Diversity Control and Multi-Parent Recombination for Evolutionary Graph Coloring Algorithms

Daniel Cosmin Porumbel[1], Jin-Kao Hao[1], and Pascale Kuntz[2]

[1] LERIA, Université d'Angers, 2 Bd Lavoisier, 49045 Angers, France
[2] LINA, Polytech'Nantes, rue Christian Pauc, 44306 Nantes, France

Abstract. We present a hybrid evolutionary algorithm for the graph coloring problem (Evocol). Evocol is based on two simple-but-effective ideas. First, we use an enhanced crossover that collects the *best* color classes out of *more than two* parents; the best color classes are selected using a ranking based on both class fitness and class size. We also introduce a simple method of using distances to assure the population diversity: at each operation that inserts an individual into the population or that eliminates an individual from the population, Evocol tries to maintain the distances between the remaining individuals as large as possible. The results of Evocol match the best-known results from the literature on almost all difficult DIMACS instances (a new solution is also reported for a very large graph). Evocol obtains these performances with a success rate of at least 50%.

1 Introduction

The graph coloring problem is one of the first problems proved to be NP-complete in the early 70's. It has a very simple formulation: label the vertices of a graph with the minimum number of colors such that no adjacent vertices share the same color. Many other problems and practical applications can be reduced to graph coloring: scheduling and timetabling problems, frequency assignment in mobile networks, register allocation in compilers, air traffic management, to name just a few.

The second DIMACS Implementation Challenge [13] introduced a large set of graphs for benchmarking coloring algorithms that has been extensively used since 1996. The most popular coloring algorithms belong to three main solution approaches: (i) sequential construction (very fast methods but not particularly efficient), (ii) local search methods (many different techniques [1, 2, 9, 11, 12, 15] can be found in the literature), and (iii) the population-based evolutionary methods that traditionally dominate the tables with the best results [4, 6, 7, 8, 14, 15].

We present in this work-in-progress paper a new hybrid evolutionary algorithm (Evocol) that makes contributions in two directions: the recombination operator (Section 3) and the population diversity control (Section 4). The recombination

C. Cotta and P. Cowling (Eds.): EvoCOP 2009, LNCS 5482, pp. 121–132, 2009.

operator picks up the best color classes out of many (and *surely diverse*) parents; the classes are ranked according to their fitness and size. The diversity control uses a set-theoretic distance measure between colorings to strictly control which individuals are inserted or deleted from the population. As such, it keeps the distances between the population individuals as large as possible and it permanently guarantees global diversity. The resulting algorithm is quite simple and lightweight, as it incorporates no important additional elements. However, it obtains very competitive results (Section 5), with plenty of room for further improvement.

2 Problem Statement and Generic Hybrid Algorithm

Given a graph $G(V, E)$, the graph coloring problem requires finding the minimal number of colors χ (the chromatic number) such that there exists a vertex coloring (using χ colors) with no adjacent vertices of the same color (with no conflicts). One could determine the chromatic number by iteratively solving the following k-coloring problem: given a number of colors $k \geq \chi$, find a k-coloring (a coloring using k colors) without conflicts. This method starts with a sufficiently large k (e.g. $k = |V|$ is surely enough) and iteratively decrements k each time the corresponding k–coloring problem is solved. The k–coloring problem becomes increasingly difficult until the algorithm can no longer solve it.

A common coloring representation consists in a function $I : V \rightarrow \{1, 2, \cdots, k\}$, usually encoded as an array of colors $I = [I(1), I(2), \ldots, I(|V|)]$. While we also encoded this representation in our programs, it is very useful to interpret a coloring as a vertex set partition.

Definition 1. *(Partition representation) A k-coloring I of V is denoted by a partition $\{I^1, I^2, \ldots, I^k\}$ of V—i.e. a set of k disjoint subsets (color classes) of V covering V such that $\forall x \in V$, $x \in I^i \Leftrightarrow I(x) = i$.*

We say that I is an individual (a candidate solution for a k-coloring problem); I^i is the color class i induced by the coloring I, i.e. the set of vertices having color i in I. This partition based definition is particularly useful to avoid symmetry issues arising from the color based encoding. As such, it is used for the crossover operator (see Section 3) and also to define a meaningful distance between colorings (see Section 4.1). Moreover, I is a legal or conflict-free coloring (a solution) if and only no color class of I contains adjacent vertices.

