Appl Intell (2013) 39:510–528
DOI 10.1007/s10489-013-0429-5

A hierarchical parallel genetic approach for the graph coloring problem

Reza Abbasian · Malek Mouhoub

Published online: 3 March 2013

Abstract Graph Coloring Problems (GCPs) are constraint optimization problems with various applications including time tabling and frequency allocation. The GCP consists in finding the minimum number of colors for coloring the graph vertices such that adjacent vertices have distinct colors. We propose a hierarchical approach based on Parallel Genetic Algorithms (PGAs) to solve the GCP. We call this new approach Hierarchical PGAs (HPGAs). In addition, we have developed a new operator designed to improve PGAs when solving constraint optimization problems in general and GCPs in particular. We call this new operator Genetic Modification (GM). Using the properties of variables and their relations, GM generates good individuals at each iteration and inserts them into the PGA population in the hope of reaching the optimal solution sooner. In the case of the GCP, the GM operator is based on a novel Variable Ordering Algorithm (VOA) that we propose. Together with the new crossover and the estimator of the initial solution we have developed, GM allows our solving approach to converge towards the optimal solution sooner than the well known methods for solving the GCP, even for hard instances. This was indeed clearly demonstrated by the experiments we conducted on the GCP instances taken from the well known DIMACS website.

Keywords Parallel genetic algorithms · Graph coloring problem

R. Abbasian · M. Mouhoub (✉)
Department of Computer Science, University of Regina,
Regina S4S 0A2, Canada
e-mail: mouhoubm@cs.uregina.ca

R. Abbasian
e-mail: abbasiar@cs.uregina.ca

1 Introduction

Graph Coloring Problems (GCPs) are very interesting constraint optimization problems with many real world applications such as Frequency Allocation for Mobile Radio and WLANs [32], Register Allocation [6], Time Tabling [33] and Scheduling [21]. A GCP on a given graph G is defined as finding the graph's chromatic number denoted by $\chi(G)$. $\chi(G)$ is the minimal number of colors needed to color the graph vertices such that any two adjacent (neighbouring) ones have different colors. The GCP is an NP-Hard problem [15] where optimal solutions can be found for its simple or medium sized instances [5, 29].

There are generally three approaches to solve the GCP [25, 28]. The first one consists in directly minimizing the number of colors by working on the legal colors space of the problem. In the second approach, the number of colors is fixed and no conflict is allowed, thus, some vertices might not be colored. The objective here is to maximize the number of colored vertices [3, 36]. The third approach consists of first choosing a number of colors K, and then iteratively try to minimize the number of conflicts for the candidate K. Whenever a solution with zero conflicts has been found, K is decremented by one and the procedure continues until we reach a K where the number of conflicts cannot be equal to zero. As a result, the last legal K will be returned as the best solution [25].

Since our aim is to develop a parallel approach for solving the GCP, we focus on the third approach as it is suitable for this purpose. This approach suffers from two major problems. First, we have to solve the GCP assuming that the graph is K colorable, and if solved via K colors, we reduce K by one and solve the problem again and continue this process until we find the minimum possible K. This phenomenon causes a waste of time and resources since it

takes the opportunity to consider other solutions at the same time. The second problem is that the number of vertices is used as the initial value of K. This will affect the efficiency of the solving algorithm especially for large problems instances where there is a big difference between the initial value of K and the optimal one.

We propose an extension of this approach addressing and improving its two limitations as follows. We solve the GCP in a hierarchical and parallel way using a set of collaborating Parallel Genetic Algorithms (PGAs). We call this new approach Hierarchical PGA (HPGA). Each PGA in the HPGA is assigned to work on solving the GCP using a unique number of colors. Moreover, given the fact that in genetic algorithms, the random crossover operator performs poorly for combinatorial optimization problems [10, 12], we extend the PGA with an additional operator that we propose, namely the Genetic Modification (GM). Using the properties of variables and their relations, the GM operator generates good individuals at each iteration and inserts them into the PGA population in the hope of reaching the optimal solution sooner. In the case of the GCP, the GM operator is based on a novel Variable Ordering Algorithm (VOA) that we propose for the GCP, namely Dependency Variable Ordering for Graph Coloring Problem (DVOGCP). To overcome the second drawback, we designed a novel greedy algorithm to estimate the upper-bound for the graph's chromatic number. We then use this estimator to evaluate the initial value of K.

In order to evaluate the performance of our new approach, we have conducted several experiments on GCP instances taken from the well known DIMACS graphs website. The results of these experiments show that our proposed approach solves the GCP efficiently and in a timely manner with a great accuracy in finding the optimal solution.

The rest of the paper discusses our contributions in detail. In Sects. 2 and 3, the design of the proposed HPGA and the estimator are respectively covered. Section 4 introduces the different components of the PGA designed specifically to solve the GCP. Section 5 is dedicated to the experimentation we conducted in order to evaluate the performance of our proposed approach. Finally, concluding remarks and possible future works are presented in Sect. 6.

2 The proposed HPGA

2.1 Background: PGAs

Genetic Algorithms (GAs) [16] are evolutionary algorithms based on the idea of natural selection and evolution. GAs have been successfully applied to a wide variety of problems [1, 11, 17, 35, 38]. In GAs, there is a population of potential solutions called individuals. The GA performs different genetic operations on the population, until the given stopping criteria are met.

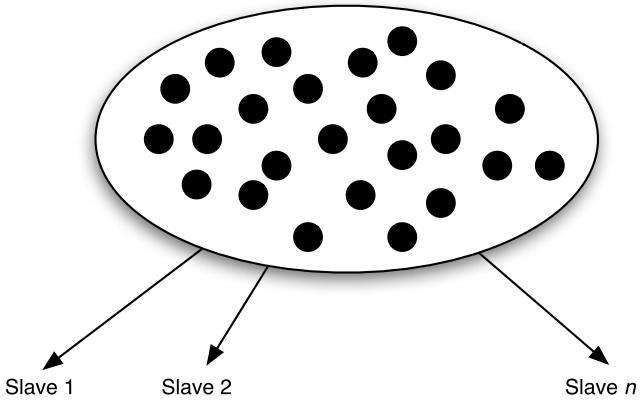


Fig. 1 Master–Slave PGA model

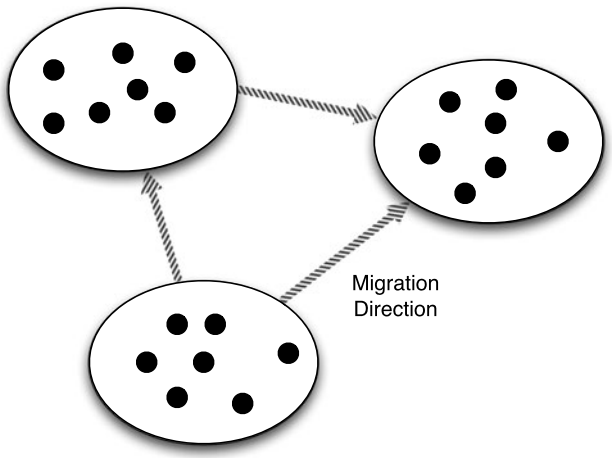


Fig. 2 Island PGA model

The Parallel Genetic Algorithm (PGA) is an extension of the GA. The well-known advantage of PGAs is their ability to facilitate different subpopulations to evolve in diverse directions simultaneously [23]. It is shown that PGAs speed up the search process and can produce high quality solutions on complex problems [9, 24, 34].

There are mainly three different types of PGA [4]. First, Master-Slave PGA in which, there is only one single population and the population is divided into fractions. Each fraction is assigned to one slave process on which genetic operations are performed (see Fig. 1) [23]. Second, Multi-Population PGA (also called Island PGA) that contains a number of subpopulations, which can occasionally exchange individuals (see Fig. 2). The exchange of individuals is called migration. Migration is controlled using several parameters. Multi-population PGAs are also known as Island PGA, since they resemble the "island model" in population genetics that considers relatively isolated demes. Finally, the Fine-Grained PGA which consists of only one single population is designed to run on closely linked massively parallel processing systems [23].

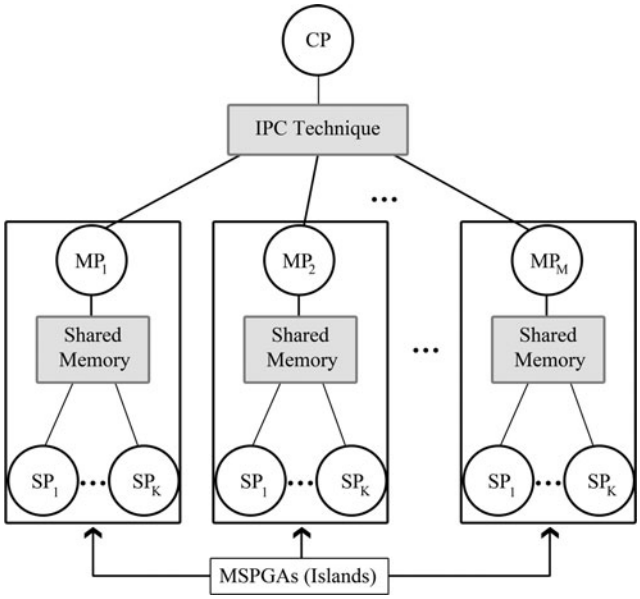


Fig. 3 HPGA architecture

2.2 HPGA architecture

As mentioned in the introduction, our proposed approach for solving a GCP executes PGAs in parallel using a Hierarchical PGA (HPGA) architecture. A HPGA can be obtained by any combination of the PGA types discussed earlier. Figure 3 shows the architecture of our proposed HPGA. To design the HPGA, we use the Island PGA (IPGA) for the top level and Master-Slave PGA (MSPGA) for the lower level. Each MSPGA, is actually an island of the IPGA; however, there is a Coordinator Process (CP) in the IPGA which is in charge of assigning different GCP problem domains to each island of the IPGA. The CP can communicate with each MSPGA using the chosen Inter-Process Communication (IPC) mechanism. The rest of this section covers the design of the HPGA in depth.

