Computing and Informatics, Vol. 24, 2005, 123–147

EFFICIENT GRAPH COLORING WITH PARALLEL GENETIC ALGORITHMS

Zbigniew Kokosiński, Krzysztof Kwarciany, Marcin Kołodziej

Faculty of Electrical and Computer Engineering
Cracow University of Technology
ul. Warszawska 24
31-155 Kraków, Poland
e-mail: zk@pk.edu.pl

Manuscript received 12 October 2004; revised 20 March 2005
Communicated by Vladimír Kvasnička

Abstract. In this paper a new parallel genetic algorithm for coloring graph vertices is presented. In the algorithm we apply a migration model of parallelism and define two new recombination operators SPPX and CEX. For comparison two problem-oriented crossover operators UISX and GPX are selected. The performance of the algorithm is verified by computer experiments on a set of standard graph coloring instances.

Keywords: Graph coloring, parallel genetic algorithm, migration model, CEX crossover, SPPX crossover

1 INTRODUCTION

Graph k-colorability problem ("chromatic number problem") belongs to the class of NP-hard combinatorial problems [13, 19]. This decision problem is defined for an undirected graph $G = (V, E)$ and positive integer $k \leq |V|$: is there an assignment of available k colors to graph vertices, providing that adjacent vertices receive different colors? With additional assumptions many variants of the coloring problem can be defined such as equitable coloring, sum coloring, contrast coloring, harmonious coloring, circular coloring, consecutive coloring, list coloring etc. [17, 25]. In optimization version of the basic problem called GCP, a conflict-free coloring with minimum number of colors is searched. Intensive research conducted in this area resulted in a large number of exact and approximate algorithms, heuristics and metaheuristics [24, 33, 34]. However, the reported results are often difficult to compare due to specific assumptions, different algorithms and their implementation details, tuning of parameters, computing platforms, test data sets, etc. GCP was the subject of Second DIMACS Implementation Challenge held in 1993 [18] and Computational Symposium on Graph Coloring and Generalizations in 2002. A collection of graph coloring instances in DIMACS format and summary of results are available at [35, 36, 37].

Genetic algorithm (GA) is a metaheuristic often used for GCP [8, 9, 10, 11, 12, 20, 29, 32]. Recently a number of parallel versions of genetic algorithms (PGA) were studied. One popular model is master-slave in which a part of computations is assigned by master processor to slave processors [4]. Another approach is based on co-evolution of a number of populations that exchange genetic informations during the evolutionary process according to a communication pattern [1, 2, 7]. PGA were applied to many hard optimization problems (e.g. [27]) but, to our best knowledge, they were not used so far to GCP.

In this paper some results of experiments with parallel genetic algorithms for graph coloring problem are described. Some earlier results obtained by the authors were published in [22, 26].

After several initial experiments with alternative parallel models the migration model of parallelism was chosen. The main purpose of the master-slave model is speeding up processing of one global population by parallelization of computations. In the migration model of PGA we can expect that interaction between co-evolving populations can affect also the quality of the solution. If speedup is not essential one can simulate and test migration-based PGA with the help of a sequential program.

In the paper two new recombination operators for coloring chromosomes are proposed: SPPX (Sum-Product Partition Crossover) in which simple set operations and random mechanisms are implemented, and CEX (Conflict Elimination Crossover) that reduces the number of color conflicts with the help of selective copy operations. For reference we use two well known operators UISX (Uniform Independent Set Crossover) [9] and GPX (Greedy Partition Crossover) [12]. They both are problem-specific crossovers designed particularly for GCP and passed a series of experimental verification in GA environment [14, 21, 23].

In experimental part of the paper widely accepted DIMACS benchmark graphs were used. The obtained results are very promising and encourage future research focused on PGA and new genetic operators for a large class of graph coloring problems.

2 GRAPH COLORING PROBLEM – DEFINITION AND NOTATIONS

Let us define formally the optimization problem GCP.

For given graph $G(V, E)$, where: V – set of graph vertices, $|V| = n$, and E – set of graph edges, $|E| = m$, the optimization problem GCP is formulated as follows:

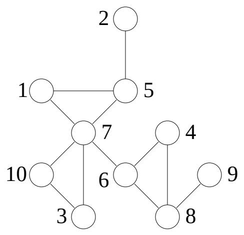


Fig. 1. Exemplary graph $G(V, E)$

find the minimum positive integer k, $k \leq n$, and a function $c : V \longrightarrow \{1, \ldots, k\}$, such that $c(u) \neq c(v)$ whenever $(u, v) \in E$. The obtained value of k is refered to as graph chromatic number $\chi(G)$.

Similarly, graph edge coloring problem for given graph $G(V, E)$ can be defined. One can find solution to mimimum edge coloring by solving vertex coloring problem for edge graph $G_e(V_e, E_e)$ associated with the given graph $G(V, E)$ [24, 17]. An exemplary graph $G(V, E)$ with ten vertices is shown in Figure 1.

In graph coloring problem k-colorings of graph vertices are encoded in chromosomes representing set partitions with exactly k blocks. There are two equivalent notations for vertex colorings that are commonly used in algorithm design.

In *assignment* representation available colors are assigned to an ordered sequence of graph vertices. Thus, the vector $c = \langle c[1], c[2], \ldots, c[n] \rangle$ represents a vertex coloring. For the graph in Figure 1, an optimal 3-coloring is denoted by vector $c = \langle 1, 2, 3, 2, 3, 1, 2, 3, 2, 1 \rangle$.

In *partition* representation a vertex coloring is a unique sequence of partition blocks in Hutchinson representation [15]. Each block of partition p does correspond to a single color. Elements inside each block are ordered in increasing lexicographic order, and all blocks are ordered increasingly according to the value of their first element. For our graph the same optimal 3-coloring is denoted by partition $p = \{1, 6, 10\}\{2, 4, 7, 9\}\{3, 5, 8\}$.

3 MODELS OF PARALLEL GENETIC ALGORITHMS

There are many models of parallelism in evolutionary algorithms: master-slave PGA, migration based PGA, diffusion based PGA, PGA with overlaping subpopulations, population learning algorithm, hybrid models etc. [3, 4, 5, 6, 16, 28, 30, 31].

The above models are characterized by the following criteria:

- number of populations: one, many;
- population types: disjoint, overlaping;
- population topologies: various graph models;
- interaction model: isolation, migration, diffusion;

- recombination, evaluation of individuals, selection: distributed/local, centralized/global;
- synchronization on iteration level: synchronous/asynchronous algorithm.

The most common models of PGA are:

- master-slave: one global population, global genetic operations, fitness functions computed by slave processors;
- massively parallel (cellular): static overlapping subpopulations with a local structure, local genetic operations and evaluation;
- migration (with island as a submodel): static disjoint subpopulations/islands, local genetic operations and migration;
- hybrid: combination of one model on the upper level and other model on the lower level (the speedup achieved in hybrid models is equal to product of level speedups).

