A Memetic Algorithm with Adaptive Operator Selection for Graph Coloring

Cyril Grelier, Olivier Goudet, and Jin-Kao Hao(✉)

LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France
{cyril.grelier,olivier.goudet,jin-kao.hao}@univ-angers.fr

Abstract. We present a memetic algorithm with adaptive operator selection for k-coloring and weighted vertex coloring. Our method uses online selection to adaptively determine the couple of crossover and local search operators to apply during the search to improve the efficiency of the algorithm. This leads to better results than without the operator selection and allows us to find a new coloring with 404 colors for C2000.9, one of the largest and densest instances of the classical DIMACS coloring benchmarks. The proposed method also finds three new best solutions for the weighted vertex coloring problem. We investigate the impacts of the different algorithmic variants on both problems.

Keywords: Graph Coloring · Memetic Algorithm · Hyperheuristics

1 Introduction

Graph coloring problems find applications across various domains, such as matrix decomposition [25], metropolitan area network design [14], and task scheduling in distributed computing [17]. Given a graph $G = (V, E)$, defined by its vertex set V and edge set E, a *legal* coloring S of the graph G partitions the vertex set V into k non-empty and disjoint color groups $\{V_1, \ldots, V_k\}$ such that the coloring constraint is satisfied, i.e., for each V_i, if $x \in V_i$ and $y \in V_i$, then $\{x, y\} \notin E$. In other words, the coloring constraint states that two adjacent vertices cannot go to the same color group (they cannot receive the same color). A coloring failing to meet the coloring constraint is an illegal coloring. Typically, graph coloring problems entail finding a legal coloring of the graph G while taking into account additional decision criteria and constraints. Specifically, the k-coloring problem (k-col) is a decision problem, where given a number of colors k, the goal is to find a legal coloring of the graph using these k colors. The graph coloring problem (GCP) is to determine the smallest number of colors (chromatic number of the graph) needed to color a graph. In the weighted vertex coloring problem (WVCP), an additional weight function $w : V \rightarrow \mathbb{R}^+$ is defined to assign a strictly positive weight $w(v)$ to each vertex $v \in V$. The objective of the WVCP is then to find a legal coloring $S = \{V_1, \ldots, V_k\}$ with a minimal score $f(S) = \sum_{i=1}^{k} \max_{v \in V_i} w(v)$, where the number of used colors k is not specified.

A. I. Esparcia Alcazar (Ed.): EvoCOP 2024, LNCS 14632, pp. 1–16, 2024.
https://doi.org/10.1007/978-3-031-57712-3_5

This paper aims to develop a learning-based framework for solving both the k-col decision problem and the WVCP minimization problem, which have been addressed by various methods in the literature. Both problems are known to be NP-hard in the general case [8], posing computational challenges in practice. For example, some random graphs with 250 vertices cannot be solved optimally by current exact algorithms (see [19] for k-col and [11] for WVCP). For this reason, a number of heuristics have been developed over the last thirty years to obtain approximate solutions to large graph coloring problems [6,20].

For the k-col, two particularly interesting local search heuristics are TabuCol [15] and PartialCol [2], which were proposed many years ago. For the WVCP, dedicated and effective local search procedures are much more recent, including AFISA [28], RedLS [29], ILS-TS [22] and TabuWeight [12]. None of these methods really dominates the others for all reference instances of the k-col and the WVCP [2,12,22]. It would be interesting to choose an appropriate local search to solve each type of instance for each problem.

However, local search methods may fail to produce high-quality solutions due to their limited ability to diversify their search. To overcome this difficulty, hybrid algorithms using the framework of memetic algorithms have been proposed, which benefit from local search for intensification and offer diversification possibilities with a population of high-quality solutions recombined with crossover operators. Hybrid algorithms have mainly been used to date to solve the k-coloring problem. The HEA (*Hybrid Evolutionary Algorithm*) algorithm [7] introduced the powerful GPX crossover (*Greedy Partition Crossover*) operator and used a local tabu search inspired by TabuCol. Evo-Div [23] and MACOL [18] both used crossover strategies with multiple parents and distance management between solutions. More recently, the HEAD algorithm (*HEA in Duet*) [21] proposes the use of only two individuals in the population and a reintegration system for high-quality individuals (elites) found earlier in the search. HEAD also uses the GPX crossover and an improved TabuCol algorithm. This algorithm HEAD is currently one of the most efficient solvers for the k-col. For the WVCP, to our knowledge, there is only one memetic algorithm in the literature, DLMCOL [10], which uses a large population (more than 20,000) and parallel GPU-based local searches, combined with a neural network-guided crossover selection.