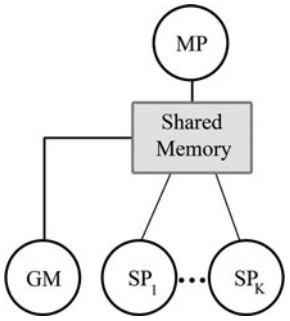
2.3 Designing the MSPGA for the GCP

Each MSPGA has only one goal: to solve the GCP problem via a color domain of size N. A MSPGA consists of one Master Process (MP) and its Slave Processes (SPs) (as shown in Fig. 3). Each SP performs genetic operations on a subpopulation assigned to it by the MP. At each step of the GA, the MP nominates P best individuals gathered from each SP's population and distributes them to SPs for the reproduction. For the sake of efficiency, we used Shared Memory as IPC since the MP and its SPs need to interact a lot in each generation of the GA.

2.4 Extending the PGA using Genetic Modification (GM)

We define the term Genetic Modification (GM) for generating good individuals based on the relations between vertices and constraints in the GCP. This means that the GM operator would purposefully insert some engineered individuals into the GA's population to give them a chance to participate in reproduction. The process of generating individuals based on this idea might be time consuming compared to just randomly generate individuals or perform a random crossover. Thus, to resolve this issue, the GM operator should not interfere with the flow of the PGA and this latter should not wait for the GM operator results to enter the reproduction. The idea here is that the GM should concurrently and independently operate beside the PGA. Whenever the GM produces a population of engineered individuals, the PGA keeps them until the next reproduction. Then, just before the reproduction, the PGA distributes them between the subpopulations. Figure 4 shows the architecture of a MSPGA including the GM operator. In our work, the size of the population produced by the GM is equal to 20 % of the subpopulation size.

Fig. 4 Architecture of a MSPGA with GM operator



Algorithm 1 GM algorithm

```
Begin
  modified_list ← ∅ (population of modified individuals)
  Wait for a command from MP
  while command ≠ STOP do
    for i := 1 to P_M do
      Generate a modified individual I
      Add I to modified_list
    Signal the MP
    Wait for a command from MP
End
```

The GM operator generates a modified population of size P_M. Algorithm 1 presents a general pseudo-code for the GM operator process. The GM operator should be designed according to the nature of the constraint optimization problem of interest. A specific GM operator for the GCP is introduced in Sect. 4.

2.5 Managing MSPGAs using the CP

As shown in Fig. 3, the interaction between a CP and its islands (MSPGAs) can use different mechanisms of the IPC. For instance, we could choose either Shared Memory or

Algorithm 2 IPGA algorithm

Require: All MSPGAs are suspended at the beginning.

1: Assign to each MSPGA, a distinct potential chromatic number from $[N - M_{suspended}, N) \cap \mathbb{N}$.
2: Start suspended MSPGAs. Wait for a MSPGA to find a solution. If a solution is found go to step 3. Meanwhile, if the Stopping Criteria are satisfied, stop the algorithm and return the best result found so far.
3: Update N to the best chromatic number found. Suspend the MSPGAs that have an equal or greater chromatic number than N.
4: Assign to each suspended MSPGA, a distinct potential chromatic number from $[N - M_{suspended}, N) \cap \mathbb{N}$. Go to step 2.

Message Passing. If we choose Message Passing, the CP can be considered as a machine with not necessarily high capabilities in a network with highly capable machines. On the other hand, we can use a multi-core super computer and choose the Shared Memory as IPC mechanism.

The CP is in charge of coordinating M MSPGAs. The value of M can be evaluated by considering available hardware resources. For example, in a Message Passing strategy, M would be the maximum number of highly capable machines available in the network for use. At the beginning of the algorithm, the CP assigns a distinct coloring domain from $[N - M, N) \cap \mathbb{N}$ to each MSPGA (N is the estimated upper-bound for $\chi(G)$). The MSPGAs then compete with each other to find a solution to the GCP with their given number of colors. Whenever the CP receives a valid solution from a MSPGA, it suspends the operation of MSPGAs that are working on color domains greater than the received solution. The CP then updates N to the current known chromatic number and assigns a distinct coloring domain from $[N - M_{suspended}, N - 1] \cap \mathbb{N}$ to each suspended MSPGA and resumes them. This process continues until the algorithm finds the minimum possible chromatic number or a given time is passed.

2.6 Proposed HPGA procedures

Consider M as the total number of PGAs, $M_{suspended}$ as the number of PGAs in the *suspend* state, and N as the estimated chromatic number received from our proposed estimator that we will present in the next section. Algorithms 2 and 3 respectively list IPGA and MSPGA procedures for solving a GCP instance.

Algorithm 3 MSPGA algorithm

1: **In Parallel**: generate a random population of size P. Calculate the fitness of each individual.
2: If a solution is found (new chromatic number), signal the CP and wait for a task from the CP. Else, go to the next step.
3: Before entering the reproduction, check if the GM process has created a modified population. If so, distribute them amongst subpopulations.
4: **In Parallel**: perform reproduction, mutation, and fitness calculation. Go to step 2.

3 A new estimator for the GCP

3.1 Background: sequential graph coloring algorithms

The GCP is a very well known NP-hard problem that has been extensively reported in the literature for the past decades. Exact methods for solving the GCP include branch and bound and other algorithms based on exhaustive search. In [8], Coudert proposes a branch and bound algorithm that uses the size of a clique in a graph to get a better lower bound of the chromatic number. Like any other systematic search technique, this method suffers from its exponential time cost especially for large graphs with arbitrary connectivity. In order to overcome this difficulty in practice, several polynomial time heuristics have been proposed in the literature. These heuristics can be seen as truncated branch and bound algorithms and include contraction algorithms as well as sequential coloring techniques. A survey of these heuristics can be found in [19, 26]. Let G be a graph with a set of vertices $V = \{x_1, x_2, \ldots, x_n\}$. Every sequential graph coloring algorithm is based on the order in which vertices are selected and has, in general, the following model.

1. The vertices $x_1, x_2, \ldots, x_n$ are ordered based on a *priority* rule.
2. Each vertex in the ordered list is colored with the minimum possible color number.

In practice, the order of processing the vertices is very important as if can greatly impact the running time of the solving method. Vertices ordering can be static or dynamic. An example of a simple static ordering *priority* rule is the one proposed in [37] for the case of scheduling and timetabling problems represented as GCPs. In this rule we simply create a decreasing ordered list of vertices based on their degrees. We then attempt to color the graph. Dynamic ordering includes DSATUR [2] and Recursive Largest First (RLF) [20] heuristics where vertices are ordered based on their degree in addition to the colors used by their adjacent nodes (neighbors). In [7] an evolutionary technique based on the RLF heuristic is proposed. Note also the new algorithm proposed in [18] based on several techniques and

Algorithm 4 DSATUR algorithm

1: Arrange the vertices by decreasing order of degrees.
2: Color a vertex of maximal degree with color 0.
3: Choose a vertex with a maximal saturation degree. If there is an equality, choose any vertex of maximal degree in the uncolored subgraph.
4: Color the chosen vertex with the least possible (lowest numbered) color.
5: If all the vertices are colored, stop. Otherwise, return to step 3.

heuristics including the least-constraining and most constrained heuristics, divide and conquer and global probabilistic search.

Based on the Saturation degree of vertices, DSATUR algorithm is defined as follows. Let G be a simple graph and C a partial coloring of G vertices. We define the saturation degree of a vertex as the number of different colors to which it is adjacent (colored vertices). Algorithm 4 illustrates the description of DSATUR [2].

Our proposed estimator is a sequential graph coloring algorithm where the priority rule is defined as follows. First, like the WP rule, a vertex x_i with the maximal degree is selected and colored. Then, the next vertices to be selected are the neighbors of x_i which are selected based on their degrees. In some sense, our proposed algorithm selects a subgraph containing the most constrained vertex together with its neighbors and colors them. It then goes to the next subgraph and continues until all the vertices are colored.

In the next subsection we present the details of our proposed estimator.

3.2 The proposed estimator

The Estimator algorithm receives an uncolored graph G at input and returns the minimum number of colors needed to color G. For that, it uses a working graph A (initially empty) that corresponds, at each step of the algorithm, to the colored sub graph of G. The details of the algorithm are shown in Algorithm 5.

The idea behind our estimator is to first identify, in G, the sub-graph that corresponds to the most constrained vertex and its adjacent vertices. This sub-graph is then solved according to the constraints that we have so far in the partially constructed graph A. Once a sub-graph is solved, the algorithm moves to the next unsolved sub-graph with similar properties. This process continues until the whole problem is solved and there is no other uncolored vertex left. The algorithm uses a greedy method for choosing a color for a vertex, since it always seeks the minimum available color.

Figure 5 shows the steps of the algorithm for a sample graph. For the sake of simplicity, we suppose that colors are

Algorithm 5 GCP estimator algorithm

1: Start with the uncolored graph G where all vertices are not marked, and the empty graph A.
2: Create a list L of all the vertices of G sorted in decreasing order of their degree.
3: **while** there exists a vertex in G that is not marked **do**
4: Pick the first non marked vertex g_i from L, add it to A and name it a_i.
5: Add the edges between a_i and the rest of vertices in A, according to their correspondence in G.
6: Solve the new sub-GCP by choosing the minimum possible color number for a_i.
7: mark g_i as colored.
8: **while** there exists an adjacent vertex adj_{g_i} to g_i in G that is not marked **do**
9: Pick the first non marked vertex adj_{g_i} from L, add it to A and name it adj_{a_i}.
10: Add the edges between adj_{a_i} and the rest of vertices in A, according to their correspondence in G.
11: Solve the new sub-GCP by choosing the minimum possible color number for adj_{a_i}.
12: mark adj_{g_i} as colored.
13: **end while**
14: **end while**
15: Return the total number of colors used.