4 MIGRATION MODEL OF PARALLEL GENETIC ALGORITHM

Migration models of PGAs consist of a finite number of disjoint subpopulations that evolve in parallel on their "islands" and only occasionally exchange genetic information under control of a migration operator. Co-evolving subpopulations are built of individuals of the same type and are ruled by one adaptation function. The selection process is decentralized.

```
procedure: genetic algorithm for a subpopulation
begin
  iteration counter $t = 0$;
  initialization of subpopulation $P_t$;
  evaluation of $P_t$;
  while (not termination condition) do
  begin
    parental population $T_t$ =selection from $P_t$;
    offspring population $O_t$ =crossover and mutation on $T_t$;
    evaluation of $\{P_t \cup O_t\}$;
    $P_{t+1}$ =selection from $\{P_t \cup O_t\}$;
    if (migration condition) then
      migration of representatives of $P_{t+1}$ to all other
      subpopulations
    $t = t + 1$;
  end;
end;
```

Fig. 2. Genetic algorithm for a subpopulation in the migration model

In our model the migration is performed on a regular basis. During the migration phase every island sends its representatives (emigrants) to all other islands and receives the representatives (immigrants) from all co-evolving subpopulations. This topology of migration reflects so called “pure” island model. The migration process is fully characterized by migration size, distance betweeen populations and migration scheme. Migration size determines the emigrant fraction of each population. This parameter is limited by capacity of islands to accept immigrants. The distance between migrations determines how often the migration phase of the algorithm occurs. Three migration schemes are applied: no migration, migration of randomly selected individuals and migration of best individuals of the subpopulation. Genetic algorithm performed in parallel for each subpopulation is shown in Fig.2. In our algorithm a specific model of migration is applied in which islands use two copies of genetic information: migrating individuals still remain members of their original subpopulation. In other words they receive new “membership” without losing the former one. Incoming individuals replace the chromosomes of host subpopulation at random. Then, a selection process is performed. The rationale behind such a model is as follows. Even if the best chromosomes of host subpopulation are eliminated they shall survive on other islands where their copies were sent. On the other hand, any eliticist scheme or preselection applied to the replacement phase leads to premature elimination of worse individuals and lowers the overall diversity of subpopulation.

5 GENETIC OPERATORS FOR GCP

In this section a collection of genetic crossover, mutation and selection operators is introduced that is used in our PGA. Three recombination operators: CEX, UISX, GPX and the mutation operator First Fit were designed especially for GCP. The operator SPPX and the mutation Transposition are more versatile. The cost function and selection operator is adapted for this version of GCP.

All examples in this section refer to the graph instance shown in Figure 1.

5.1 Sum-Product Partition Crossover

The first recombination operator called Sum-Product Partition Crossover (SPPX) employs for offspring generation simple set sum and set product operations on block of partitions and a random mechanism of operand selection from randomly determined 2 parental chromosomes. As a result 0, 2 or 4 children are obtained. The procedure SPPX$(p, r, s_1, t_1, s_2, t_2, sum, product)$ contains two procedures SUM(p, r, s, t) and PRODUCT(p, r, s, t), which are applied to the pair of chromosomes $p = \{V_1^p, \ldots, V_k^p\}$, $r = \{V_1^r, \ldots, V_l^r\}$. Each one may produce a pair of chromosomes $s = \{V_1^s, \ldots, V_m^s\}$ and $t = \{V_1^t, \ldots, V_n^t\}$ with probabilities of elementary operations satisfying $0 <$product $=$ Prob(PRODUCT) $\leq$ *sum* $=$ Prob(SUM) ≤ 1. If new offspring is generated by the procedures SUM or PRODUCT, the variables sum or product are set to 1, respectively. Otherwise they are set to 0.

```
procedure:  SPPX($p, r, s_1, t_1, s_2, t_2, sum, product$)
begin
  $s_1 = t_1 = s_2 = t_2 = \emptyset$;
  generate random numbers $rand_1$, $rand_2 : 0 \leq rand_1, rand_2 \leq 1$;
  if $rand_1 \leq sum$ then begin SUM($p, r, s_1, t_1$); $sum = 1$; end
    else $sum = 0$;
  if $rand_2 \leq product$ then begin PRODUCT($p, r, s_2, t_2$); $product = 1$; end
    else $product = 0$;
end SPPX;
procedure:  SUM($p, r, s, t$)
begin
  select at random $h$ $(1 \leq h \leq k)$ and $j$ $(1 \leq j \leq l)$;
  $V_1^s = V_1^t = (V_h^p \cup V_j^r)$;
  for $i = 1$ to $k$ do
    if $i \neq h$ do if $(V_i^p \setminus V_j^r)$ nonempty then
      add next block $V_i^p \setminus V_j^r$ to $s$;
  for $i = 1$ to $l$ do
    if $i \neq j$ do if $(V_i^r \setminus V_h^p)$ nonempty then
      add next block $V_i^r \setminus V_h^p$ to $t$;
end SUM;
procedure:  PRODUCT($p, r, s, t$)
begin
  select at random $h$ $(1 \leq h \leq k)$ and $j$ $(1 \leq j \leq l)$;
  $V_1^s = V_1^t = (V_h^p \cap V_j^r)$;
  for $i = 1$ to $k$ do
    if $i \neq h$ do if $(V_i^p \setminus V_1^s)$ nonempty then
      add next block $V_i^p \setminus V_1^s$ to $s$;
  for $i = 1$ to $l$ do
    if $i \neq j$ do if $(V_i^r \setminus V_1^t)$ nonempty then
      add next block $V_i^r \setminus V_1^t$ to $t$;
end PRODUCT;
```

Fig. 3. The crossover operator SPPX

A pseudocode of the procedure SPPX is presented in Figure 3. An application of the operator SPPX is shown in Example 1.

Example 1. Two parents represent different 5-colorings of a graph with 10 vertices $p = \{1,8\}\{2,7\}\{3,10\}\{4,6\}\{5,9\}$ and $r = \{1\}\{2,8\}\{3,7,10\}\{4,9\}\{5,6\}$. Let us assume $sum = \text{Prob(SUM)} = 0.8$, $product = \text{Prob(PRODUCT)} = 0.7$.

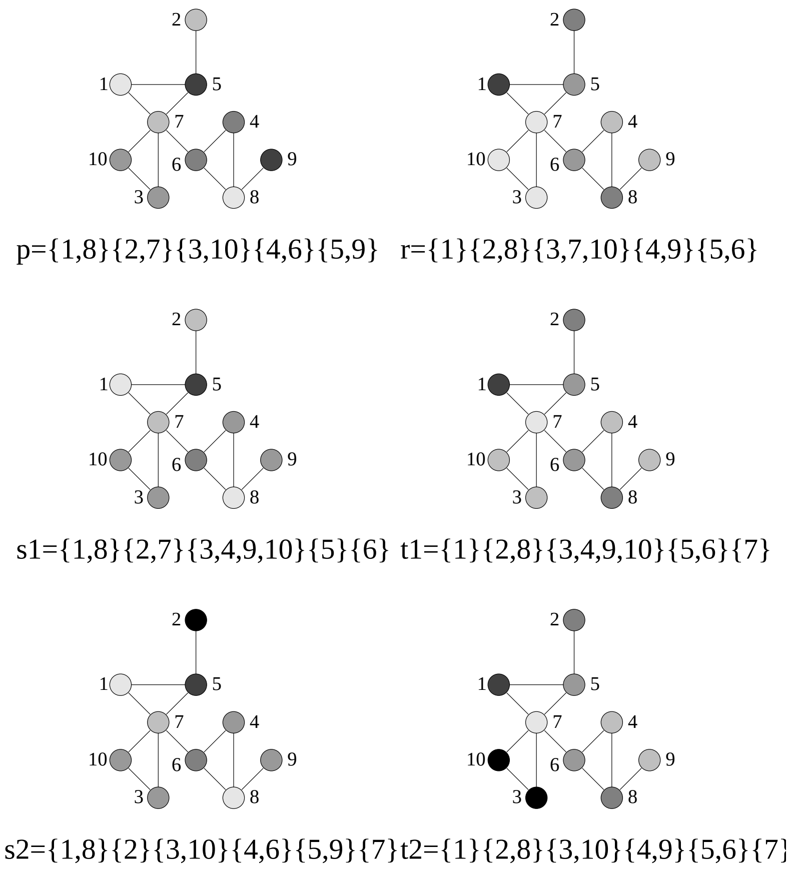

p={1,8}{2,7}{3,10}{4,6}{5,9}
r={1}{2,8}{3,7,10}{4,9}{5,6}
s1={1,8}{2,7}{3,4,9,10}{5}{6}
t1={1}{2,8}{3,4,9,10}{5,6}{7}
s2={1,8}{2}{3,10}{4,6}{5,9}{7}
t2={1}{2,8}{3,10}{4,9}{5,6}{7}

Fig. 4. An illustration of SPPX crossover (see Example 1)

The procedure SPPX$(p, r, s_1, t_1, s_2, t_2, sum, product)$ is called.