Definition 2. *(Conflict number and fitness function) Given an individual I, we call conflict (or conflicting edge) any edge having both ends in the same class. The set of conflicts is denoted by $C(I)$ and the number of conflicts (i.e. $|C(I)|$—also referred to as the conflict number of I) is the fitness function $f(I)$. A conflicting vertex is a vertex $v \in V$, for which there exists an edge $\{v, u\}$ in $C(I)$.*

In this paper, we deal with the k-coloring problem as an optimization problem: given a pair (G, k), our algorithm searches the search space (call it Ω) for a k-coloring I^* such that $f(I^*) = Minf(I)$; if $f(I^*) = 0$, a legal coloring is found.

2.1 General Design of the Evolutionary Algorithm

The generic algorithmic template of Evocol shares some basic ideas with other evolutionary algorithms from the graph coloring literature [2, 4, 6, 7, 8, 14, 15], but we enriched the traditional template with new features: (i) the possibility to combine $n \geq 2$ parents to generate an offspring, (ii) the possibility to reject an offspring if it does not fit some diversity criteria (with respect to the existing individuals). To be specific, the skeleton of Evocol is presented in Algorithm 1. The *stopping condition* is either to find a legal coloring or to reach a predefined time limit. In our experiments, most of the CPU time is spent by the local search operator. Depending on the graph, a time limit is equivalent to a limit on the number of local search iterations—which is a constant multiple of the number of crossovers (step 2.A.2 and 2.A.3 are always performed together).

Algorithm 1. Evocol: Evolutionary Hybrid Algorithm for Graph Coloring

```
Input: the search space Ω
Result: the best configuration ever found
1. Initialize (randomly) parent population Pop = (I_1, I_2, ..., I_|Pop|)
2. While a stopping condition is not met
   A. For i = 1 to p (p = number of offspring)
      Repeat
        1. (I_1, I_2, ..., I_n)=SelectParents(Pop, n) /* n ≥ 2 */
        2. O_i =Crossover(I_1, I_2, ... I_n)
        3. O_i =LocalSearch(O_i, maxIter)
      Until AcceptOffspring(Pop, O_i)
   B. Pop=UpdatePopulation(Pop, O_1, O_2, ..., O_p)
```

The performance of Evocol closely depends on several *independent* components, most notably: the crossover operator (function `Crossover`), the population management (functions `AcceptOffspring` and `UpdatePopulation`), and the local search algorithm. The `SelectParents` and `LocalSearch` procedures are quite classical and we only briefly describe them.

The parent selection simply consists in choosing n different individuals uniformly at random from the population. Such a selection favors diversity of the chosen parents in comparison with the roulette wheel or tournament selection that favors the selection of the fittest individuals.

The `LocalSearch` procedure is an improved version of a classical Tabu Search algorithm for graph coloring: Tabucol [12]. Basically, this algorithm iteratively moves from one coloring to another by modifying the color of a conflicting vertex until either a legal coloring is found, or a predefined number of iterations (i.e. $maxIter = 100000$) is reached. Each performed move (or color assignment) is marked Tabu for a number of iterations referred to as the Tabu tenure T_ℓ; in this manner, Tabucol cannot re-perform a move that was already performed during the last T_ℓ iterations.

Numerous versions of this algorithm can be found in the literature, and one of the most notable differences between them lies in the way they set the Tabu tenure. In our case, $T_\ell = \alpha * f(C) + random(A) + \left\lfloor \frac{M}{M_{\max}} \right\rfloor$, where α, A and $M_{\max}$ are predefined parameters, M is number of the last consecutive moves that kept the fitness function constant, $random(A)$ is a random integer in $[1..A]$. Concerning the parameters values, we use: $\alpha = 0.6$, $A = 10$ (as previously published in [7]), and $M_{\max} = 1000$. The last term is a new reactive component only introduced to increment T_ℓ after each series of $M_{\max}$ iterations with no fitness variation. This situation typically appears when the search process is completely blocked cycling on a plateau; an extended Tabu list can more easily trigger the search process diversification that is needed in this case.