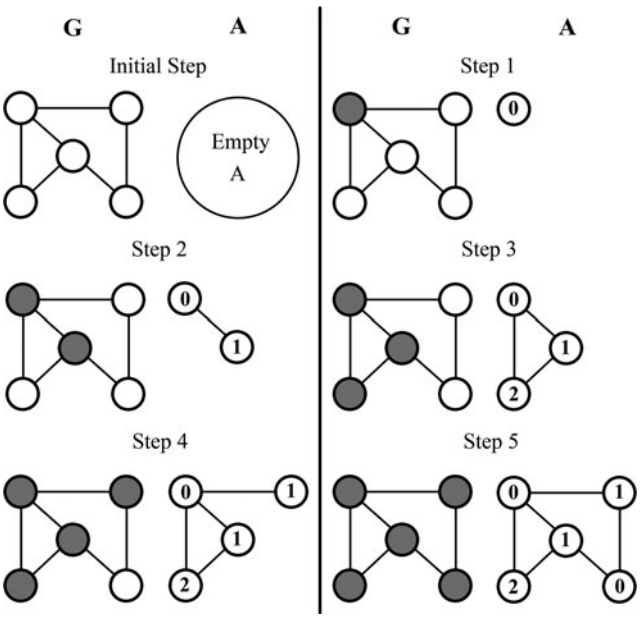


Fig. 5 Steps of estimator algorithm for a sample graph

enumerated starting with zero. In each step of the algorithm, we add one vertex to the partial graph A. As a result, we just need to check the adjacency matrix for the newly added vertex and choose a color with the minimum possible number for the added vertex. The algorithm discussed above is

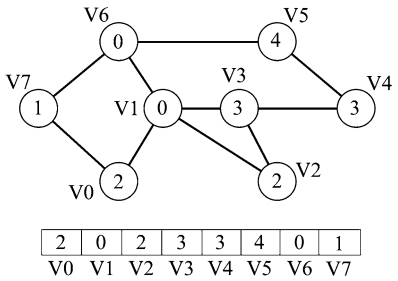


Fig. 6 Individual representation of an eight-vertex graph

rather conceptual as we can implement it without actually using the partial graph A. We only need to keep track of the colored vertices (every colored vertex is in A). The algorithm can be implemented to run in $O(|V|^2)$ where $|V|$ is the number of vertices.

4 PGA components

4.1 Representation of individuals

Each individual in the population is represented with an integer array, which has a length equal to the number of graph vertices. The value of each array entry is a color number within the color domain. Figure 6 illustrates an example of an individual for an eight-vertex graph with a color domain of size 5, and its correspondence in the graph.

4.2 Fitness function

The fitness function of an individual is the number of conflicts between adjacent vertices. This corresponds to the number of adjacent vertices with the same color. In order to compute this value, we simply find adjacent vertices from the graph adjacency matrix and check their color number in the integer array of the individual. When the fitness function is equal to zero, a solution is found. The fitness of an individual, f_I, is defined as follows:

$$f_I = \sum_{i \in V_G} \sum_{j \in adj_i} conflict(i, j)$$

where V_G is the set of all vertices of the graph and adj_i is the set of all vertices adjacent to vertex i.

The conflict function is defined as follows:

$$conflict(i, j) = \begin{cases} 1 & \text{if } i \text{ and } j \text{ have the same color} \\ 0 & \text{otherwise} \end{cases}$$

4.3 Parental success crossover

Reproduction takes place amongst a number of fittest individuals in each subpopulation. The chosen individuals are

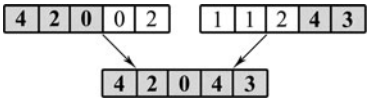


Fig. 7 A one point crossover of a five-vertex graph

then passed to crossover as parents of new individuals. We first adopted the k-point crossover where the value of k is generated randomly at the time of the crossover. Figure 7 shows an example of a 1-point crossover on two individuals of a five-vertex graph coloring problem.

However, as discussed in [10] and verified by our preliminary experimentations, a completely random crossover performs poorly for constraint optimization problems. Indeed, in these problems the k-point crossover may eliminate some useful individuals, since when applied to two good individuals it does not necessarily generate a good or a better one. The main reason for such a phenomenon is that in constraint optimization problems in general, and GCPs in particular, changing the value of a variable can have direct effects on other variables that are in constraint relation with the changing variable and indirect effect on other variables. As a result, performing a random crossover often reduces the quality of the solution. This has been a motivation to propose the following new crossing method. Based on this method, each individual in the population maintains two records; the total number of times it has participated in reproduction (N_p), and the number of times the offspring it produced was fitter. We refer to the latter as Parental Success Number and denote it by P_s. The parental success ratio, denoted by S, can then be calculated as follows:

$$S = \frac{P_s}{N_p}$$

Note that the parental success ratio of each individual is initially equal to zero. Furthermore, we define the term Fitness Around Variable (FAV) as the number of conflicts between a given variable (corresponding to a vertex) and its neighbors in the constraint graph. Each individual keeps a record of the fitness around all of its variables.

Using these new parameters, we create a Crossover Mask. The offspring is then produced according to the crossover mask. Let p_1 and p_2 be two parents chosen for the crossover. The crossover mask specifies which allele in the new chromosome should inherit from which parent. Since the crossover is performed here using two parents, the crossover mask consists of binary digits. Given two parent p_1 and p_2, 1 corresponds to choosing the allele from p_1 while 0 corresponds to choosing it from p_2. To create the mask, we compare each allele in p_1 with its correspondence in p_2. If the FAV of the allele in p_1 is less than the one in p_2, we put a 1 in the mask. If the FAV of the allele in p_2 is less than the one in p_1, we put a 0 in the mask. In case of

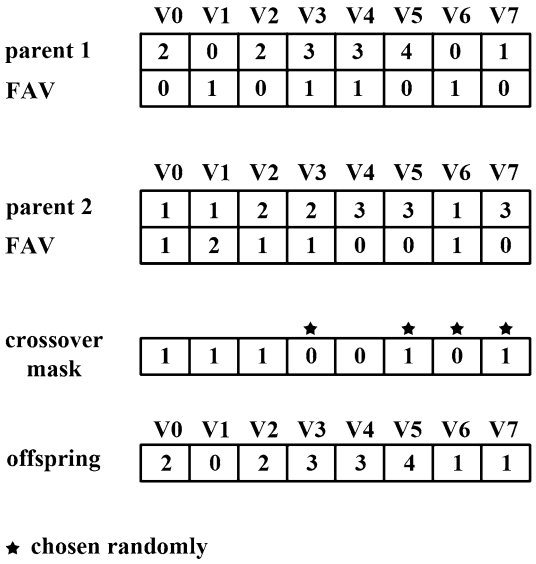


Fig. 8 Example of the parental success crossover on an eight vertex graph

equality, we use the following probabilities for choosing the allele:

$$P(\text{choosing from } p_1) = 1/2 + (S_{p_1} - S_{p_2}) \times 1/2$$

$$P(\text{choosing from } p_2) = 1/2 + (S_{p_2} - S_{p_1}) \times 1/2$$

Where, S_{p_1} and S_{p_2} are the parental success ratios of p_1 and p_2, respectively. An example is illustrated in Fig. 8. The Parental Success Crossover (PSC) in fact favours the parent that has a higher parental success ratio and a lower FAV for each allele. A parent with a higher success ratio tends to be a more well-instantiated CSP solution. As a result, choosing the crossing values for the new offspring from this parent is more promising. PSC tends to preserve the structure of a good parent while generating a new, different offspring based on that.

Figure 9 illustrates an example of using PSC on a GCP and compares it with the case of using a One Point Crossover (OPC). In this figure, patterned colors are used to color each vertex of the graph. As it can be clearly seen, when we apply the PSC to the two candidate parents, the produced offspring could be much better than the parents (in this case it is actually a solution). On the other hand, when a OPC is applied two the same parents, the produced offspring could be much worse than the parents. Algorithm 6 shows the function for generating PSC Mask.

4.4 Mutation

We propose two different methods for the mutation. The first method that is called *mutation to minimize the number of conflicts* is used to locally improve the solution while performing the mutation. On the other hand, the second

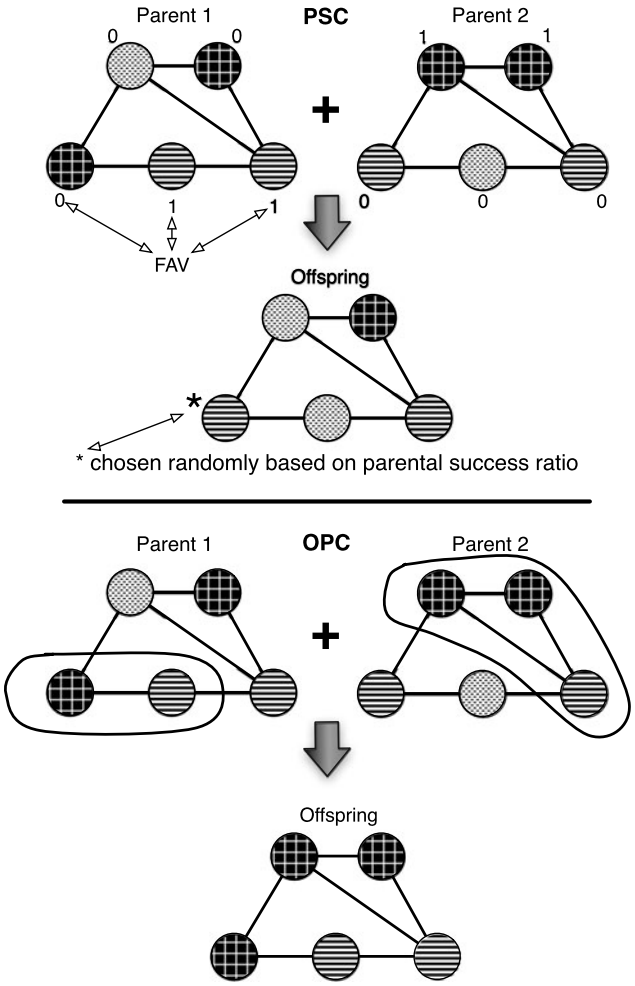


Fig. 9 Comparison of PSC and OPC in Producing an Offspring in GA for GCP

method, *stochastic color change*, performs a complete random mutation. Since *mutation to minimize the number of conflicts* has a greedy strategy to modify the individuals, using *stochastic color change* is necessary to preserve the diversity of the population.