Let $rand_1 = 0.3$ and the procedure SUM(p, r, s_1, t_1) be called, where p, r are parents and s_1, t_1 are offspring chromosomes. Let us assume $h = 3$, $j = 4$, where h, j – randomly selected block indices of parental chromosomes p and r, respectively. Thus, the corresponding partition blocks are $V_3^p = \{3, 10\}$ and $V_4^r = \{4, 9\}$.

Their sum gives one common block $V_1^s = V_1^t = \{3, 4, 9, 10\}$ in offspring chromosomes. Then the elements of the common block are removed from p and r and their remaining blocks are copied to offspring chromosomes s_1 and t_1, respectively. After lexicographic reordering of blocks the partitions $s_1 = \{1, 8\}\{2, 7\}\{3, 4, 9, 10\}\{5\}\{6\}$ and $t_1 = \{1\}\{2, 8\}\{3, 4, 9, 10\}\{5, 6\}\{7\}$ are obtained, respectively. Next, the variable sum is set to 1.

Let $rand_2 = 0.4$ and the procedure PRODUCT(p, r, s_2, t_2) be called, where p, r are parents and s_2, t_2 are offspring chromosomes. Let us assume $h = 2$,

$j = 3$, where h, j – randomly selected block indices of parental chromosomes p and r, respectively. Thus, the corresponding partition blocks are $V_2^p = \{2, 7\}$ and $V_3^r = \{3, 7, 10\}$. Their product gives one common block $V_1^s = V_1^t = \{7\}$ in offspring chromosomes. Then the elements of the common block are removed from p and r and their remaining blocks are copied to offspring chromosomes s_2 and t_2, respectively. After lexicographic reordering of blocks the resulting partitions are $s_2 = \{1, 8\}\{2\}\{3, 10\}\{4, 6\}\{5, 9\}\{7\}$ and $t_2 = \{1\}\{2, 8\}\{3, 10\}\{4, 9\}\{5, 6\}\{7\}$, respectively. Finally, the variable product is set to 1.

As a result of the SPPX crossover we obtain four children: s_1, t_1, s_2 and t_2 representing 5-colorings and 6-colorings of the given graph (see Figure 4).

It is observed that operation PRODUCT may increase the initial number of colors while the operation SUM may reduce this number. The probability of PRODUCT should be lower than or equal to the probability of SUM. Since the recombination operator SPPX is oriented not only for coloring problems it can be used as a versatile operator in evolutionary algorithms for many other partition problems.

5.2 Conflict Elimination Crossover

In conflict-based crossovers for GCP the assignement representation of colorings is used and the offspring tries to copy conflict-free colors from the parents. The next recombination operator called Conflict Elimination Crossover (CEX) reveals some similarity to the classical crossover. Each parental chromosome p and r is partitioned into two blocks. The first block consists of conflict-free nodes while the second block is built of the remaining nodes that break the coloring rules.

```
procedure:  CEX($p, r, s, t$)
begin
  $s = r$;
  $t = p$;
  copy block of conflict-free vertices $V_{cf}^p$ from $p$ to $s$;
  copy block of conflict-free vertices $V_{cf}^r$ from $r$ to $t$;
end
```

Fig. 5. The crossover operator CEX

The last block in both chromosomes is then replaced by corresponding colors taken from the other parent. This recombination scheme provides inheritance of all good properties of one parent and gives the second parent a chance to reduce the number of existing conflicts. However, if a chomosome represents a feasible coloring the recombination mechanism will not work properly. Therefore, the recombination must be combined with an efficient mutation mechanism. As a result two chromosomes s and t are produced. The operator CEX is almost as simple and easy to implement as the classical crossover (see Figure 5).

An application of the operator CEX is shown in Example 2.

Example 2. Two parents represent different 5-colorings of a graph with 10 vertices i.e. sequences $p = \langle 5, 2, \mathbf{3}, \mathbf{4}, 1, \mathbf{4}, 2, 5, 1, \mathbf{3} \rangle$, and $r = \langle 1, 4, \mathbf{5}, 2, 3, 3, \mathbf{5}, 4, 2, \mathbf{5} \rangle$. Vertices with color confficts are marked by bold fonts. Thus, the chomosome p has 6 vertices with feasible colors and 4 vertices with color conflicts while the chomosome r has 7 vertices with feasible colors and 3 vertices with color conflicts.

Replacing the vertices with color conflicts by vertices taken from the other parent we obtain the following two chromosomes: $s = \langle 5, 2, \mathbf{5}, 2, 1, 3, 2, 5, 1, \mathbf{5} \rangle$ and $t = \langle 1, 4, \mathbf{3}, 2, 3, 3, 2, 4, 2, \mathbf{3} \rangle$ (see Figure 6).

It is observed that obtained chromosomes represent now two different 4-colorings of the given graph (reduction by 1 with respect to initial colorings) and the number of color conflicts is now reduced to 2 in each chromosome.

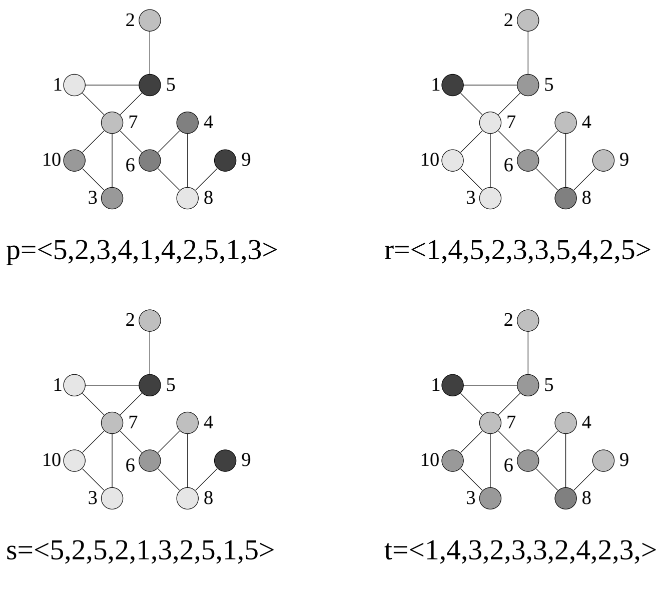


Fig. 6. An illustration of CEX crossover (see Example 2)

5.3 Union Independent Set Crossover

The greedy operator proposed by Dorne and Hao [9] and called Union Independent Sets (UISX) works on pairs of independent sets taken from two parent colorings. In any feasible graph coloring all graph vertices are partitioned into blocks that are disjoint independent sets (ISs). A coloring is not feasible if it contains at least one block which is a non-independent set. Each block of a partition is assigned one color. The authors mentioned that “if we try to maximize the size of each IS by a combination mechanism, we will reduce the sizes of non-independent sets, which in turn helps to push these sets into independent sets" [9].

In the initial step disjoint ISs in both parents are determined. Let us compute coloring for the first child. At first, the maximum IS is selected from the first parent and computed set intersections with ISs from the second parent. The union of a pair of ISs with maximum intersection is colored in the offspring with the IS color from the first parent. In the case of a tie a random IS is always chosen. Then the colored vertices are removed from the both parents and the coloring procedure is repeated as long as posible. The vertices without any color are assigned the original color from the first parent. The coloring for the second child is computed with reversed roles of both parents.

Application of the operator UISX is shown in Example 3.

Example 3. For the given graph with 10 vertices two parents—$p = \{1,8\}\{2,7\}$ $\{5,9\}$ and $r = \{1\}\{2,8\}\{4,9\}\{5,6\}$—represent a partial 3-coloring and a partial 4-coloring, respectively.