It was observed by the authors of HEAD [21], that the quality (number of conflicts) of an offspring solution for the k-col, generated with the GPX crossover from two parents, is highly correlated with the partition distance [24] between these two parents, which is the minimum number of vertices that must change color to transform one solution to another. Based on this insight, the authors suggested employing modified versions of the GPX crossover, such as conservative asymmetric crossovers, which can lead to better results for certain types of instances. In the context of a memetic algorithm, it is therefore important to choose not only the right local search, but also the right crossover operator to perform an efficient search.

To this end, in this work, we investigate the use of online hyperheuristics, suggested in the literature to dynamically select adapted low-level heuristic com-

ponents during the search process for solving a specific problem instance. We refer the reader to [3,4] for an overview of existing hyperheuristics used to solve various combinatorial optimization problems. More specifically, hyperheuristics have been used to solve partitioning problems, with applications to planning and graph coloring [5,13,26,27]. However, to our knowledge, no hyperheuristic-based memetic algorithm for graph coloring has yet been proposed in the literature. This work fills this gap by:

- investigating new memetic algorithms for the k-col and the WVCP using the HEAD framework [21] with diverse local search procedures and GPX variants.
- examining the ability of adaptive operator selectors to jointly choose crossovers and local search procedures during the search for specific instances.

In the rest of the paper, we first present our new general framework, AHEAD (for Adaptive HEAD), with different strategies for selecting local search procedures and crossovers (Sect. 2). Next, we show the results of the different variants of AHEAD in comparison with state-of-the-art algorithms (Sect. 3).

2 Adaptive Memetic Algorithm

In this section, we present the general framework of the adaptive memetic algorithm developed for the k-col and the WVCP, as well as the selection operators.

For the k-col, with a graph $G = (V, E)$ and k colors, the search explores the space Ω_k of legal and illegal colorings where all vertices are colored, but allowing color conflicts between adjacent vertices:

$$\Omega_k = \{\{V_1, \ldots, V_k\} : (\cup_{i=1}^{k} V_i = V) \wedge (V_i \cap V_j = \emptyset, i \neq j, 1 \leq i, j \leq k)\}. \quad (1)$$

In this case, the fitness f (to be minimized) corresponds to the number of conflicts in the solution. A solution with the fitness of 0 is a legal coloring.

For the WVCP, the algorithm works in the space of legal solutions Ω_l, where all vertices are colored with no limit on the number of colors, but no adjacent vertices are allowed to share the same color:

$$\Omega_l = \{\{V_1, \ldots, V_k\} : (\cup_{i=1}^{k} V_i = V) \wedge (V_i \cap V_j = \emptyset, i \neq j, 1 \leq i, j \leq k) \\ \wedge (\forall v_1, v_2 \in V_i, (v_1, v_2) \notin E, 1 \leq i \leq k)\} \quad (2)$$

and we search in this search space a solution $S = \{V_1, \ldots, V_k\}$ whose score $f(S) = \sum_{i=1}^{k} \max_{v \in V_i} w(v)$ is minimum.

2.1 Main Scheme

The architecture of the AHEAD framework, illustrated in Fig. 1, extends the state-of-the-art HEAD framework [21] by introducing an operator selector. In particular, its simplicity with only two individuals in the population facilitates

the parent matching and population updating phases compared to other memetic algorithms in the literature [7,10,18,23].

AHEAD takes as input a graph $G = (V, E)$, and an integer value k for the k-col or a weight function w for the WVCP, a set of local search operators $\mathcal{O}^l$, a set of crossover operators $\mathcal{O}^x$, and a high-level selection strategy π_θ characterized by a parameter vector θ. The π_θ strategy chooses two pairs of operators $<o^l, o^x>$ with $o^l \in \mathcal{O}^l$ and $o^x \in \mathcal{O}^x$ to apply in the current generation to create two new individuals, replacing their parents in the population for the next generation.