3 New Multi-Parent Crossover

As already indicated in [4,7], effective graph coloring crossovers can be designed by considering a coloring as a partition of V (Definition 1). Here, we propose a Multi-Parent Crossover (MPX) for k-coloring that collects in the offspring the *best color classes* from several parents. To formally define the notion of "best class", each class in each parent receives a score based on two criteria: (i) the number of conflicts (generated only by the class vertices) and (ii) the size of the class. Note that, in comparison with other crossovers in the literature, MPX also takes into account the class conflict number.

The MPX operator (see Algorithm 2) actually searches (Steps 2.A and 2.B) for the largest class among those with the minimum class conflict number (i.e. minimum number of conflicting edges *in* the class). After assigning the class to the offspring (Step 2.C), it chooses the next best class and repeats. At each step, all class scores are calculated only after erasing the vertices that already received a color in the offspring (Step 2.A.1). It stops when k colors classes are assigned and a simple greedy procedure then fills any remaining unassigned vertex (Step 3).

The only risk of this crossover is to inherit most classes only from one parent, especially if there is a (very fit) parent whose classes "eclipse" the others. However, the similarity between the offspring and the parents is implicitly checked afterward by the `AcceptOffspring` procedure (see Section 4.2) that rejects the offspring if it is too similar to any existing individual.

The complexity of MPX is $O(k^2 \times n \times \overline{|I_i^j|}^2)$ where $\overline{|I_i^j|}$ is the average class size and n is the number of parents—the term $k^2 \times n$ is due to the three `For/Foreach` loops in step 2 and $\overline{|I_i^j|}^2$ is due to step 2.A.2. Since $\overline{|I_i^j|}$ is about $\frac{|V|}{k}$, this complexity is roughly equivalent to $O(|V|^2 \times n)$. In our practical case $n = 3$, and thus, the crossover takes much less time than the $maxIter = 100000 \geq |V|^2$ iterations of the local search procedure (in which, each iteration takes at least $O(|V|)$).

Notice that the authors of [5] report multi-parent crossover for the 3-coloring problem. Contrary to MPX, their crossover operates on an order-based

Algorithm 2. The multi-parent crossover MPX

Input: parents $I_1, I_2, \ldots, I_n$
Result: offspring O
1. O =empty, i.e. start with no vertex color assigned
2. For *currentColor*= 1 **To** k
  A. **Foreach** parent $I_i \in \{I_1, I_2, \ldots, I_n\}$
    Foreach color class I_i^j in I_i
      1. Remove from I_i^j all vertices already assigned in O
      2. $conflicts = |\{(v_1, v_2) \in I_i^j \times I_i^j : (v_1, v_2) \in E\}|$
      3. $classSize = |I_i^j|$
      4. $score[I_i^j] = conflicts \times |V| - classSize$
  B. **Set** $(i^*, j^*) = arg\min_{(i,j)} score[I_i^j]$
  C. **Foreach** $v \in I_{i^*}^{j^*}$
    $O[v]$ =*currentColor*
3. Foreach unassigned $v \in O$
  $O[v]$ =a color that generates the least number of conflicts

representation of colorings. Experimental results are reported on two small random graphs of 90 vertices.

4 Population Management

It is well known that the population diversity is a key element of an effective evolutionary algorithm [7, 16]. In fact, a low diversity constitutes a stopping condition for numerous practical algorithms—it usually indicates a premature convergence on poor solutions. By using a distance metric on the search space, Evocol *strictly* controls diversity with two mechanisms:

- It rejects a new offspring if it is too close to an existing individual of the population;
- It introduces a diversity criterion in the selection of the individuals to be eliminated (diversity-based replacement strategy).

4.1 Search Space Distance Metric

Let us first describe the distance metric on which the diversity control is based. We define the distance function between individuals I_A and I_B using the partition coloring representation (see Definition 1). As such, we view two colorings as two partitions of V and we apply the following set-theoretic partition distance (call it d): the minimum number of elements that need to be moved between classes of the first partition so that it becomes equal to the second partition. This distance was defined several times since the 60's and the currently used computation methodology was first described in the 80's (see [3], or, more recently [10]); it was also already used for graph coloring [7, 9].