The first child is computed as follows. Maximum ISs in p are $\{1,8\}$, $\{2,7\}$ and $\{5,9\}$. Let $\{1,8\}$ be selected. There are two maximum intersections of $\{1,8\}$ and ISs in r, and let $\{2,8\}$ be selected for the union. Thus, $\{1,2,8\}$ is obtained as the first block of s and now $p = \{7\}\{5,9\}$, while $r = \{4,9\}\{5,6\}$. The single maximum IS in p is $\{5,9\}$ and it is selected. There are two maximum intersections of $\{5,9\}$ and ISs in r, and let $\{4,9\}$ be selected for the union. Thus, $\{4,5,9\}$ is obtained as the second block of s and now $p = \{7\}$ and $r = \{6\}$. Repeating the procedure $s = \{1,2,8\}\{4,5,9\}\{7\}$ is received, which is a partial 3-coloring of the given graph.

Similarly the second child is constructed. Maximum ISs in r are $\{2,8\}$, $\{4,9\}$ and $\{5,6\}$. Let $\{2,8\}$ be selected. There exist two maximum intersections of $\{2,8\}$ and ISs in p and let $\{2,7\}$ be selected for the union. Thus, $\{2,7,8\}$ is obtained as the first block of t and now $p = \{1\}\{5,9\}$ and $r = \{1\}\{4,9\}\{5,6\}$. There are two maximum ISs in r: $\{4,9\}$ and $\{5,6\}$ and let $\{5,6\}$ be selected. There is one maximum intersection of $\{5,6\}$ and ISs in r and let $\{5,9\}$ be selected for the union. Thus, $\{5,6,9\}$ is obtained as the second block of t and now $p = \{1\}$ and $r = \{1\}\{4\}$. Repeating the procedure $t = \{1\}\{2,7,8\}\{4\}\{5,6,9\}$ is received, which is a partial 4-coloring of the given graph.

The UISX crossover is depicted in Figure 7. It is observed that obtained partial colorings cover more vertices of the given graph than initial partial colorings and the union operation can lead to bigger ISs.

5.4 Greedy Partition Crossover

The method called Greedy Partition Crossover (GPX) was designed by Galinier and Hao for recombination of colorings or partial colorings in partition representation [12]. It is assumed that both parents are randomly selected partitions with exactly k blocks that are independent sets. The result is a single offspring (a coloring or partial coloring) that is built successively in a greedy way. In each odd

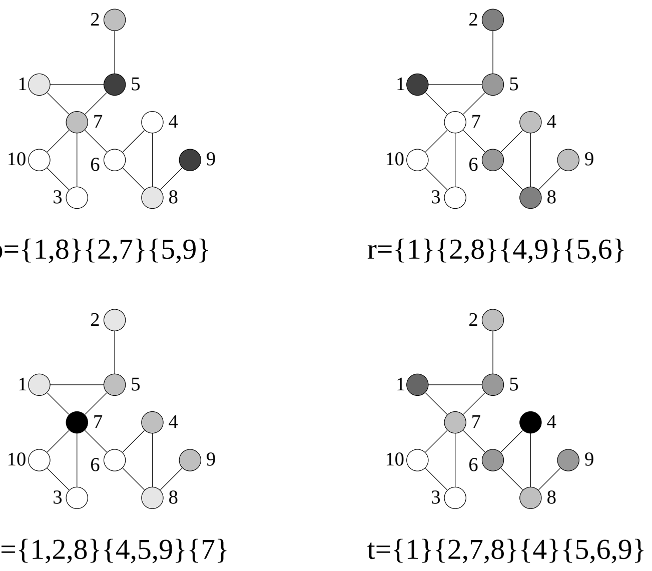


Fig. 7. An illustration of UISX crossover (see Example 3)

step select the maximum block from the first parent is selected. Then the block is added to the result and all its nodes from the both parents are removed. In each even step the maximum block is selected from the second parent. Then the block is added to the result and all its nodes from the both parents are removed. The procedure is repeated at most k times since in some cases the offspring has less blocks than the parents. This possibility is not considered in the original paper [12]. Finally, unassigned vertices (if they exist) are assigned at random to existing blocks of partition. A corrected version of GPX is shown in Figure 8. The first parent is replaced by the offspring while the second parent is returned to population and can be recombined again in the same generation. GPX crossover is performed with a constant probability.

An application of the operator GPX to partition chromosomes is shown in Example 4.

Example 4. Two parents represent different 5-colorings of a graph with 10 vertices, i.e. partitions $p_0 = \{1,8\}\{2,7\}\{3,10\}\{4,6\}\{5,9\}$, $p_1 = \{1\}\{2,8\}\{3,7,10\}\{4,9\}$ $\{5,6\}$.

For $i = 1$ the maximum block $\{3,7,10\}$ is selected from p_1 and is added to s. After removing the block vertices from the parents we obtain $p_0 = \{1,8\}\{2\}\{4,6\}$ $\{5,9\}$, $p_1 = \{1\}\{2,8\}\{4,9\}\{5,6\}$. For $i = 2$ the maximum block $\{1,8\}$ is selected from p_0 and added to s. After removing the block vertices from the parents we obtain $p_0 = \{2\}\{4,6\}\{5,9\}$, $p_1 = \{2\}\{4,9\}\{5,6\}$. For $i = 3$ the maximum block $\{4,9\}$ is selected from p_1 and added to s. After removing the block vertices from

```
procedure:  GPX($p_0, p_1, s, k$)
begin
  $s = \emptyset$;
  $i = 1$;
  repeat
    select block $V$ with maximum cardinality from the partition $p(i$ mod $2)$;
    $s = s \cup V$ -- add the block $V$ to partition $s$;
    remove all vertices of $V$ from $p_0$ and $p_1$;
    $i = i + 1$;
  until $(i > k)$ or (all blocks of $p_1$ and $p_2$ empty);
  assign randomly all unassigned vertices to existing blocks of $s$;
end
```

Fig. 8. The modified crossover operator GPX

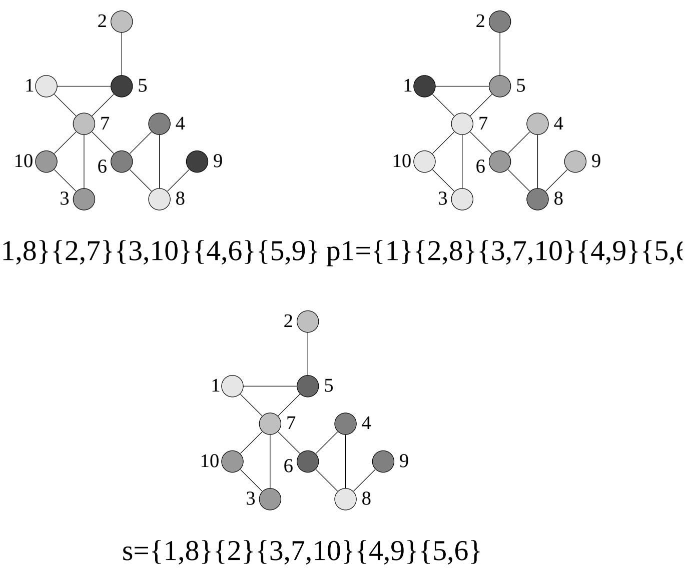

p0={1,8}{2,7}{3,10}{4,6}{5,9}
p1={1}{2,8}{3,7,10}{4,9}{5,(
s={1,8}{2}{3,7,10}{4,9}{5,6}

Fig. 9. An illustration of GPX crossover (see Example 4)

the parents we obtain $p_0 = \{2\}\{6\}\{5\}$, $p_1 = \{2\}\{5,6\}$. Repeating this procedure subsequent block partitions are added to the result. When termination condition is satisfied the partition $s = \{1,8\}\{2\}\{3,7,10\}\{4,9\}\{5,6\}$ is the resulting 5-coloring (see Figure 9).

5.5 Mutation Operators

Two types of mutation operators described in literature are used. Transposition (T) is a classical type of mutation that exchanges colors of two randomly selected vertices in the assignment representation. The second mutation operation called First Fit (FF) is designed for colorings in partition representation and is well suited for GCP. In First Fit mutation one block of the partition is selected at random and we try to make a conflict-free assignment of its vertices to other blocks using the heuristic First Fit. Vertices with no conflict-free assignment remain in the original block. Thus, as a result of the mutation First Fit the color assignment is partially rearranged and the number of partition blocks is often reduced by one.