4.4.1 Mutation to minimize the number of conflicts

In this type of mutation, $N_{mutation}$ random vertices of the individual are selected and the numbers of color conflicts around the chosen vertices and their adjacent vertices are minimized. Say vertex A is randomly chosen for the mutation, then, according to the adjacency matrix of the graph, for every vertex B that is adjacent to A, if their colors are the same, B will take a new random color that is not equal to A's color (see Algorithm 7).

Each time we perform a mutation, $N_{mutation}$ takes a random value between 1 and N_{Max}. N_{Max} is computed as follows using another parameter that we introduce called Allele Mutation Percentage. Suppose we have an individual of

Algorithm 6 Generating PSC mask

```
function PSC-MASK(FAV_p1, FAV_p2 as Arrays and S_p1, S_p2 as Doubles)
    Define: mask as Array
    Define: P_p1 as Double        ▷ probability of choosing from p1
    P_p1 = 0.5 + (S_p1 − S_p2) × 0.5
    for i = 0 to individualLength do
        if FAV_p1[i] < FAV_p2[i] then
            mask[i] = 1
        else if FAV_p1[i] > FAV_p2[i] then
            mask[i] = 0
        else
            if U(0, 1) ≤ P_p1 then ▷ U is uniform random function
                mask[i] = 1          ▷ choose from p1
            else
                mask[i] = 0          ▷ choose from p2
            end if
        end if
    end for
    return mask
end function
```

Algorithm 7 Mutation to minimize the number of conflicts

```
function MUTATE1(indiv as Array of Integers)
    Define: A as Integer    ▷ a vertex index of individual
    for i = 0 to N_mutation do
        A ← a unique, randomly chosen vertex index.
        for all B adjacent to A do
            if indiv[A] == indiv[B] then
                indiv[B] ← random color not equal to A's color
            end if
        end for
    end for
end function
```

size 100 and an allele mutation percentage of 20 %, then $N_{Max} = 20\ \% \times 100 = 20$.

4.4.2 Stochastic color change

This mutation method randomly chooses $N_{mutation}$ vertices and assigns a random color to each (see Algorithm 8).

4.5 Genetic Modification (GM) operator

We implemented the GM using a Variable Ordering Algorithm (VOA) that we propose for solving the GCP. At the beginning of the GM process, a variable ordering of the GCP is calculated using the proposed VOA. This variable ordering is considered as the best order for vertices to be colored in turn. Each variable in this ordering has an initial color domain of size N. Whenever the GM needs to create a new individual, it starts from the first variable in the ordering and generates a random value for each variable in turn. Following a look ahead principle, when a variable (vertex) is assigned a value, the GM propagates this change by removing that value (color) from the color domain of its neighbors. This way, it is guaranteed that at each time, the chosen value for a variable (vertex) will not cause a conflict. However, at the end of initializing variables, we might end up with some variables that have empty color domains. In this case, the GM randomly chooses a color for them.

Algorithm 8 Stochastic color change

```
function MUTATE2(indiv as Array of Integers)
    Define: A as Integer    ▷ a vertex index of individual
    for i = 0 to N_mutation do
        A ← a unique, randomly chosen vertex index.
        indiv[A] ← random color
    end for
end function
```

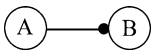


Fig. 10 The graph representation of the dependency relation $A \rightarrow B$

4.5.1 Dependency variable ordering for GCP

In Dependency Variable Ordering for GCP (DVOGCP), the dependency level of a vertex means the coloring of a vertex A depends on the color of the k adjacent vertices that are in a dependency relation with A. For instance, to color a vertex A with dependency level 2, we first have to color the two adjacent vertices involved in a dependency relation with A. A dependency relation between two vertices A and B is denoted by $A \rightarrow B$ and is interpreted as the color of vertex B depends on the color of vertex A. Figure 10 illustrates $A \rightarrow B$ in a graph.

We propose the following preceding rules in DVOGCP.

- A vertex with a lower dependency level always precedes the one with higher dependency.
- If two vertices have the same dependency, the one with the higher degree precedes the other.
- There is no ordering between two vertices with the same degree and dependency level.

Before creating the dependency relations, the graph vertices are sorted in a descending order of their degree. The first vertex in the sorted list, that is the vertex with maximum degree is the starting point in the algorithm. This vertex has a

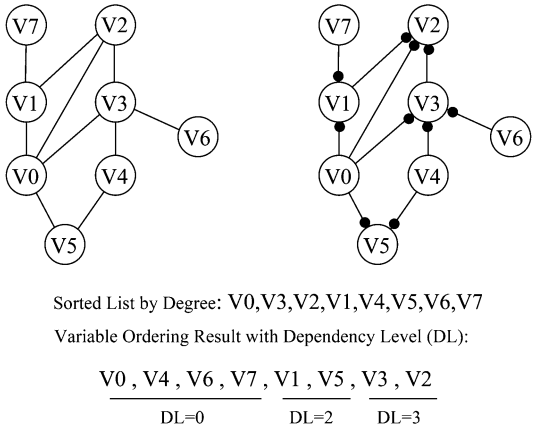


Fig. 11 Creating dependency relations of a GCP

dependency level of zero. Dependency relations are created according to the following rule:

– Considering two vertices A and B, $A \rightarrow B$ holds if and only if A has a higher or equal degree and a lower or equal dependency level in comparison to B. Otherwise, $B \rightarrow A$ holds.

Creating relations starts from the first vertex in the sorted list and continues for the rest of vertices in the list. At each iteration of the algorithm, we create dependency relations between the chosen vertex from the list and its adjacent vertices. Note that, the dependency relations are updated only when the algorithm is done creating them for the current vertex. The algorithm continues until we create all dependency relations for the vertices in the list. Then, the variable ordering is generated by creating a list of vertices with the following properties.

– First, the list is created by sorting vertices according to their dependency level in an ascending order.
– Second, each subsequence of vertices with the same dependency level in the list will be sorted according to their degree in a descending order.

Figure 11 shows a GCP instance and its DVOGCP.

4.6 Stopping criteria

The algorithm stops if a given timeout T is reached or a maximum number of generations is exceeded without finding a solution to the GCP.

5 Experimentation

Our proposed algorithm has been implemented using Java language (JDK 1.6) and has been applied to a variety of graph coloring instances. The GCP instances used in this section are from a benchmarking website formally named DIMACS graphs.[1]

Table 1 shows the results (in CPU time) of solving selected GCP instances with our proposed approach. The problem instances are taken from a range of small to large DIMACS problems where their chromatic number is reported. In this experiment, we used Shared Memory as the IPC mechanism between the CP and the MSPGAs. We defined 5 processes as the MPs for the 5 islands and a variable number of processes for SPs operating under each MP in MSPGAs (this number is reported in Table 1, "SPs per Island" column). The test machine is a Ciaratech FUSION SMP with 72 CPU cores. For each experiment we conduct 20 runs, each with a different random number generator seed, and take the average, min and max CPU time needed to return the solution.

In the experiments, the top 30 % of each subpopulation plus a number of randomly selected individuals are chosen for the crossover. In addition, in order to maintain a diverse population, every ten iterations we perform a crossover between randomly chosen parents. More precisely and following the idea of migration probability, every 10 iterations a totally random crossover is performed by choosing random parents amongst sub populations of each island. This is to control elitism in GA and maintain a diverse population on each island.

The mutation probability is set to 0.2. The probability to choose *mutation to minimize the number of conflicts* is 0.8 and the probability to choose the *Stochastic color change* mutation is 0.2. The allele mutation percentage is 20 % (see Sect. 4.4 for the definition of the above parameters).

Note that for all problems reported in Table 1, our estimator reaches the same upper bound (χ) as DSATUR of Brèlaz [2] but in a much better running time. This is justified by the fact that, in theory, the complexity of our estimator is $O(|V|^2)$ while it is $O(|V|^3)$ for DSATUR [19].

Next, we have compared our algorithm with the Parallel Genetic-Tabu Algorithm (PGTA) designed to solve GCPs [25]. In terms of the resulting chromatic number, both algorithms return the same results, except for the problem instance queen7_7.col, that the PGTA returns 7 while our HPGA returns 8. Figure 12 shows the comparative results of our proposed algorithm and PGTA with 24 processors in terms of runtime.

According to the figure, the results of our proposed algorithm are much better in all cases. The reason for such a significant improvement is that our Estimator finds a very good upper-bound for the chromatic number, causing the algorithm to start from a point near the optimal solution. This way, if the algorithm reaches the optimal solution, considering the fact that determining whether or not a solution

[1] http://mat.gsia.cmu.edu/COLOR03/.