The distance $d(I_A, I_B)$ is determined using the formula $d(I_A, I_B) = |V| - s(I_A, I_B)$, where s denotes the (complementary) similarity function: the maximum number of elements in I_A that do not need change their class in order to transform I_A into I_B. This similarity function reflects a structural similarity: the better the I_A classes can be mapped to the I_B classes, the higher the value of $s(I_A, I_B)$ becomes; in case of equality, this mapping is an isomorphism and $s(I_A, I_B)$ is $|V|$. Both the distance and the similarity take values between 0 and $|V|$ and this is why we usually report them in terms of percentages of $|V|$.

To compute these values, we define the $k \times k$ matrix S with elements $S_{ij} = |I_A^i \cap I_B^j|$; thus, s can be determined by solving a classical assignment problem: find a S assignment (i.e. a selection of S cells with no two cells on the same row or column) so that the sum of all selected cell values is maximized. This assignment problem is typically solved with the Hungarian algorithm of complexity $O(k^3)$ in the *worst* case. However, in our practical application, there are very few situations requiring this worst-case time complexity. We did not observe any significant slow-down caused by distance computations; the most time consuming procedure is still the local search.

4.2 Offspring Reject Mechanism

As we are committed to maintaining population diversity, we insert an offspring in the population only if its distance to each existing individual is greater than a predefined threshold; denote it by R. Consequently, if an offspring O is situated at a distance of less than R from an individual I, the `AcceptOffspring` procedure (see Algorithm 1) either (i) rejects O or, (ii) directly replaces I with O if $f(O) \leq f(I)$ (i.e. if O is better than I). However, in both cases, a new offspring is generated by starting with the parent selection—see the `Repeat-Until` loop in step 2.A of Algorithm 1.

The only delicate issue in the application of this simple mechanism is to determine a suitable R value. Let us denote by $S_R(I)$ the closed sphere of radius R centered at I, i.e. the set of individuals $I' \in \Omega$ such that $d(I, I') \leq R$. If I is a local minimum, an appropriate value of R should imply that all other local minima from $S_R(I)$ share important color classes with I, i.e. they bring no new information into the population (or they are structurally related to I). We have to determine the maximum value of R such that all local minima, that are structurally *unrelated* to I, are situated outside $S_R(I)$.

Since all individuals in the population are local minima obtained with Tabu Search, we determine R from an analysis of its exploration path. Consider this classical scenario: start from an initial local minima I_0, and let Tabu Search visit a sequence of neighboring colorings as usually; we denote by I_0, I_1 ,$I_2, \ldots I_N$ all visited individuals satisfying $f(I_i) \leq f(I_0)$ $(\forall i \in [1..N])$. After recording all these individuals up to $N = 40000$, we computed the distance for each pair (I_i, I_j) with $1 \leq i, j \leq N$ and we constructed a histogram to show the number of occurrences of each distance value.

This histogram directly showed that the distribution of the distance value is bimodal, with numerous occurrences of small values (around $5\%|V|$) and of some

much larger values. This provides evidence that the $I_i's$ are arranged in distant groups of close points (clusters); the large distances correspond to inter-cluster distances and the small ones to intra-cluster distances. If we denote a "cluster diameter" by C_d, we can say that C_d varies from $7\%|V|$ to $10\%|V|$ depending on the graph, such that: (i) there are numerous pairs (i, j) such that $d(I_i, I_j) < C_d$, (ii) there are very few (less than 1%) pairs (i, j) such that $C_d < d(I_i, I_j) < 2C_d$ and, (iii) there are numerous occurrences of some larger distance values.

To determine a good value of R, it is enough to note that any two local minima situated at a distance of more than $10\%|V|$ (approximately the highest possible C_d value) are not in the same cluster—because (ideally) they have some different essential color classes. We assume that this observation holds on all sequences of colorings visited by Tabu Search and we set the value of R to $10\%|V|$ for all subsequent runs.