Example 5. In chromosome $p = \langle 5, 2, \mathbf{3}, 4, 1, 4, 2, \mathbf{5}, 1, 3\rangle$ that represents a 5-coloring of a graph with 10 vertices, Transposition mutation exchanges colors of 2 randomly selected vertices 3 and 8. The resulting chromosome is $s = \langle 5, 2, 5, 4, 1, 4, 2, 3, 1, 3\rangle$ which is another 5-coloring of the given graph in assignment representation. The number of color conflicts has changed from 2 to 1.

In chromosome $r = \{1\}\{2, 8\}\{3, 7, 10\}\{4, 9\}\{5, 6\}$ that represents a 5-coloring of a graph with 10 vertices, the mutation First Fit performs a conflict-free assignment of vertices from the maximum partition block, i.e. $\{3, 7, 10\}$ to all remaining blocks. The resulting chromosome is $t = \{1, 3\}\{2, 7, 8\}\{4, 9, 10\}\{5, 6\}$ which is a conflict-free 4-coloring of the given graph in partition representation. The number of color conflicts has been reduced by 3.

Both mutations operators are shown in Figure 10.

5.6 Selection Operator

Selection process maintains constant size of population selected by means of a fitness function.

The quality of a solution is measured by the following cost function:

$$f(p) = \sum\nolimits_{(u,v)\in E} q(u, v) + d + k, \text{ where:}$$

p – is a graph coloring,
q – is a penalty function for pairs of vertices connected by an edge $(u, v) \in E$:

$$q(u, v) = \begin{cases} 2, & \text{when} \quad c(u) = c(v) \\ 0, & \text{otherwise} \end{cases}$$

d – is a general penalty function applied to graph colorings:

$$d = \begin{cases} 1, & \text{when} \quad \sum_{(u,v)\in E} q(u, v) > 0 \\ 0, & \text{when} \quad \sum_{(u,v)\in E} q(u, v) = 0 \end{cases},$$

k – is the number of colors used.

In many cases less colors cause more conflicts. Modeling the cost function we can favour conflict-free colorings by setting values of $q(u, v)$ and d. On the other

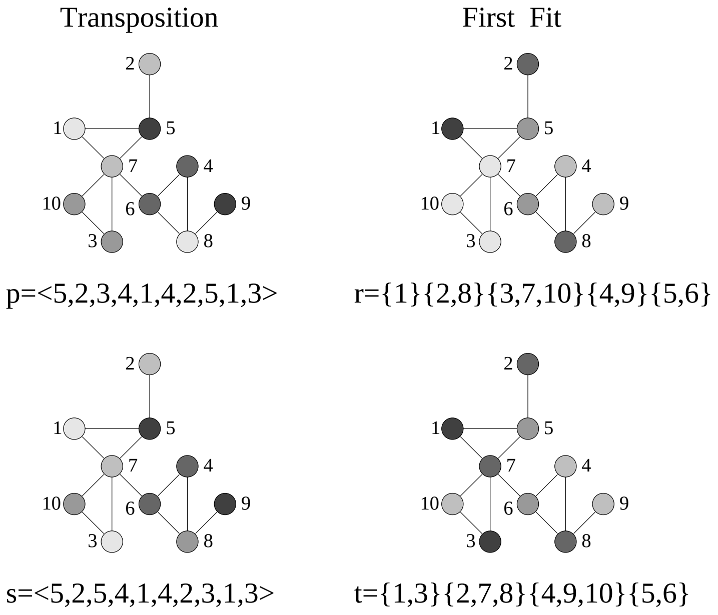


Fig. 10. An illustration of mutation Transposition and First Fit (see Example 5)

hand conflict colorings with less colors can also be useful. Therefore, we decided to set relatively low values of $q(u, v)$ and d. Thus, with the cost function given above, all k'-colorings with i conflicts, $k' \leq (k - 2i - 2)$, are better than conflict-free k-colorings.

The proportional (roulette) selection is performed in two phases of the algorithm (see Figure 2) with the fitness function $1/f(p)$.

6 EXPERIMENTAL VERIFICATION

For computer experiments nine graph instances were used that are available in the web archives [35, 36]. They are collections of graphs in DIMACS format with known parameters m, n and usually $\chi(G)$. One random graph $r.50$ with density 50 % was used from authors' test set.

In our program *PGA for GCP* two basic models of PGA—migration and master-slave—can be simulated. It is possible to set up most parameters of evolution, monitor evolution process on each island and measure both the number of generations and time of computations. In order to avoid misunderstanding we always report throughout the paper the total execution time of the sequential simulation of the PGA.

In the preprocessing phase we converted list of edges representation into adjacency matrix representation. The program generates detailed reports and basic statistics [22]. All computer experiments were performed on a computer with Pentium 4 processor (3 GHz, 1 GB RAM).

The research was focused on the migration model of PGA. In the experiments the following aspects of this model were taken into consideration:

1. comparison of PGA versus GA,
2. cost of best individuals in concurrent evolutions on isolated islands,
3. cost of optimal solution versus distance between migrations,
4. influence of migration scheme on PGA efficiency, measured by number of iterations required for finding the first conflict-free and optimal solution,
5. comparison of four selected crossover operators.

In all experiments and for all crossover operators we used constant *crossover probability* = 0.8 and *mutation probability* = 0.1. In SPPX crossover *prob*(PRODUCT) = *prob*(SUM) = 0.8.

In the first experiment the migration PGA was tested against traditional GA. GA was obtained as a special case in the program *PGA for GCP* with parameters *number of islands* = 1, *migration size* = 0 and *population size* = 100. In migration-based PGA three islands with *subpopulation size* = 33 and migration of best individuals with *migration distance* = 10 and *migration size* = 5 were selected. Both algorithms terminated after constant *number of iterations* = 5000 or after finding an optimum coloring. Initial population was generated at random. Very efficient operators GPX crossover and First Fit mutation were applied for recombination. For testing a popular graph queen8.8 ($n = 64$, $m = 728$, $\chi(G) = 9$) was used. All tests were repeated 30 times. The results of comparison are presented in Table 1.

graph $G(V,E)$	algo-rithm	colo-rings		cost/number of iterations/time			
				min	max	avg.	std. dev.
*queen*8.8 $\|V\| = 64$ $\|E\| = 728$ $\chi(G) = 9$	GA	30/30 conflict-free	cost	9	11	10.3	0.6
			it	2 190	5 000	4 838	626
			t[s]	44	102	90.4	11.7
		2/30 optimal (7 %)	cost	9	9	9	0
			it	2 190	2 936	2 563	528
			t[s]	44	57	50.5	9.2
	PGA	30/30 conflict-free	cost	9	11	10.1	0.7
			it	340	5 000	4 516	1 178
			t[s]	9	132	108.5	28.3
		6/30 optimal (20 %)	cost	9	9	9	0
			it	340	4 290	2 580	1 559
			t[s]	9	103	61.3	35.1

Table 1. Comparison of GA and PGA

Both algorithms produced only conflict-free colorings but the number of optimal solutions was 3 times greater with PGA (6 out of 30). Although the total population size and the number of iterations *it* were equivalent in the algorithms, the sequential simulation of PGA consumed in average 20 % of execution time more than GA. With respect to other data, GA and PGA revealed similar features. Therefore, the main adventage of PGA is better quality of the solution.