The population is initialized with two random colorings, S_1 and S_2. Two elite solutions, E_1 and E_2, are also created, which are updated by the best solution found during the search and are used to reintroduce diversity into the population under specific conditions. Then, each generation of the algorithm performs the next six steps until the stopping condition is met.

1. A selection phase to select two pairs of operators <crossover, local search> to be applied during this generation: $<o_1^x, o_1^l>$ and $<o_2^x, o_2^l>$. This selection is made using the function π_θ, which takes as input the two individuals in the population, with S_1 as the first input for the selection of $<o_1^x, o_1^l>$, then S_2 as the first input for the selection of $<o_2^x, o_2^l>$ (Sect. 2.2).
2. A crossover phase to create two offspring individuals C_1 and C_2 with $C_1 = o_1^x(S_1, S_2)$ and $C_2 = o_2^x(S_2, S_1)$ (see Sect. 2.3).
3. An intensification phase to obtain improved offspring solutions with local search, $C_1' = o_1^l(C_1)$ and $C_2' = o_2^l(C_2)$ (Sect. 2.4).
4. An insertion phase to replace S_1 and S_2 by C_1' and C_2', regardless of the fitness of the offspring solutions compared to the parents.
5. An update phase to adjust the π_θ selection policy from the examples collected over the last few generations (Sect. 2.5).
6. As in the original HEAD algorithm [21], each generation ends with a step of storing the best individual (*elite*) from the cycle (10 generations). The elite individual is reintroduced two cycles later (Sect. 2.6).

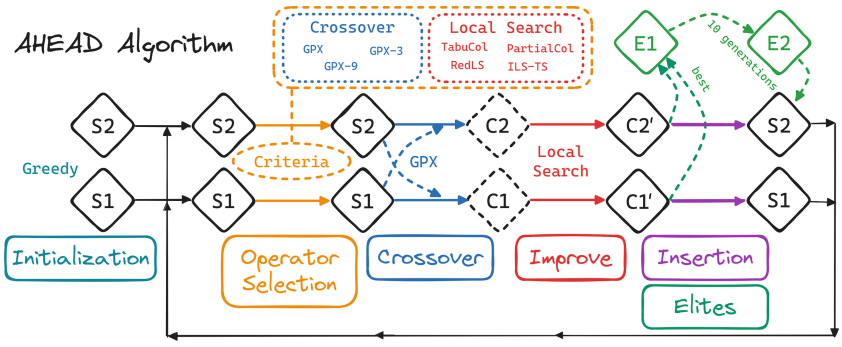

Fig. 1. General architecture of the Adaptive HEAD (AHEAD) framework.

2.2 Automatic Operator Selection

To select the crossover and local search operators, a high-level π_θ strategy automatically selects a pair of operators $<o^x, o^l> \in \mathcal{O}^x \times \mathcal{O}^l$. Therefore, there are $|\mathcal{O}^x| \times |\mathcal{O}^l|$ different <crossover, local search> pairs possible considered as independent meta-operators. The set of crossover operators $\mathcal{O}^x$ is presented in Sect. 2.3 and the set of local search operators $\mathcal{O}^l$ in Sect. 2.4.

In the general case, the π_θ function takes a pair of parents as input. For example, for (S_1, S_2), π_θ chooses a pair of operators $<o^x, o^l>$, with a crossover operator o^x to be applied with S_1 as the first parent and S_2 as the second parent. It will produce a child C_1, which will be improved by the local search procedure o^l to obtain a new individual C_1'.

In this work, we examine the effects of six operator selection policies π_θ with varying levels of complexity, including four fitness-based criteria, a neural network, and a random selector. Note that, except for the Deleter criteria, they are also used in combination with a Monte Carlo tree search in [13].

