*, J. J. Grefenstette, Ed., Cambridge, MA, 1987, pp. 59-68, Hillsdale, NJ: Lawrence Eribaum Associates.

[20] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Reading, MA: Addison-Wesley, 1989.

[21] D. E. Goldberg, "Real-coded Genetic Algorithms, virtual alphabets, and blocking", *Technical Report No. 90001*, University of Illinois at Urbana-Champaign, September 1990.

[22] M. Gorges-Schleuter, "ASPARAGOS an asynchronous parallel genetic optimization strategy" in *Proc. of the 3rd Int. Conf. on Genetic Algorithms*, J. Schaffer, Ed., 1989, pp. 422-427, San Mateo, CA: Morgan Kaufmann Publishers.

[23] R. Hinterding, "Mapping, order-independent genes and the knapsack problem," in *Proc. of the 1st IEEE Conf. on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, Z. Michalwicz, Ed., Orlando, FL, July 27-29, 1994, vol. 1, pp. 13-17, Piscataway, NJ: IEEE Service Center.

[24] J. H. Holland, *Adaptation in Natural and Artificial Systems*, Ann Arbor, University of Michigan Press, 1975.

[25] R. B. Hollstien, *Artificial Ggenetic Aadaptation in Computer Control Systems*, Doctoral Dissertation, University of Michigan, 1971.

[26] J. Horn, N. Nafpliotis, and D. E. Goldberg, "A niched pareto genetic algorithm for multiobjective optimization," in *Proc. of the 1st IEEE Conf. on Evolutionary Computation, IEEE world Congress on Computational Intelligence*, Z. Michalwicz, Ed., Orlando, FL, July 27-29, 1994, vol. 1, pp. 82-87, Piscataway, NJ: IEEE Service Center.

[27] S. A. Kauffman, "Adaptation on rugged fitness landscapes," *Lectures in the Sciences of Complexity*, D. L. Stein, Ed., vol. 1, pp. 527-618, Addison Wesley, 1989.

[28] K. Kristinsson and G. A. Dumont, "System identification and control using genetic algorithms," *IEEE Trans. on Systems, Man and Cybernetics*, vol. 22, no. 5, pp. 1033-1046, 1992.

[29] J. R. Koza, *Genetic Programming: On the Programming of Computers by Natural Selection*, Cambridge, MA: MIT Press, 1992.

[30] P. J. M. van Laarhoven and E. H. L. Aarts, *Simulated Annealing: Theory and Applications*, D. Reidel, Dordrecht, Holland, 1987.

[31] J. Lewis, E. Hart and G. Ritvhie, "A comparison of dominance mechanisms and simple mutation on non-stationary problems," in *Parallel Problem Solving from Nature: PPSN V*, F. E. Eiben, T. Back, M. Schoenauer and H-P. Schwefel, Eds., 1998, pp. 139-148, Springer-Verlag.

[32] K. F. Man, K. S. Tang, S. Kwong and W. A. Halang, *Genetic Algorithms for Control and Signal Processing*, Springer-Verlag, London, 1997.

[33] P. Merz and B. Freisleben, "Fitness landscape analysis and memetic algorithms for the quadratic assignment problem," *IEEE Trans. on Evolutionary Computation*, vol. 4, no. 4, pp. 337-352, 2000.

[34] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd edition, Springer-Verlag, New York, 1996.

[35] M. Mitchell, S. Forrest, and J. H. Holland, "The royal road for genetic algorithms: fitness landscapes and GA performance," in *Proc. of the 1st European Conf. on Artificial Life*, F. J. Varela and P. Bourgine, Eds., 1992, pp. 245-254, Cambridge, MA: MIT Press.

[36] M. Mitchell, J. H. Holland, and S. Forrest, "When will a genetic algorithm outperform hill climbing?" in *Advances in Neural Information Processing Systems 6*, J. D. Cowan, G. Tesauro, and J. Alspector, Eds., 1994, Morgan Kaufmann Publishers.

[37] M. Mitchell, *An Introduction to Genetic Algorithms*, Cambridge, MA: MIT Press, 1996.

[38] H. Mühlenbein, "Parallel genetic algorithms, population genetics and combinatorial optimization," in *Proc. of the 3rd Int. Conf. on Genetic Algorithms*, J. Schaffer, Ed., 1989, pp. 416-421, San Mateo, CA: Morgan Kaufmann Publishers.

[39] K. P. Ng and K. C. Wong, "A new diploid scheme and dominance change mechanism for non-stationary function optimization," in *Proc. of the 6th Int. Conf. on Genetic Algorithms*, 1995, pp. 159-166, San Mateo, CA: Morgan Kaufmann Publishers.

[40] I. Rechenberg, *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*, Frommann-Holzboog Verlag, Stuttgat, 1973.

[41] E. Ronald, "When selection meets seduction," in *Proc. of the 6th Int. Conf. on Genetic Algorithms*, L. J. Eshelman, Ed., Pittsburgh, PA, 1995, pp. 167-173, San Mateo, CA: Morgan Kaufmann Publishers.

[42] D. J. Schaffer, "Multiple objective optimization with vector evaluated genetic algorithms," in *Proc. of the 1st Int. Conf. on Genetic Algorithms and Their Applications*, J. J. Grefenstette, Ed., Pittsburgh, PA, July 24-26, 1985, pp. 93-100, Hillsdale, NJ: Lawrence Eribaum Associates.

[43] D. J. Schaffer, L. J. Eshelman and D. Offnut, "Spurious correlations and premature convergence in genetic algorithms," in *Foundations of Genetic Algorithms 1*, G. Rawlins, Ed., 1991, pp. 102-112, Morgan Kaufmann Publishers.

[44] H.-P. Schwefel, *Numerical Optimization for Computer Models*, John Wiley, Chichester, UK, 1981.

[45] N. Srinivas and K. Deb, "Multiobjective optimization using nondominated sorting genetic algorithm," *Evolutionary Computation*, vol. 2, no. 3, pp. 221-248, 1995.

[46] K. Vekaria and C. Clack, "Hitchhikers get around," in *Proc. of Artificial Evolution*, 1999, November 3-5, JIL, Universite du Littoral, Dunkarque France.

[47] D. Whitley, "GENITOR II: a distributed genetic algorithm," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 2, pp. 189-214, 1990.

[48] S. Wright, "Evolution in mendelian populations," *Genetics*, vol. 16, pp. 97-159, 1931.

[49] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach," *IEEE Trans. on Evolutionary Computation*, vol. 3, no. 4, pp. 257-271, 1999.