Table 1 CPU time needed by our proposed algorithm for solving selected DIMACS GCP instances. V and E are respectively the number of nodes and edges of the graph. χ and *Res* are respectively the optimal solution and the solution returned by our method. *Avg Time*, *Max Time* and *Min Time* are respectively the Average, Maximum and Minimum running times in seconds needed by our method to return the solution. *SPs per Isld* and *PopSize per slv* are respectively the number of Slave Processes per Island and the population size per slave

Problem instances				Proposed algorithm results					
Instance	V	E	χ	*Res*	Avg Time (sec)	Max Time (sec)	Min Time (sec)	SPs per Isld	PopSize per slv
fpsol2.i.1.col	496	11654	65	65	46.1436	49.856	38.427	10	200
fpsol2.i.2.col	451	8691	30	30	31.3506	43.762	20.315	10	200
fpsol2.i.3.col	425	8688	30	30	32.9034	44.025	22.33	10	200
inithx.i.1.col	864	18707	54	54	99.639	131.633	82.748	20	200
inithx.i.2.col	645	13979	31	31	40.4542	54.401	31.1	10	200
inithx.i.3.col	621	13969	31	31	41.8646	58.643	32.206	10	200
mulsol.i.1.col	197	3925	49	49	8.7354	16.023	6.245	20	50
mulsol.i.2.col	188	3885	31	31	7.4734	12.948	5.703	20	50
mulsol.i.3.col	184	3916	31	31	7.1652	10.894	5.36	20	50
mulsol.i.4.col	185	3946	31	31	7.5034	13.184	4.339	20	50
mulsol.i.5.col	186	3973	31	31	8.0924	13.073	6.038	20	50
zeroin.i.1.col	211	4100	49	49	7.9662	13.407	6.095	20	50
zeroin.i.2.col	211	3541	30	30	9.0712	15.196	6.706	20	50
zeroin.i.3.col	206	3540	30	30	8.616	13.185	6.804	20	50
anna.col	138	493	11	11	2.9016	7.067	1.531	4	50
david.col	87	406	11	11	2.5522	5.538	1.288	4	50
homer.col	561	1629	13	13	9.7898	15.435	7.603	20	50
huck.col	74	301	11	11	4.6278	9.219	2.891	10	50
jean.col	80	254	10	10	4.398	8.644	2.69	10	50
games120.col	120	638	9	9	5.0038	9.299	3.1	10	50
miles1000.col	128	3216	43	43	6.425	12.049	4.546	10	100
miles1500.col	128	5198	73	73	10.1062	15.47	7.808	20	100
miles250.col	128	387	8	8	5.8828	11.476	3.244	10	100
miles500.col	128	1170	20	20	6.0624	11.356	4.253	10	100
miles750.col	128	2113	31	31	6.4144	9.927	4.928	10	100
queen5_5.col	25	160	5	5	2.3716	5.93	1.057	10	100
queen6_6.col	36	290	7	8	2.5062	5.067	1.221	10	100
queen7_7.col	49	476	7	8	2.8361	4.775	1.296	10	100
queen8_8.col	64	728	9	10	3.809	4.917	2.988	10	100
myciel3.col	11	20	4	4	0.753	0.836	0.648	2	50
myciel4.col	23	71	5	5	0.865	0.931	0.805	2	50
myciel5.col	47	236	6	6	1.7608	3.94	0.358	2	50
myciel6.col	95	755	7	7	1.479	3.239	0.47	2	50
myciel7.col	191	2360	8	8	5.3352	11.731	3.004	10	50
mug88_1.col	88	146	4	4	2.173	5.377	0.17	10	50
mug88_25.col	88	146	4	4	1.8002	3.92	0.182	10	50
mug100_1.col	100	166	4	4	2.016	4.951	0.182	10	50
mug100_25.col	100	166	4	4	2.5286	5.52	1.085	10	50
1-Insertions_4.col	67	232	4	5	2.4476	6.382	2.025	10	100
2-Insertions_3.col	37	72	4	4	1.8964	3.549	1.623	10	100
2-Insertions_4.col	149	541	4	5	6.4666	10.335	3.29	10	100
3-Insertions_3.col	56	110	4	4	3.347	5.247	0.983	10	100
qg.order30.col	900	26100	30	30	59.887	65.43	47.371	10	200

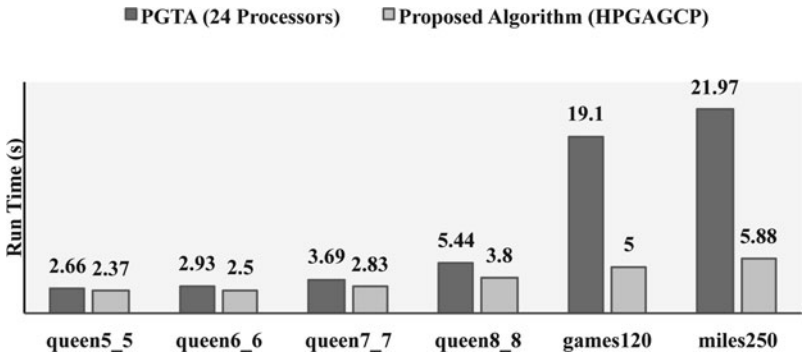


Fig. 12 CPU time comparison of the proposed algorithm and PGTA with 24 processors

is optimal is not possible for the algorithm, the maximum number of permitted generations without any solution is executed more quickly.

The Estimator also incredibly reduces the size of the search space. Then, the GM operator plays its role by inserting interesting individuals in the population. This will increase the chance of moving towards the optimal solution faster. In the end, since the GA is running in parallel, the runtime is significantly reduced. This phenomenon suggests that using our estimator and utilizing the idea of the GM operator by DVOGCP, together with the parental success crossover and a set of collaborating PGAs, we can significantly facilitate the conventional design of the genetic algorithms for solving a GCP.

To demonstrate the effectiveness of parallelism, we compared our proposed parallel algorithm to its sequential version. Figure 13 reports the CPU time in seconds needed by each version to reach the optimal solution with the speedup chart (number of times the parallel method is faster) shown in Fig. 14. As we can easily see the parallel version is much faster than the sequential one. Note that for some instances such as `myciel7.col` there is almost no difference between the running times of the parallel and sequential versions. This is due to the fact that for these particular instances the chromatic number given by our estimator to the solving method is actually equal to the optimal solution. The GA-based solving method does not do much in this case.

In order to assess the effect of the GM and the parental success crossover on the convergence of our proposed approach, we have experimentally compared three variants of our method on four problem instances. In the first one called *One Point Crossover + Mutation* only the one point crossover is used. In the second and third variants the parental success crossover is used without and with the GM respectively. These two methods are respectively called *Parental Success Crossover + Mutation* and *Parental Success Crossover + GM + Mutation*. Moreover we have implemented the most known and efficient hybrid evolutionary heuristic for solving the GCP [13, 14] and run it on the same four problem instances. This latter is a genetic local search method that combines local search with genetic algorithms.

Figures 15, 16, 17 and 18 show the number of iterations (generations of individuals) needed by each of the three variants and the hybrid evolutionary heuristic (denoted by HEA+LS) [13, 14] to return the solution with a corresponding fitness for each of the four selected problems. As we can easily see from the four figures, our 2 methods (*Parental Success Crossover + Mutation* and *Parental Success Crossover + GM + Mutation*) are definitely superior to the HEA+LS and *One Point Crossover + Mutation* techniques. In particular the *Parental Success Crossover + GM + Mutation* is the only method that finds the optimal solution for all the problem instances.

As we can easily see, the first variant fails to reach the optimal solution for all the four instances. On the other hand, the third variant is the only variant that converges towards the optimal solution in all four problems. Moreover, it is easy to see that for each problem the third variant converges sooner (with less iterations) than the second variant. This definitely demonstrates that our new crossover is always better than the one point crossover. In addition, the GM always helps when added to the new crossover. Finally the results shown in the figures demonstrate the superiority of our approach over the hybrid evolutionary heuristic on all the four problem instances.

6 Conclusion and future work

In this paper, we first discussed the limitations for solving the GCP using evolutionary algorithms. To address those issues, we proposed a number of algorithms including the HPGA to solve the GCP with different color domains simultaneously and to search in diverse directions of the search space. We also proposed a novel estimator to find an upper-bound for the graph's chromatic number.

Furthermore, we proposed an extension to the genetic algorithms, namely the Genetic Modification (GM) and the parental success crossover operators, specifically designed for solving discrete optimization problems.