The second experiment concluded with observation that best individuals on isolated islands reveal significant differences of their cost functions. Experiments were conducted on two graphs: randomly generated graph r.50 with density 50 % ($n = 100$, $m = 2\,474$, $\chi(G) \leq 17$) and graph le450.15b ($n = 450$, $m = 8169$, $\chi(G) = 15$). Cost of best solutions was measured after constant *number of iterations* = 150 on each island with the following PGA parameters: *number of islands* = 3, *subpopulation size* = 60, migration of best individuals with *migration distance* = 10 and *migration size* = 5, *crossover* = GPX, CEX, *mutation* = FF, random initial populations with *initial number of colors* = 4.

graph	cross-	itera-	Best cost on island			itera-	Best cost on island		
$G(V,E)$	over	tions	1	2	3	tions	1	2	3
r.50	GPX	0	1 171	1 169	1 161	80	31	20 +	25
$\lvert V\rvert = 100$		10	79	69	96	90	31	20	25
$\lvert E\rvert = 2\,474$		20	57	45	79	100	31	20	25
$\chi(G) \leq 17$		30	43	35	48	110	29	20	25
		40	31	34	28	120	25	20	20 +
		50	31	34	25	130	23	20	20
		60	31	32	25	140	23	20	20
	time = 31 s	70	31	32	25	150	23	19	20
	CEX	0	1 171	1 145	1 175	80	21	19	20
		10	195	20 +	21 +	90	21	19	20
		20	195	20	21	100	20	19	20
		30	21 +	19	21	110	20	19	20
		40	21	19	20	120	20	19	20
		50	21	19	20	130	20	19	20
		60	21	19	20	140	20	19	20
	time = 3.2 s	70	21	19	20	150	20	19	20
le450.15b	GPX	0	3 939	3 911	3 901	80	25	20 +	23
$\lvert V\rvert = 450$		10	645	512	297	90	25	19	22
$\lvert E\rvert = 8\,169$		20	408	332	159	100	25	19	22
$\chi(G) = 15$		30	230	218	37	110	25	19	22
		40	128	158	26	120	25	19	22
		50	84	105	23	130	25	19	22
		60	58	74	23	140	25	19	22
	time = 325 s	70	32	51	23	150	25	19	19 +
	CEX	0	3 885	3 949	3 917	80	19	19	19
		10	799	1 013	880	90	19	19	19
		20	20 +	20 +	20 +	100	19	19	19
		30	20	20	20	110	19	19	19
		40	20	19	20	120	19	19	19
		50	19	19	20	130	19	19	19
		60	19	19	20	140	19	19	19
	time= 32 s	70	19	19	20	150	19	19	19

Table 2. Concurrent evolution on isolated islands

The obtained results are shown in Table 2. "+" denotes the first conflict-free coloring received on the given island.

Evolutions on isolated islands do not influence each other and run with different speed. It is observed that for the two test graphs equivalent optimization results are obtained faster with CEX crossover than with GPX, both in terms of the required number of iterations and the computation time. One run of PGA with GPX crossover takes approximately as much time as 10 runs of PGA with CEX.

By introducing migration between islands the subpopulations become more homogenous, since more "advanced" islands pass their representatives to "backward" islands. At a later stage, with similar results on all islands, sufficient diversity for further improvement can be provided by rare migrations.

The distance between migrations is an important factor in PGA. Experiments were conducted on the same test graphs, i.e. r.50 and le450.15b. Cost of optimal solutions was measured on each island after constant *number of iterations* = 200. All combinations of four crossover and two mutation operators were tested with the following parameters: *number of islands* = 3, *subpopulation size* = 60, migrations of best individuals *migration size* = 5, *mutation* = FF, *initial number of colors* = 5. All experiments were repeated 10 times. The obtained results are shown in Table 3.

graph $G(V,E)$	cross-over	muta-tion	distance between migrations					
			2	5	10	20	50	100
r.50	UISX	T	731	744	753	759	778	791
$\lvert V\rvert = 100$		FF	18.5	18.7	18.4	18.7	19.0	18.8
$\lvert E\rvert = 2\,474$	GPX	T	804	799	797	803	811	812
$\chi(G) \leq 17$		FF	19.1	19.0	18.4	19.0	19.0	18.8
	SPPX	T	148	149	158	145	151	143
		FF	23.2	26.6	28.3	32.5	28.1	24.6
	CEX	T	856	854	856	856	866	863
		FF	18.9	18.9	19.0	19.0	18.6	19.0
le450.15b	UISX	T	2 705	2 678	2 775	2 760	2 813	2 798
$\lvert V\rvert = 450$		FF	18.8	19.6	19.2	19.0	19.0	18.8
$\lvert E\rvert = 8\,169$	GPX	T	2 557	2 595	2 639	2 631	2 622	2 606
$\chi(G) = 15$		FF	19.1	19.0	18.8	19.2	19.0	18.8
	SPPX	T	632	574	551	560	565	584
		FF	36.5	41.6	37.4	37.0	41.2	34.6
	CEX	T	2 761	2 796	2 834	2 857	2 887	2 907
		FF	18.8	19.0	18.9	19.1	18.9	18.9

Table 3. Average cost of best solution versus distance between migrations

One can expect that too frequent migrations reduce diversity of populations while too rare migration influences the convergence of PGA. It is observed that in some cases for the given graph instance and some combinations of operators there exists a single optimal value of the distance between migrations. Below and above this optimum the same number of iterations *it* gives worse results. The examples are as follows: opt{r.50, UISX, T} = 2, opt{r.50, GPX, T} = 10, opt{r.50, GPX, FF} = 10, opt{r.50, CEX, T} = 5, opt{le450.15b, UISX, T} = 2, opt{le450.15b, SPPX, T} = 10, opt{le450.15b, CEX, T} = 2. More such cases appear for Transposition mutation. For the First Fit mutation such regularities are very rare. In conclusion, Table 2 provides a strong evidence that First Fit mutation is always better than Transposition but no direct hint about the optimum distance of migrations can be derived.

The influence of migration scheme on the PGA efficiency was tested and measured by the number of iterations *it* needed to obtain an optimal coloring (graph chromatic numbers $\chi(G)$ are known). Since we must not expect that the optimum coloring could always be obtained, even if extremally long time of computations is allowed, the termination condition was either 500 iterations (5000 iterations for SPPX) or conflict-free coloring. Only the optimal colorings were selected.

The experiments were performed on graphs *games*120, *myciel*7 and *mulsol.i.*4 for all four crossover operators with the following parameters: *number of islands* = 5, *subpopulation size* = 60, *migration rate* = 5, *mutation* = FF. All experiments were repeated 30 times and average number of iterations was computed.

The obtained results are shown in Table 4.

For most crossover operators migration of best individuals usually gives the best results. One exception is CEX crossover. Migration of random individuals is often a very good choice for CEX operator. No migration gives the worst results.

In the main experiment the efficiency of all four crossover operators was tested. The number of iterations *it* and the computation time were measured ($t = 0$ denotes the computation time less then 1 [s]). The termination condition was the first conflict-free coloring generated by the algorithm or *number of iterations* = 500. The percentage of optimal colorings was determined.

Simulations were performed on five graphs *anna*, *david*, *miles*500, *myciel*7 and *mulsol.i.*1 with the following parameters: *number of islands* = 3, *subpopulation size* = 60, migration of best individuals with *migration rate* = 5, *migration size*= 5, *mutation* = First Fit, *initial number of colors* = 4. All experiments were repeated 30 times. The results are presented in Table 5.

It is observed that the proposed crossover operators differ in terms of computation time and efficiency. Under the first criterion we receive ordering CEX, GPX, SPPX and UIXX. Under the second criterion we receive another ordering {GPX, CEX}, UISX and SPPX. The only operator that wins both classifications for two graphs is CEX. For the remaining three graphs CEX takes the second place in efficiency and the first one in speed. GPX two times takes the first place in efficiency and the second in speed. UISX shows such result in one case.

For the selected graphs simple SPPX crossover requires more than 500 iterations. Hence, the number of optimal colorings is 0 % for this operator. SPPX never wins any of the above classifications.