4.3 Diversity-Based Replacement Strategy

The `UpdatePopulation` procedure determines which existing individual is eliminated for each offspring that needs to be inserted. While most previous algorithms take into account only the fitness values of the population (e.g. by replacing the least fit individual), we also take interest into the population diversity. To control diversity, this procedure encourages the elimination of individuals that are too close to some other individuals; in this manner, it gets rid of small distances in the population.

Generally speaking, the procedure (see Algorithm 3 bellow) selects two very close individuals that candidate for elimination and only the least fit of them is eliminated. The first candidate C_1 is chosen by a random function using some fitness-based guidelines (via the `AcceptCandidate` function). The second candidate C_2 is chosen by introducing the following *diversity criterion*: C_2 is the closest individual to C_1 respecting the same fitness-based guidelines as C_1.

The `AcceptCandidate` function makes a distinction between the first half of the population (the individuals with a fitness value lower than the *median*), the second half of the population and the best individuals. As such, this function always accepts a candidate C_i for elimination if C_i belongs to the second half, but it accepts C_i only with 50% probability if C_i belongs to the first half. Only the best individual is fully protected; it can never become a candidate for elimination—unless there are too many best individuals (more than half of the population) in which case *any* individual can be eliminated. As such, the role of the first half of the population is to permanently keep a sample of the best individuals ever discovered. The first half of the population stays quite stable in comparison with the second half that is changing very rapidly.

5 Experimental Results

The experimental studies are carried out on the most difficult instances from the well-known DIMACS Benchmark [13]: (i)$dsjcA.B$—classical random graphs

Algorithm 3. The replacement (elimination) function

```
Input: population Pop = (I_1, I_2, ..., I_|Pop|)
Result: the individual to be eliminated
1. Repeat
     C_1 = RandomIndividual(Pop)
   Until AcceptCandidate(C_1) (fitness-based acceptance)
2. minDist = maximum possible integer
3. Foreach I ∈ Pop − {C_1}
     If d(I, C_1) <minDist
        If AcceptCandidate(I)
          • minDist = d(I, C_1)
          • C_2 = I
4. If f(C_1) < f(C_2)
     Return C_2
   Else
     Return C_1
```

with unknown chromatic numbers (A denotes $|V|$ and B denotes the density), (ii)$le450.25c$ and $le450.25d$—the most difficult "Leighton graphs" with $|V| = 450$ and $\chi = 25$ (they have at least one clique of size χ), (iii)$flat300.28$ and $flat1000.76$—the most difficult "flat" graphs with χ denoted by the last number (generated by partitioning the vertex set in χ classes, and by distributing the edges only between vertices of different classes), (iv) $r1000.1$, $r1000.5$ and $dsjr500.5$—random geometric graphs, generated by picking points (vertices) uniformly at random in the square and by adding edges between each two vertices situated within a certain distance, (v) $C2000.5$—a very large graph (2.000 vertices and 1.000.000 edges).

We report in Table 1 the general results[1] obtained by Evocol with the following settings: $|Pop| = 15$ (population size), $n = 3$ (number of parents), $p = 3$ (number of offspring constructed each generation), $maxIter = 100000$ (the maximum number of iterations of the Tabu Search procedure), $R = 10\%|V|$ (the sphere radius, the minimum imposed distance between two individuals in the population). For each important value of k, this table reports the success rate over 10 independent runs (Column 3), the average number of generations required to solve each problem (Column 4), the average number of crossovers (Column 5) and the average CPU time in seconds (last column). The reported times are measured on a 2.8GHz Xeon processor using the C++ programming language compiled with the $-O3$ optimization option (gcc version 4.1.2 under Linux).

The total number of local search iterations is in close relation with the number of crossovers because the local search procedure (with $maxIter = 100000$) is applied once for each crossover. The algorithm performs at least $p = 3$ crossovers per generation, but it can perform many more (e.g. for the $dsjc500.1$ instance)

[1] The best colorings reported in this paper are publicly available at: www.info.univ-angers.fr/pub/porumbel/graphs/evocop/

Table 1. The results of Evocol with a CPU time limit of 5 hours. The algorithm finds most of the best known solutions with a success rate of more than 50% (see Column 3)—the minimal value of k for which a solution was ever reported in the literature (i.e. k^*) is given in the parentheses of Column 1.