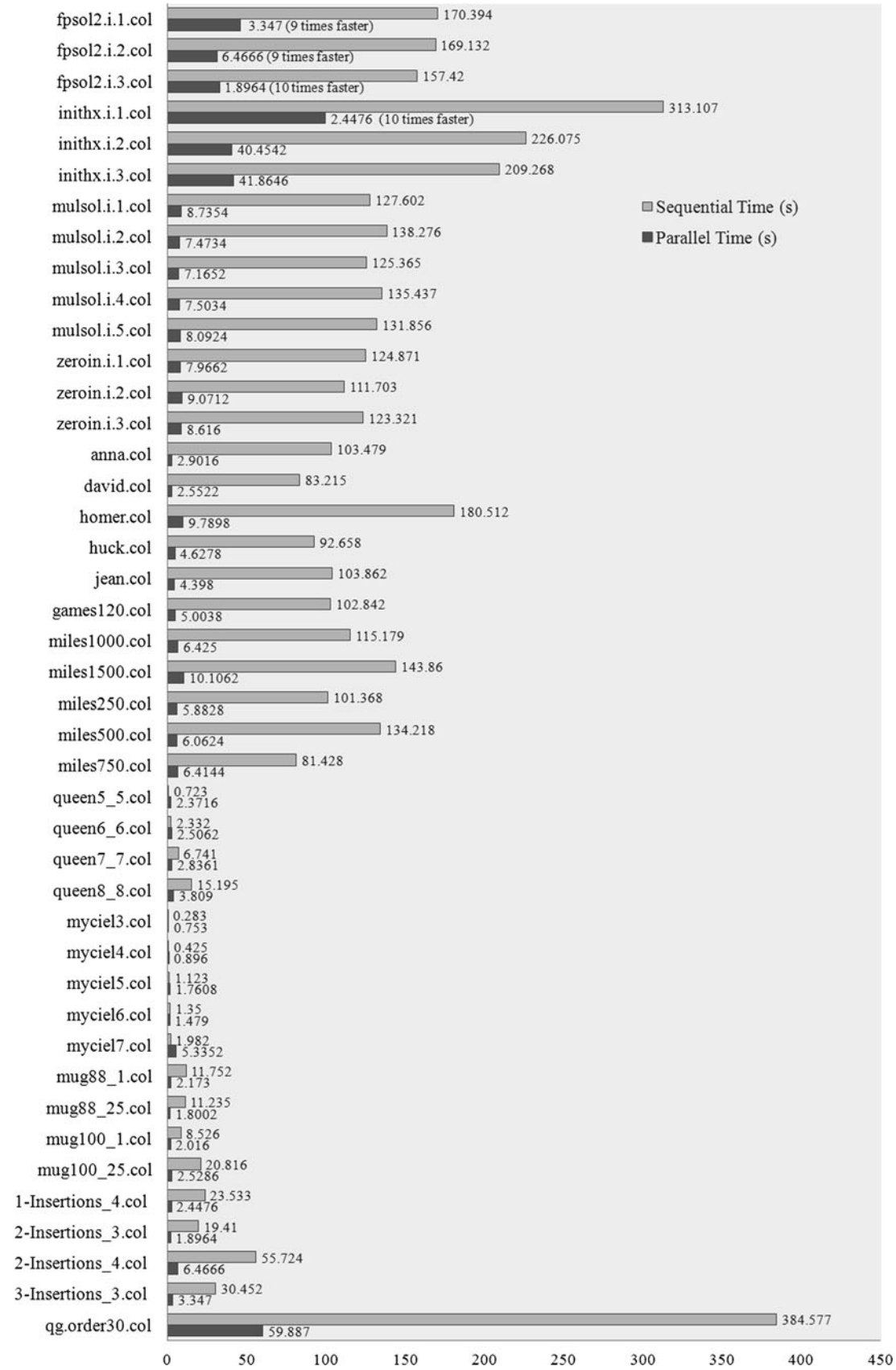


Fig. 13 CPU time comparison of the proposed parallel algorithm to its sequential version

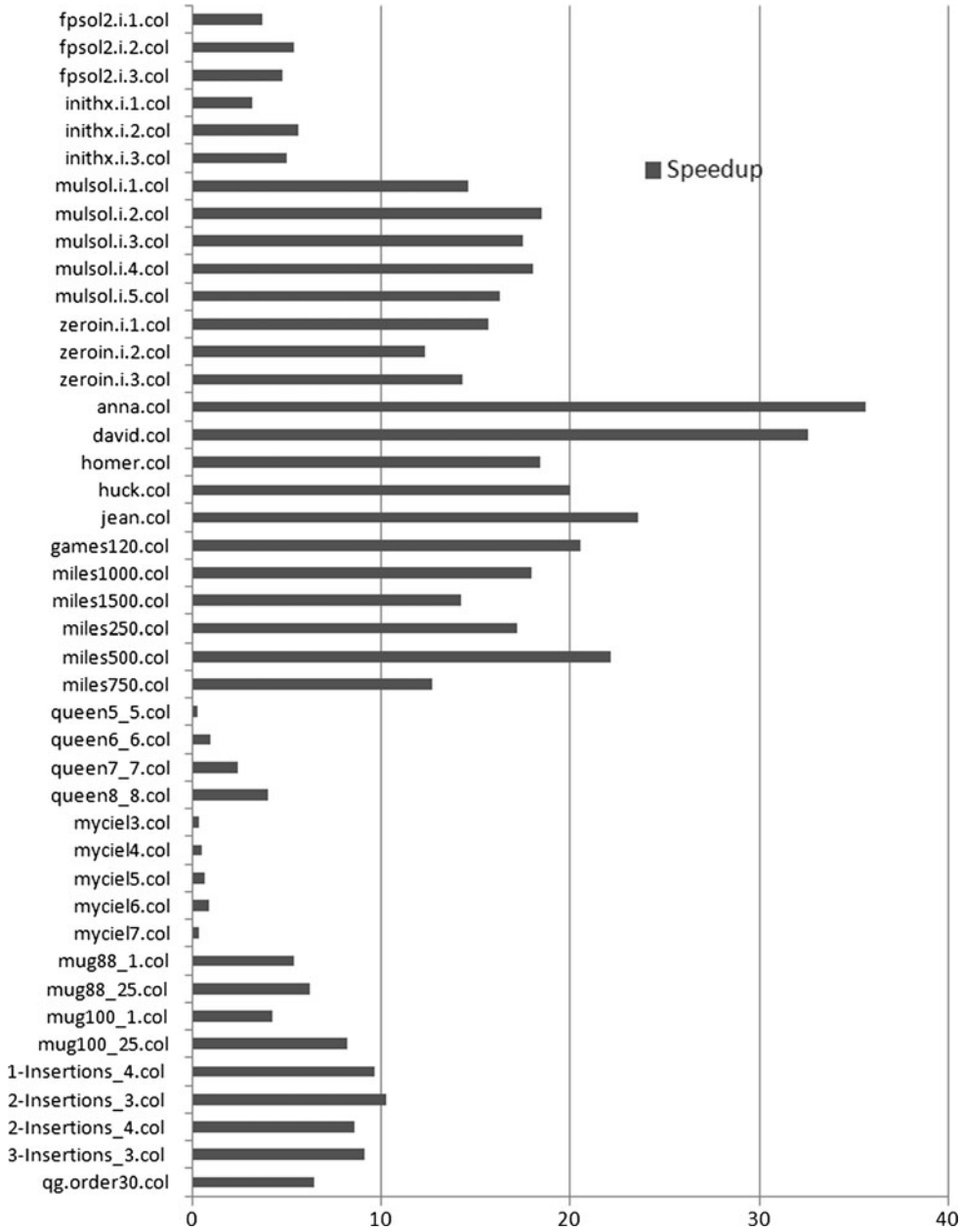


Fig. 14 Speedup chart (number of times the parallel method is faster than its sequential version)

In the experimentations that we conducted on various GCP instances, we showed that our proposed approach is very accurate and fast for solving the GCP. Apart from the efficiency provided by using a Hierarchical PGA, our proposed estimator together with the GM and crossover operators play an important role respectively in reducing the search space and generating near optimal solutions.

In the near future, efforts will be made to conduct more tests on other problem instances in order to further study the possible relations between our HPGA method and the structural features of the GCPs (such as the number of color classes) as well as to see how does our method scale up with the hardness of the problem. While the DIMACS suite is the common known library used for evaluating and comparing heuristic-based methods, it does not answer our needs for these new experiments. Instead, we will use the generator described in [12]. This latter has the ability to generate different types of k-colorable graphs as well as instances near the phase transition (transition between solvable and unsolvable instances) as hard instances are known to be concentrated near this region.

We also plan to generalize the whole proposed system to solve a variety of structured constraint optimization problems [22, 27, 33], including those in a dynamic environment [1, 30, 31]. This can be done by developing a generalized estimator for discrete optimization problems based on the idea