The most efficient operator in the experiment is CEX which dominates all other operators under the time criterion and is approximately as good as GPX in terms of efficiency. Generalization of this statement requires further research with much more graphs.

graph $G(V,E)$	cross-over	migra-tion	number of iterations			
			min	max	avg.	std. dev.
$games120$	UISX	best	1	10	4.2	2.0
$\|V\| = 120$		random	1	10	4.2	2.3
$\|E\| = 638$		none	1	37	6.2	7.9
$\chi(G) = 9$	GPX	best	5	10	6.2	1.9
		random	5	15	10.2	3.9
		none	5	68	27.9	16.0
	SPPX	best	5	50	20.1	13.2
		random	5	65	34.7	22.5
		none	9	274	111	72.8
	CEX	best	4	165	25.1	39.3
		random	5	135	17.2	25.4
		none	5	501	81.4	138.1
$myciel7$	UISX	best	2	5	4.4	0.9
$\|V\| = 191$		random	2	10	4.9	2.0
$\|E\| = 2360$		none	2	28	6.2	6.0
$\chi(G) = 8$	GPX	best	4	11	6.6	2.3
		random	5	35	8.8	5.7
		none	5	119	29.8	23.6
	SPPX	best	5	80	39.8	21.0
		random	10	115	61.0	37.6
		none	27	371	158	79.2
	CEX	best	5	45	10.2	10.0
		random	4	46	10.7	11.8
		none	5	101	20.3	26.0
mulsol.i.4	UISX	best	2	177	26.0	38.0
$\|V\| = 185$		random	2	196	28.6	41.5
$\|E\| = 3\,946$		none	2	500	105.4	146.9
$\chi(G) = 31$	GPX	best	15	180	68.8	45.0
		random	5	285	121.4	66.9
		none	83	440	232.3	89.3
	SPPX	best	73	1 532	533.1	330.1
		random	280	1 770	817.1	382.9
		none	171	5 000	2 591.5	1 433.6
	CEX	best	5	210	59.1	58.9
		random	5	195	38.6	61.6
		none	6	498	134.7	162.0

Table 4. Influence of migration schemes on performance of migration based PGAs

graph	cross-	opt. colorings			time/number of iterations				time
$G(V,E)$	over	[%]	rank		min	max	avg.	std. dev.	rank
anna	UISX	60	3	t[s]	0	34	4.3	7.2	4
$\|V\| = 138$				it	2	59	9.9	11.8	
$\|E\| = 494$	GPX	80	2	t[s]	0	7	1.8	1.8	3
$\chi(G) = 11$				it	5	37	11.1	7.4	
	SPPX	20	4	t[s]	0	4	1.2	1.0	2
				it	5	99	35.4	24.4	
	CEX	100	1	t[s]	0	4	0.6	1.1	1
				it	4	190	27.8	51.1	
david	UISX	35	3	t[s]	0	10	2.8	2.8	4
$\|V\| = 87$				it	1	35	11.3	9.0	
$\|E\| = 406$	GPX	75	1	t[s]	0	2	0.8	0.7	2
$\chi(G) = 11$				it	5	45	11.6	8.1	
	SPPX	5	4	t[s]	0	10	1.1	1.8	3
				it	5	115	38.7	24.1	
	CEX	63	2	t[s]	0	3	0.5	0.8	1
				it	4	205	29.9	51.1	
$miles500$	UISX	4.4	4	t[s]	1	42	15.3	23.1	4
$\|V\| = 128$				it	2	31	12.7	16.0	
$\|E\| = 1\,170$	GPX	15	1	t[s]	2	12	6.3	3.8	2
$\chi(G) = 20$				it	8	50	25.8	14.5	
	SPPX	10	3	t[s]	5	19	13.2	5.3	3
				it	110	405	279	109	
	CEX	11	2	t[s]	0	19	2.8	3.8	1
				it	5	376	100	106	
$myciel7$	UISX	93	1	t[s]	0	4	1.6	1.2	2
$\|V\| = 191$				it	2	6	4.0	1.2	
$\|E\| = 2\,360$	GPX	67	3	t[s]	0	8	2.4	1.4	3
$\chi(G) = 8$				it	4	25	8.8	2.8	
	SPPX	17	4	t[s]	0	7	3.5	1.9	4
				it	5	100	55.7	27.0	
	CEX	80	2	t[s]	0	2	0.5	0.8	1
				it	4	58	10.6	15.0	
mulsol.1	UISX	57	3	t[s]	1	53	18.8	18.9	2
$\|V\| = 197$				it	1	10	6.0	3.5	
$\|E\| = 3\,925$	GPX	100	1	t[s]	5	354	95.6	19.7	4
$\chi(G) = 49$				it	20	302	142	23.0	
	SPPX	0	4	t[s]	46	52	48.9	1.8	3
				it	500	500	500	0.0	
	CEX	100	1	t[s]	0	21	3.6	5.2	1
				it	5	490	80.7	120.8	

Table 5. Performance of the migration based PGAs with various crossover operators

7 CONCLUSIONS

Simulation experiments reported in the paper provide an evidence that parallel genetic algorithms can be efficiently used for a class of graph coloring problems. In island model of PGA the migration between co–evolving subpopulations can improve the overall convergence of the algorithm.

The presented results justify several conclusions:

1. In island model migration is always better then isolation. In some cases, mostly for Transposition mutation, there exists an optimal migration rate with best algorithm efficiency measured in iterations required for finding an optimal solution. In most cases, in particular for First Fit mutation, no optimal migration rate was found.
2. For almost all tested graphs migration of best individuals is always the most efficient. Migration of random individuals is efficient in some cases with CEX crossover.
3. Mutation First Fit in coloring problems always outperforms Transposition and it is best combined with CEX or GPX operator. The Transposition mutation works relatively well with SPPX crossover.
4. The best crossover operators under efficiency criterion are CEX and GPX with the First Fit mutation. The third place is occupied by UISX. SPPX is very fast but requires much more iterations to be efficient. It is the worst crossover in our experiments.
5. For all tested graphs the fastest crossover operator under the time criterion is CEX with the First Fit mutation. In most cases the slowest operator is UISX. GPX and SPPX occupy places in the middle.
6. In migration model the solution space is searched efficiently due to co-evolution in subpopulations and periodic migration. Theoretical basis for this model is given in [6].
7. In migration model efficient convergence towards optimal solution is observed and PGA better escapes local maxima than GA.
8. The role of migration during algorithm execution is changing from a mechanism for fast improvement of "late" islands in the initial phase, to intensive exploration of the most attractive regions of the search space at a later stage of the co–evolutionary process.
9. Advantages of the migration model are present also in simulated co-evolution performed by a sequential algorithm.

Migration model of PGA is shown to be efficient for hard problems like GCP. The results presented in this paper encourage further research in this area. One obvious direction is to extend the experiments on other DIMACS benchmarks including a class of random graphs. It is also worth to consider some variants of SPPX operator that will make it more "intensive" and problem-oriented. The search for new efficient genetic operators and development of better evolutionary techniques for solving GCP still remain open questions.

Acknowledgements

The authors are grateful to anonymous referees for their constructive criticism and valuable comments that were useful in reworking the experimental part and preparing the final version of the paper.

REFERENCES

[1] Alba, E.—Tomasini, M.: Parallelism and Evolutionary algorithms, IEEE Trans. Evol. Comput., Vol. 6, 2002, No. 5, pp. 443–462.

[2] Bäck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press, 1996.

[3] Barbucha, D.—Jędrzejowicz, P.—Ratajczak, E.—Forkiewicz, M.: Population Learning Algorithm Versus Evolutionary Computation. 15th International Parallel and Distributed Processing Symposium, IPDPS '2001, Workshop on Biologically Inspired Solutions to Parallel Processing Problems, Proceedings, IEEE Comput. Society, 2001, pp. 1–8 (CD-ROM edition).

[4] Cantú-Paz, E.: A Survey of Parallel Genetic Algorithms. Calculateurs Paralleles, Reseaux et Systems Repartis Vol. 10, 1998, No. 2, Paris, Hermes, pp. 141–171.

[5] Cantú-Paz, E.: Topologies, Migration Rates, and Multi-Population Parallel Genetic Algorithms. Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '99, Morgan Kaufmann, 1999, pp. 91–98.