Graph (best known k)	k	successes/runs	generations	crossovers	time[s]
$dsjc250.5$ ($K^* = 28$)	28	10/10	9	33	20
$dsjc500.1$ ($K^* = 12$)	12	10/10	96	1573	928
$dsjc500.5$ ($K^* = 48$)	48	10/10	258	827	1428
$dsjc500.9$ ($K^* = 126$)	126	10/10	222	985	1804
$dsjc1000.1$ ($K^* = 20$)	20	8/10	301	3350	4688
$dsjc1000.5$ ($K^* = 83$)	84	10/10	274	839	4729
	83	6/10	798	2722	13251
$dsjc1000.9$ ($K^* = 224$)	225	10/10	268	857	4328
	224	8/10	500	1702	8487
$le450.25c$ ($K^* = 25$)	26	10/10	1	3	2
	25	7/10	1102	8232	5690
$le450.25d$ ($K^* = 25$)	26	10/10	1	3	2
	25	5/10	650	4479	3152
$flat300.28.0$ ($K^* = 28$)	31	10/10	16	56	44
$flat1000.76.0$ ($K^* = 82$)	83	10/10	261	802	4539
	82	5/10	583	1940	9956
$r1000.1c$ ($K^* = 98$)	98	7/10	80	1939	4591
$r1000.5$ ($K^* = 234$)	248	10/10	326	1088	5368
	247	8/10	524	1836	8698
	246	7/10	448	1417	7497
	245	3/10	682	2242	11049
$dsjr500.5$ ($K^* = 122$)	125	10/10	265	1207	1764
	124	6/10	612	3113	4508
$C2000.5$ ($K^* = 153$)	152	5/5	380	1163	27262[a]
	151	4/5	433	1368	32520[a]

[a] Only for this very large graph, we used an exceptional time limit of 10 hours.

if many offspring are rejected by the `AcceptOffspring` procedure—see more discussions in the next section.

6 Discussion

In this section, we investigate the algorithm evolution, placing a special emphasis on the number of parents (in the recombination) and on the diversity control. Figure 1 compares the *running profile* of Evocol (i.e. the graph of the function $t \mapsto f_*(t)$, where t is the time and $f_*(t)$ is the best known fitness value at time t) for different values of the number of parents n. We first notice that the two-parent recombination ($n = 2$) always gives poor results in comparison with any value $n > 2$. This confirms that the multi-parent recombination has more potential; however, it seems more difficult to determine which is the exact optimum number of parents. We set $n = 3$ in this paper because this is the most stable choice: it always produces reasonable results on all graphs—the choice $n = 7$, even if it seems surprisingly competitive on some random instances, has great difficulties in solving the Leighton graphs.