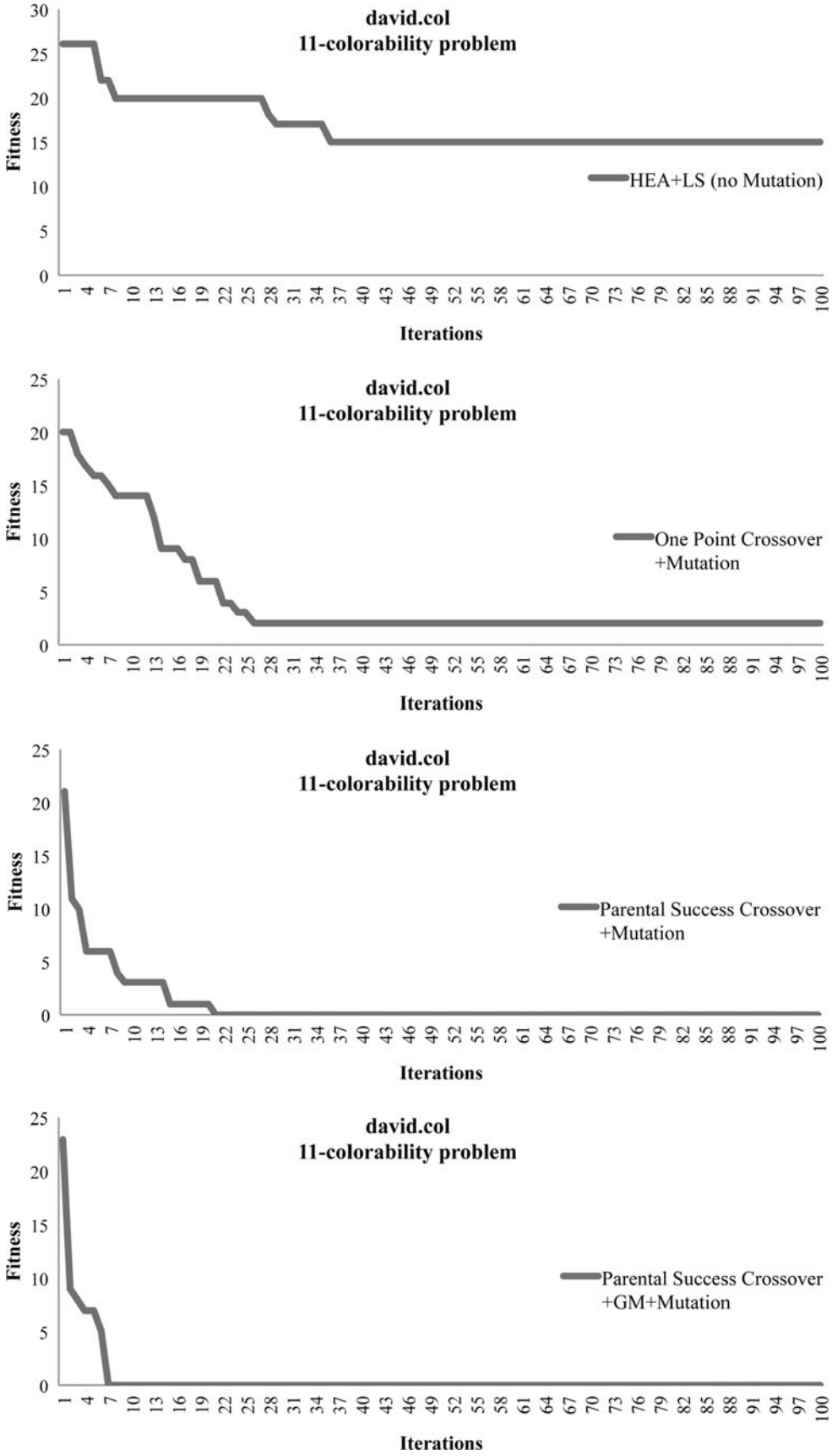


Fig. 15 Comparative test results for the `david.col` problem instance

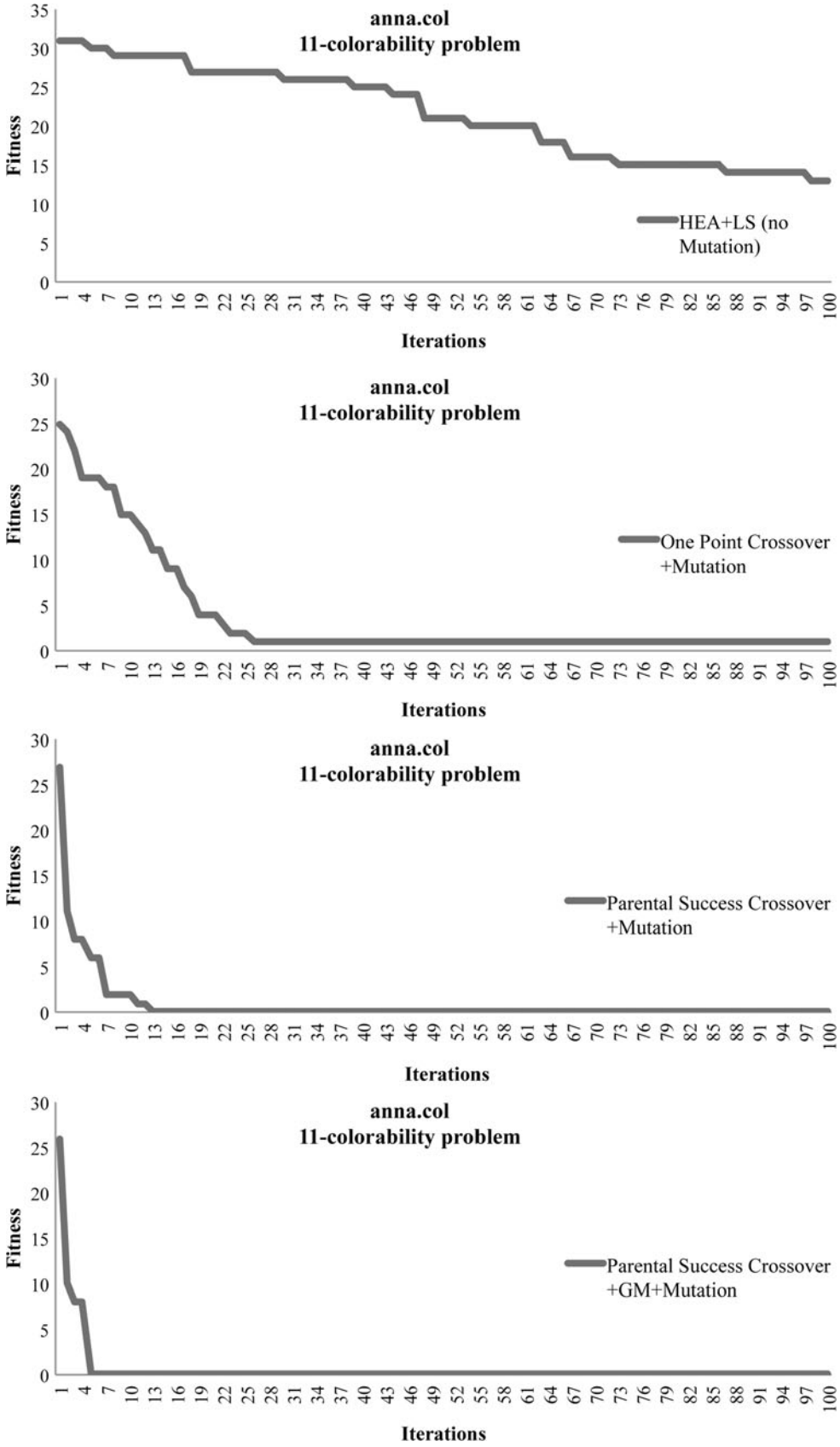


Fig. 16 Comparative test results for the `anna.col` problem instance

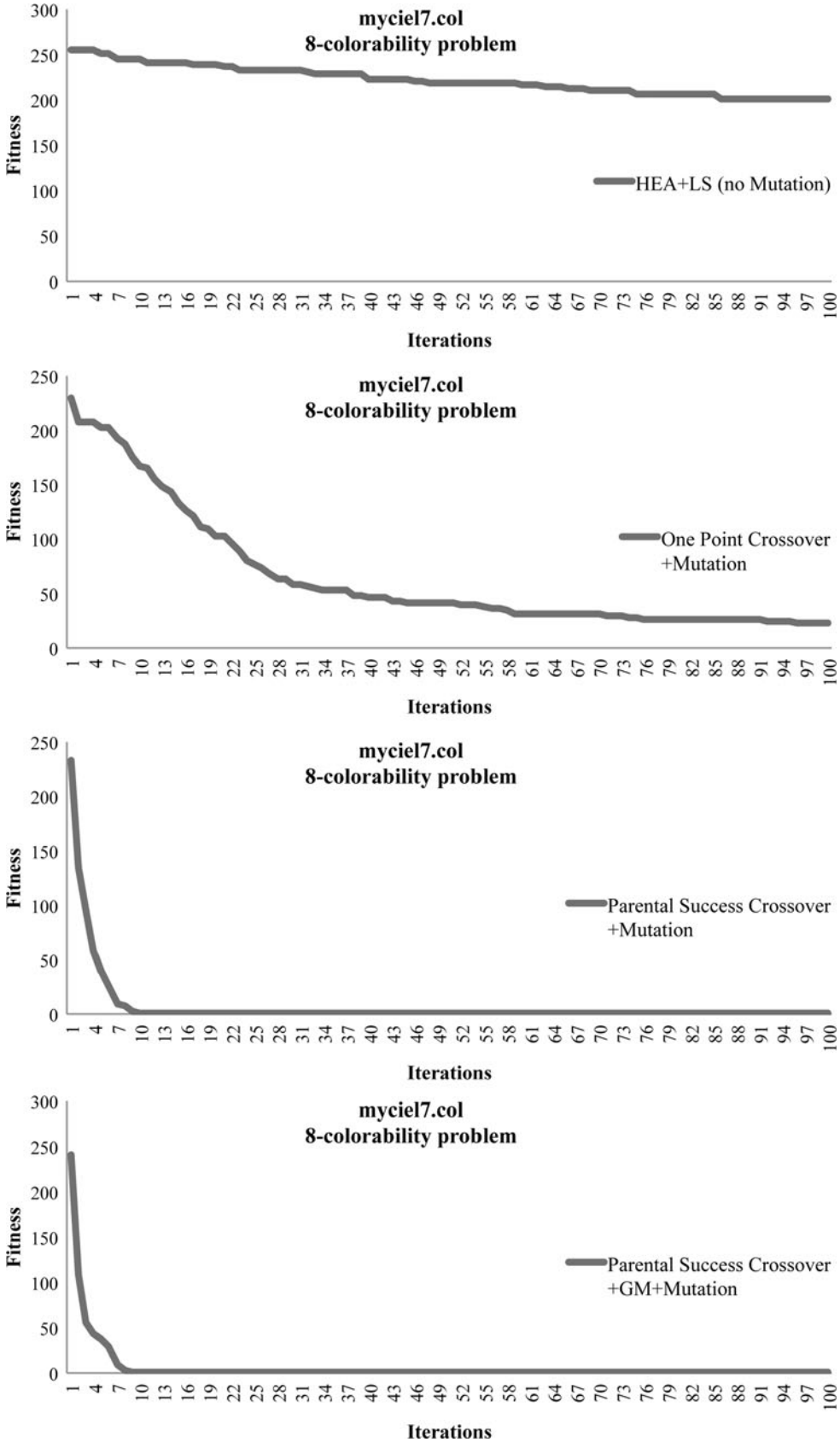


Fig. 17 Comparative test results for the `myciel7.col` problem instance

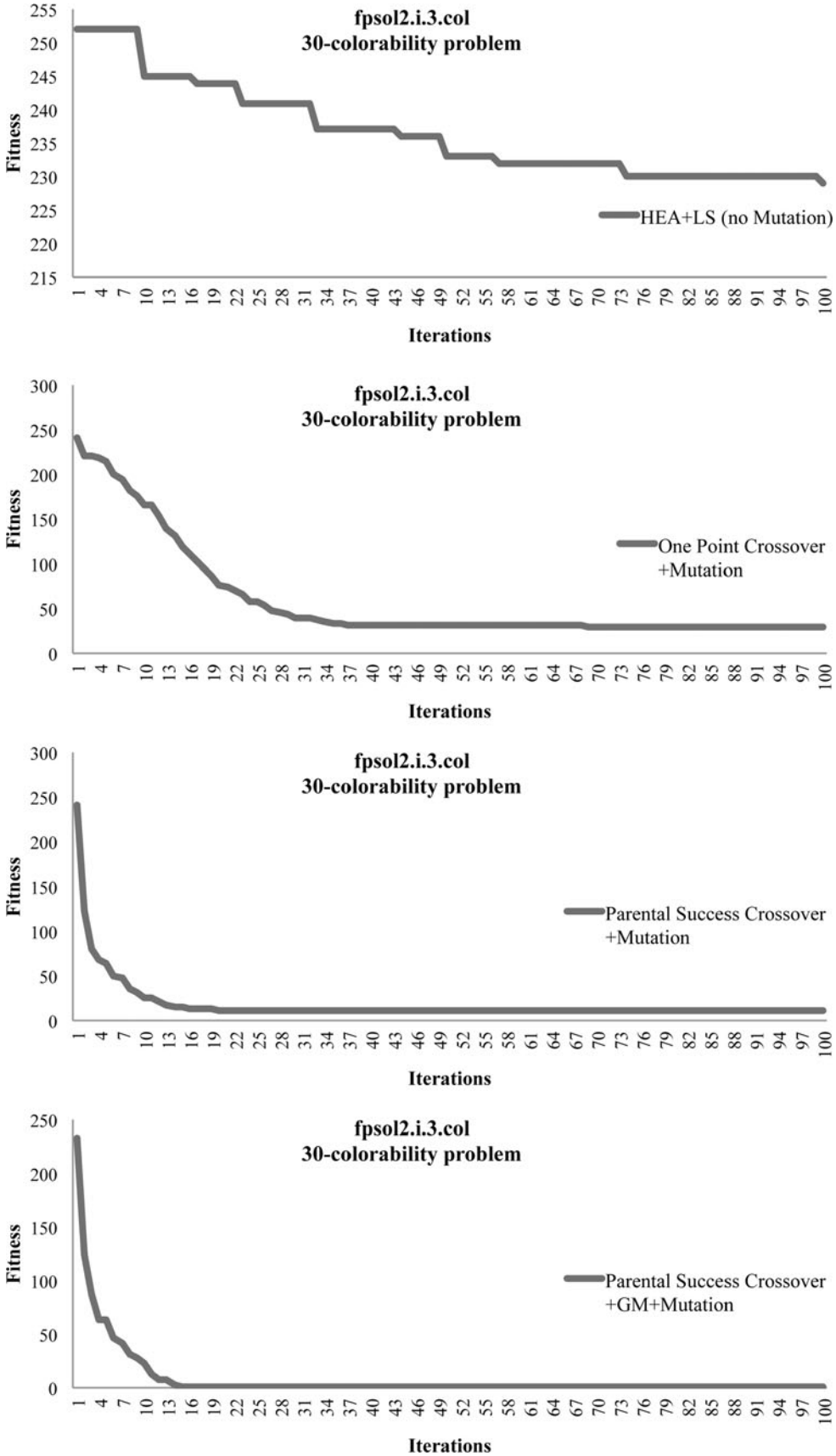


Fig. 18 Comparative test results for the `fpsol.col` problem instance

of our proposed Estimator. Moreover, different algorithms can be embedded into the GM operator and the Dependency Variable Ordering (DVO) can be generalized to operate on a wide range of problems.

References

1. Ayvaz D, Topcuoglu HR, Gürgen FS (2012) Performance evaluation of evolutionary heuristics in dynamic environments. Appl Intell 37(1):130–144
2. Brélaz D (1979) New methods to color the vertices of a graph. Commun ACM 22:251–256
3. Briggs P, Cooper KD, Torczon L (1994) Improvements to graph coloring register allocation. ACM Trans Program Lang Syst 16(3):428–455
4. Cantu-Paz E (2000) Efficient and accurate parallel genetic algorithms. Kluwer Academic, Norwell
5. Caramia M, Dell'Olmo P (2001) Iterative coloring extension of a maximum clique. Nav Res Logist 48(6):518–550
6. Chaitin G (2004) Register allocation and spilling via graph coloring. SIGPLAN Not 39(4):66–74
7. Costa D, Hertz A, Dubuis O (1995) Embedding of a sequential algorithm within an evolutionary algorithm for coloring problems in graphs. J Heuristics 1:105–128
8. Coudert O (1997) Exact coloring of real-life graphs is easy. In: 34th design automation conference, pp 121–126
9. Cui J, Fogarty TC, Gammack JG (1993) Searching databases using parallel genetic algorithms on a transputer computing surface. Future Gener Comput Syst 9(1):33–40
10. Cutello V, Nicosia G, Pavone M (2003) A hybrid immune algorithm with information gain for the graph coloring problem. In: Proceedings of the 2003 international conference on genetic and evolutionary computation: Part I (GECCO'03). Springer, Berlin, pp 171–182
11. da Silva FJM, Perez JMS, Pulido JAG, Rodriguez MAV (2010) AlineaGA—a genetic algorithm with local search optimization for multiple sequence alignment. Appl Intell 32:164–172
12. Fister I, Mernik M, Filipič B (2012) Graph 3-coloring with a hybrid self-adaptive evolutionary algorithm. Comput Optim Appl. doi:10.1007/s10589-012-9496-5
13. Galinier P, Hao JK (1999) Hybrid evolutionary algorithms for graph coloring. J Comb Optim 3(4):379–397
14. Galinier P, Hertz A, Zufferey N (2008) An adaptive memory algorithm for the k-coloring problem. Discrete Appl Math 156(2):267–279
15. Garey MR, Johnson DS (1990) Computers and intractability; A guide to the theory of NP-completeness. Freeman, New York
16. Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Reading
17. Kang MH, Choi HR, Kim HS, Park BJ (2012) Development of a maritime transportation planning support system for car carriers based on genetic algorithm. Appl Intell 36(3):585–604
18. Kirovski D, Potknojak M (1997) Exact coloring of many real-life graphs is difficult, but heuristic coloring is almost always effective. Technical report
19. Klotz W (2002) Graph coloring algorithms. In: Mathematics Report, pp 1–9. Technical University Clausthal
20. Leighton F (1997) A graph coloring algorithm for large scheduling algorithms. J Res Natl Bur Stand 84:489–506