[6] Cantú-Paz, E.: Efficient and Accurate Parallel Genetic Algorithms. Kluwer, 2000.

[7] Cantú-Paz, E.—Goldberg, D. E.: Parallel Genetic Algorithms: Theory and Practice. Computer Methods in Applied Mechanics and Engineering, Elsevier, 2000.

[8] Croitoriu, C.—Luchian, H.—Gheorghies, O.—Apetrei A.: A New Genetic Graph Coloring Heuristic. Computational Symposium on Graph Coloring and Generalizations COLOR '02. In: Constraint Programming, Proceedings of the International Conference, CP '02, 2002.

[9] Dorne, R.—Hao, J.-K.: A New Genetic Local Search for Graph Coloring. Parallel Problem Solving from Nature, Proceedings of the International Conference, PPSN '98, LNCS, Vol. 1498, 1998, pp. 745–754.

[10] Filho, G. R.,—Lorena, L. A. N.: Constructive Genetic Algorithm and Column Generation: An Application to Graph Coloring. Proc. Asia Pacific Operarions Research Symposium, APORS '2000, 2000.

[11] Fleurent, C.—Ferland, J. A.: Genetic Algorithms and Hybrids for Graph Coloring. Annals of Operations Research Vol. 63, 1996, pp. 437–461.

[12] Galinier, P.—Hao, J.-K.: Hybrid Evolutionary Algorithms for Graph Coloring. J. Combinatorial Optimization, 1999, pp. 374–397.

[13] Garey, R.—Johnson, D. S.: Computers and Intractability. A Guide to the Theory of NP-Completeness. Freeman, 1979.

[14] Glass, C. A.—Prügel-Bennett, A.: Genetic Algorithm for Graph Coloring: Exploration of Galinier and Hao's Algorithm, J. Combinatorial Optimization, 2003, pp. 229–236.

[15] Hutchinson, G.: Partitioning Algorithms for Finite Sets. Comm. ACM No. 6, 1963, pp. 613–614.

[16] Iorio, A.—Li, X.: Parameter Control with a Co-Operative, Co-Evolutionary Genetic Algorithm. Parallel Problem Solving from Nature, Proceedings of the International Conference, PPSN '2002, LNCS, Vol. 2439, 2002, pp. 247–256.

[17] Jensen, T. R.—Toft, B.: Graph Coloring Problems. Wiley Interscience, 1995.

[18] Johnson, D. S.,—Trick, M. A.: Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge. DIMACS Series in Discr. Math. and Theor. Comp. Sc., Vol. 26, 1996.

[19] Karp, R. M.: Reducibility Among Combinatorial Problems. In: Miller R. E. and Thatcher J. W. (Eds.), Complexity of Computer Computations, Plenum Press, 1972, pp. 85–103.

[20] Khuri, S.—Walters, T.—Sugono, Y.: Grouping Genetic Algorithm for Coloring Edges of Graph, Proc. 2000 ACM Symposium on Applied Computing, 2000, pp. 422–427.

[21] Kokosiński, Z.: Effects of Versatile Crossover and Mutation Operators on Evolutionary Search in Partitions and Permutation Problems, Intelligent Information Processing and Web Mining. Proceedings of the International IIS: IIPWM '2005 Conference, Advances in Soft Computing, Springer, 2005, pp. 299–308.

[22] Kokosiński, Z.—Kołodziej, M.—Kwarciany, K.: Parallel Genetic Algorithm for Graph Coloring Problem. Computational Science, Proceedings of the International Conference, ICCS '2004, LNCS, Vol. 3036, 2004, pp. 215–222.

[23] Kolodziej, M.: Genetic Algorithms for Edge Coloring Problem. M. S. Thesis. Cracow University of Technology, Kraków, 2003 (in Polish).

[24] Kubale, M.: Introduction to Computational Complexity and Algorithmic Graph Coloring. GTN, Gdańsk, 1998.

[25] Kubale, M. (Ed.): Graph Colorings. American Mathematical Society, 2004.

[26] Kwarciany, K.—Kokosiński, Z.: Parallel Genetic Algorithms in Application to Coloring of Graph Vertices. Technologie Informacyjne, Proceedings of the 2nd National Conference, TI '2004, Scientific Papers of ETI Faculty, Gdańsk University of Technology No. 5, 2004, pp. 747–754 (in Polish).

[27] Levine, D.: A Parallel Genetic Algorithm for the Set Partitioning Problem. Argonne Nat. Lab., Argonne, IL, 1996.

[28] Lis, J.: Parallel Genetic Algorithm with the Dynamic Control Parameter. Evolutionary Computation, ICEC '1996, Proceedings of the International Conference, IEEE Computer Society, 1996, pp. 324–329.

[29] Lorena, L. A. N.—Filho, G. R.: Constructive Genetic Algorithm for Graph Coloring. Proc. Asia Pacific Operarions Research Symposium, APORS '97, 1997.

[30] Nowostawski, M.—Poli, R.: Parallel Genetic Algorithm Taxonomy. Knowledge-based Intelligent Information Engineering Systems, KES '99, Proceedings of International Conference, IEEE, 1999, pp. 88–92.

[31] Nowostawski, M.—Poli, R.: Dynamic Demes Parallel Genetic Algorithm. Knowledge-based Intelligent Information Engineering Systems, KES '99, Proceedings of International Conference, IEEE, 1999, pp. 93–98.

[32] Szyfelbein, D.: Genetic Algorithms for Graph Coloring. Neural Networks and Their Applications, Proceedings of the Conference, Polish Neural Network Society, 1999, pp. 605–610.

[33] Szyfelbein, D.: Metaheuristics in Graph Coloring. In: Kubale M. (Ed.): Discrete Optimization. Models and Methods for Graph Coloring. WNT, Warszawa, 2002, pp. 26–52 (in Polish).

[34] de Werra, D.: Heuristics for Graph Coloring. In: Tinhofer, G. et al. (Eds.) Computational Graph Theory, Springer-Verlag, 1990, pp. 191–208.

[35] COLOR web site.
Availaible on: `http://mat.gsia.cmu.edu/COLOR/instances.html`.

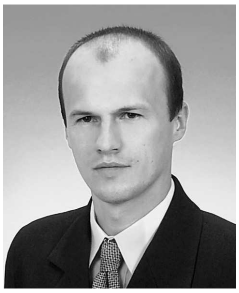
[36] DIMACS ftp site.
Availaible on: `ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/`.

[37] COLORING3 web site. Availaible on: `http://mat.gsia.cmu.edu/COLORING03/`.

Zbigniew Kokosiński received his M. Sc. degree in 1982 from Cracow University of Technology, Kraków, Poland. In 1992 he received Ph. D. degree with distinction in computer science from the Gdańsk University of Technology, Gdańsk, Poland. In 1994–1997 he was employed as an Assistant Professor at the Department of Computer Software, University of Aizu, Aizu-Wakamatsu, Japan. Currently, dr. Kokosiński is an Assistant Professor at the Department of Control Engineering, Cracow University of Technology, Cracow.

His research is focused on combinatorial optimization and metaheuristics, generation of combinatorial objects in parallel, associative processors and algorithms, programmable devices and systems. The publications include over 30 refereed papers in scientific journals and conference proceedings.

In the past years he was a member of IASTED, ACM and IEEE Computer Society.



Krzysztof Kwarciany graduated from Cracow University of Technology (Politechnika Krakowska), Kraków, Poland in 2003, where he received M. Sc. degree in electrical engineering. Currently he works as a control engineer in NGK Ceramics, Gliwice, Poland, where he is a member of the maintenance team. His interests include automatic control, computer programming and genetic algorithms.

Marcin Kołodziej graduated from Cracow University of Technology (Politechnika Krakowska), Kraków, Poland in 2003, where he received M. Sc. degree in electrical engineering. Currently he works as a service engineer in Electra, Rzeszów, Poland. His interests include control engineering, teleinformatics and artificial intelligence.