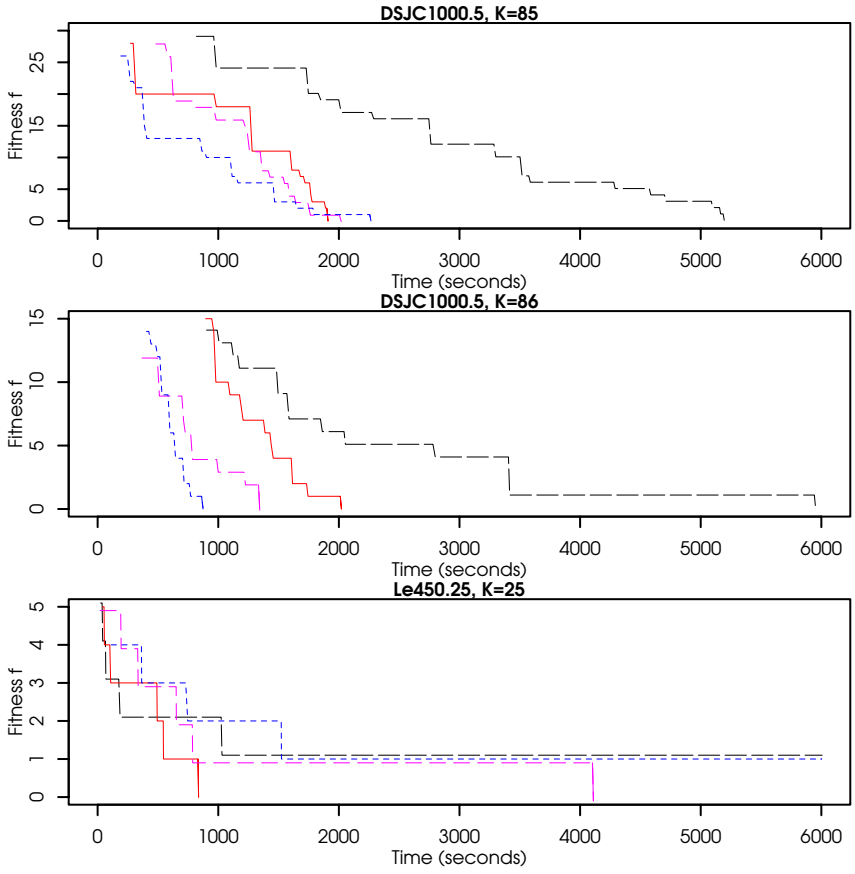

Fig. 1. The running profile (i.e. the evolution of the fitness of the best individual in the population) for several values of the number of parents: 3 parents (red, continuous line), 5 parents (magenta, with long dashes), 7 parents (blue, with normal dashes), 2 parents (black, with very long dashes)

The population management also plays a very important role in the evolution of the algorithm. In all situations from Figure 1 where the 7-parent crossover operator is not effective, we observed that the `AcceptOffspring` procedure rejects a very large proportion of the offspring. Table 1 shows important information on this issue: if we denote by g the number of generations and by c the number of crossovers, the number of rejected offspring is $c - 3g$. As such, the probability to reject an offspring can vary from 0 (e.g. for $G = flat1000.76$ and $k = 83$, we obtain $\frac{c-3g}{c} = \frac{802-3\times261}{802} \approx 0.02$) to 90% (e.g. for $G = r1000.1$ $\frac{c-3g}{c} = \frac{1939-3\times80}{1939} \approx 0.88$). In the cases where the offspring rejection rate is high, the population management accounts for the most important performance gain.

7 Conclusions

We described a new hybrid evolutionary algorithm (Evocol) that distinguishes itself from other population-based heuristics by introducing a strict population diversity control and by employing a multi-parent recombination operator (MPX). Compared with six state-of-the-art algorithms from the literature (see Table 2), the results of Evocol are very encouraging. Evocol finds most of the best-known colorings with at least 50% success rate (see also Table 1, column 3).

Table 2. Comparison of the best values of k for which a legal coloring is found by Evocol (Column 3) and by the best algorithms (Columns 4-9). Column 2 reports the chromatic number (? if unknown) and the best k for which a solution was ever reported.

Graph	χ, k^*	*Evocol*	VSS [11] 2008	PCol [1] 2008	ACol [8] 2008	MOR [15] 1993	GH [7] 1999	MMT [14] 2008
$dsjc250.5$	$?, 28$	*28*	–	–	28	28	28	28
$dsjc500.1$	$?, 12$	*12*	12	12	12	12	–	12
$dsjc500.5$	$?, 48$	*48*	48	49	48	49	48	48
$dsjc500.9$	$?, 126$	*126*	127	126	126	126	–	127
$dsjc1000.1$	$?, 20$	*20*	20	20	20	21	20	20
$dsjc1000.5$	$?, 83$	*83*	87	88	84	88	83	83
$dsjc1000.9$	$?, 224$	*224*	224	225	224	226	224	225
$le450.25c$	$25, 25$	*25*	26	25	26	25	26	25
$le450.25d$	$25, 25$	*25*	26	25	26	25	26	25
$flat300.28$	$28, 28$	*31*	28	28	31	31	31	31
$flat1000.76$	$76, 82$	*82*	86	87	84	89	83	82
$r1000.1c$	$?, 98$	*98*	–	98	–	98	–	98
$r1000.5$	$?, 234$	*245*	–	247	–	241	–	234
$dsjr500.5$	$?, 122$	*124*	125	125	125	123	–	122
$C2000.5$	$?, 153$[a]	*151*	–	–	–	165	–	–

[a] This graph was colored with $k = 153$ [6] by first removing several independent sets.