21. Leighton FT (1979) A graph coloring algorithm for large scheduling problems. J Res Natl Bur Stand 84(6):489–506
22. Li J, Burke EK, Qu R (2010) A pattern recognition based intelligent search method and two assignment problem case studies. Appl Intell. doi:10.1007/s10489-010-0270-z
23. Lim D, Ong YS, Jin Y, Sendhoff B, Lee BS (2007) Efficient hierarchical parallel genetic algorithms using grid computing. Future Gener Comput Syst 23(4):658–670
24. Liu Z, Liu A, Wang C, Niu Z (2004) Evolving neural network using real coded genetic algorithm (ga) for multispectral image classification. Future Gener Comput Syst 20(7):1119–1129
25. Mabrouk BB, Hasni H, Mahjoub Z (2009) On a parallel genetic-tabu search based algorithm for solving the graph colouring proble. Eur J Oper Res 197(3):1192–1201
26. Malaguti E, Toth P (2010) A survey on vertex coloring problems. Int Trans Oper Res 17(1):1–34
27. Mansour N, Isahakian V, Ghalayini I (2011) Scatter search technique for exam timetabling. Appl Intell 34(2):299–310
28. Marx D (2004) Graph coloring with local and global constraints. PhD thesis, Budapest University of Technology and Economics
29. Mehrotra A, Trick MA (1995) A column generation approach for graph coloring. INFORMS J Comput 8:344–354
30. Miguel I, Shen Q (2000) Dynamic flexible constraint satisfaction. Appl Intell 13(3):231–245
31. Mouhoub M, Sukpan A (2012) Conditional and composite temporal CSPs. Appl Intell 36(1):90–107
32. Riihijarvi J, Petrova M, Mahonen P (2005) Frequency allocation for wlans using graph colouring techniques. In: Proceedings of the second annual conference on wireless on-demand network systems and services. IEEE Comput Soc, Los Alamitos, pp 216–222
33. Sabar NR, Ayob M, Qu R, Kendall G (2011)A graph coloring constructive hyper-heuristic for examination timetabling problems. Appl Intell. doi:10.1007/s10489-011-0309-9
34. Sena GA, Megherbi D, Isern G (2001) Implementation of a parallel genetic algorithm on a cluster of workstations: traveling salesman problem, a case study. Future Gener Comput Syst 17(4):477–488
35. Shi K, Li L (2012) High performance genetic algorithm based text clustering using parts of speech and outlier elimination. Appl Intell. doi:10.1007/s10489-012-0382-8
36. Svenson P, Nordahl MG (1999) Relaxation in graph coloring and satisfiability problems. Phys Rev E 59(4):3983–3999
37. Welsh D, Powell M (1967) An upper bound for the chromatic number of a graph and its application to timetabling problems. Comput J 10:85
38. Xing H, Qu R (2012) A compact genetic algorithm for the network coding based resource minimization problem. Appl Intell 36:809–823

Reza Abbasian obtained his M.Sc. degree in Computer Science from the University of Regina in Canada. His research interests are in the area of Constraint Solving and Evolutionary Computation.

Malek Mouhoub obtained his M.Sc. and Ph.D. in Computer Science from the University of Nancy in France. He is currently Professor of Computer Science at the University of Regina in Canada. His research interests are in Artificial Intelligence and include Temporal Reasoning, Constraint Solving and Programming, Scheduling and Planning. Dr. Mouhoub's research is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) federal grant in addition to several provincial and University funds and awards.