Indeed, for 11 out of the 15 difficult graphs, Evocol matches the previously best results: only for 3 DIMACS instances the results of Evocol are worse. For the largest graph $C2000.5$, it is remarkable that Evocol manages to find a 151-coloring (i.e. with 2 colors less than the best coloring known today) with a 4/5 success rate within 10 hours —for such a large instance, other algorithms might need several days. Note that for most unlisted DIMACS instances, all modern algorithms report the same k because these instances are not difficult; they can be easily colored by Evocol using the same number of colors k reported by most algorithms (like in the case $k = 12$ for $dsjc500.1$).

The general principles behind Evocol are quite simple and natural; moreover, its practical implementation only consists in relatively lightweight programming procedures. Nevertheless, it can quite quickly find the best known k-colorings and leaves plenty of room for further development.

Acknowledgments. This work is partially supported by the CPER project "Pôle Informatique Régional" (2000-2006) and the Régional Project MILES (2007-2009). We thank the referees for their useful suggestions and comments.

References

1. Blöchliger, I., Zufferey, N.: A graph coloring heuristic using partial solutions and a reactive tabu scheme. Computers and Operations Research 35(3), 960–975 (2008)
2. Costa, D., Hertz, A., Dubuis, C.: Embedding a sequential procedure within an evolutionary algorithm for coloring problems in graphs. Journal of Heuristics 1(1), 105–128 (1995)
3. Day, W.H.E.: The complexity of computing metric distances between partitions. Mathematical Social Sciences 1, 269–287 (1981)
4. Dorne, R., Hao, J.K.: A new genetic local search algorithm for graph coloring. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) PPSN 1998. LNCS, vol. 1498, pp. 745–754. Springer, Heidelberg (1998)
5. Eiben, A.E., Raué, P.E., Ruttkay, Z.: Genetic algorithms with multi-parent recombination. In: Davidor, Y., Männer, R., Schwefel, H.-P. (eds.) PPSN 1994. LNCS, vol. 866, pp. 78–87. Springer, Heidelberg (1994)
6. Fleurent, C., Ferland, J.A.: Genetic and hybrid algorithms for graph coloring. Annals of Operations Research 63(3), 437–461 (1996)
7. Galinier, P., Hao, J.K.: Hybrid Evolutionary Algorithms for Graph Coloring. Journal of Combinatorial Optimization 3(4), 379–397 (1999)
8. Galinier, P., Hertz, A., Zufferey, N.: An adaptive memory algorithm for the k-coloring problem. Discrete Applied Mathematics 156(2), 267–279 (2008)
9. Glass, C.A., Pruegel-Bennett, A.: A polynomially searchable exponential neighbourhood for graph colouring. Journal of the Operational Research Society 56(3), 324–330 (2005)
10. Gusfield, D.: Partition-distance: A problem and class of perfect graphs arising in clustering. Information Processing Letters 82(3), 159–164 (2002)
11. Hertz, A., Plumettaz, A., Zufferey, N.: Variable space search for graph coloring. Discrete Applied Mathematics 156(13), 2551–2560 (2008)
12. Hertz, A., Werra, D.: Using tabu search techniques for graph coloring. Computing 39(4), 345–351 (1987)
13. Johnson, D.S., Trick, M.: Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge. DIMACS series in Discrete Mathematics and Theoretical Computer Science, vol. 26. American Mathematical Society, Providence (1996)
14. Malaguti, E., Monaci, M., Toth, P.: A Metaheuristic Approach for the Vertex Coloring Problem. INFORMS Journal on Computing 20(2), 302 (2008)
15. Morgenstern, C.: Distributed coloration neighborhood search. In: [13], pp. 335–358
16. Sörensen, K., Sevaux, M.: MA—PM: Memetic algorithms with population management. Computers and Operations Research 33(5), 1214–1225 (2006)