- *Random* performs a uniform selection among all operators.
- *Deleter* deletes the operator with the worst average result every 5 generations, until only one remains.
- *Roulette* (or Adaptive roulette wheel) [9] selects randomly the operator with a bias induced by the reward r (see Sect. 2.5) obtained by each operator. The better the reward is, the higher its associated probability of being picked is, using the same parameters as in [13].
- *UCB* (Upper Confidence Bound, One-armed bandit strategy) [1] selects the operator depending on the rewards obtained in previous generations and the number of times it has been picked using the UCB formula managing the exploitation-exploration trade-off.
- *Pursuit* [9] randomly selects the operator with a proportional bias in favor of the best performing operator. This strategy is among the most elitist, as it gives more chances to the best strategy applied in previous iterations.
- *NN* (Neural Network) uses the recommendations of a *deep set* neural network architecture [10,13,31]. This neural network g_θ takes as input a coloring S as a set of k binary vectors $\mathbf{v_j}$ of size n, $S = \{\mathbf{v_1}, \ldots, \mathbf{v_k}\}$, where each $\mathbf{v_j}$ indicates the vertices belonging to the color group j. From such an entry the neural network outputs a vector with $|\mathcal{O}^l|$ values in $\mathbb{R}$ corresponding to the expected reward of each local search. In order to select a pair of operators $<o^x, o^l> \in |\mathcal{O}^x| \times |\mathcal{O}^l|$, from a pair of parents (S_1, S_2), the neural network selector π_θ works as follows:
  1. Each crossover operator $o^x \in \mathcal{O}^x$ creates an individual $C^x = o^x(S_1, S_2)$.
  2. Each raw solution C^x is passed as input to the neural network g_θ to obtain a vector of $|\mathcal{O}^l|$ values in $\mathbb{R}$ corresponding to the estimated score that can be obtained after applying each local search $o^l \in \mathcal{O}$ to C^x.
  3. The pair $<o^x, o^l>$ corresponding to the highest output value of the neural network (for the $|\mathcal{O}^x|$ evaluations given by g_θ) is selected. 10% of the time, the selection is random to encourage diversity.

2.3 Application of Crossover Operators

In this step, the parent solutions S_1 and S_2 are combined to create two offspring solutions $C_1 = o_1^x(S_1, S_2)$ and $C_2 = o_2^x(S_2, S_1)$ from the crossover operators o_1^x and o_2^x chosen during the selection phase detailed in the previous section.

For both problems, we use the popular GPX crossover [7] used in the original HEAD algorithm [21]. It consists of alternately taking the largest color group from each parent and transmitting it to the offspring solution. Note that GPX is asymmetrical, applying it with the pair (S_1, S_2) or the pair (S_2, S_1) does not produce the same offspring solution. Three variants of this crossover can be selected in AHEAD: GPX, GPX-3, and GPX-9 taking respectively 1, 3, and 9 groups of color in the first parent for 1 color in the second parent. Therefore the last two are more conservative than the original GPX as more groups of the first parent are transmitted to the offspring. This generally results in offspring with lower (better) fitness, but less different from its parents. The two new solutions C_1 and C_2 generated during the crossover phase will be improved in the local search phase detailed in the next subsection.

2.4 Local Searches

In this step, the new individuals C_1 and C_2 are improved by the selected local search operators o_1^l and o_2^l. These two independent local searches are run in parallel on two CPUs with a time limit of T_{LS} seconds. The best solutions C_1' and C_2' found by o_1^l and o_2^l with this time budget are returned. For the k-col, two state-of-the-art local search operators can be chosen: the efficient implementation of TabuCol [15] proposed in [21] and PartialCol [2]. For the WVCP, the two best performing algorithms RedLS [29] and ILS-TS [22] can be chosen.

2.5 Operator Selection Strategy Update

At each generation, new learning examples are collected from the results of pairs of operators to update the selection strategy for future generations.

Learning Examples Memory. Reward scores $r_1 = -f(C_1')$ and $r_2 = -f(C_2')$ are associated with the choice of operator pairs $<o_1^x, o_1^l>$ and $<o_2^x, o_2^l>$, with f the fitness function of the k-col or the WVCP. These rewards are negative, as the fitness f is to be minimized in both problems. Then, two learning examples (o_1^x, o_1^l, C_1, r_1) and (o_2^x, o_2^l, C_2, r_2) are stored in a database D, specific to the current execution. D is a queue of the N last examples obtained during previous generations (N is set to 50 empirically). This limited queue size enables us to better adapt to potential variations in operator results, in the event that certain operators are better at the beginning of the search than at the end.

Online Learning of the Selection Criteria. Every generation, the π_θ policy is trained on the database D and all its θ parameters are updated. For the NN policy, the training phase occurs every $nb = 20$ generations. During this training

phase, each training example (o^x, o^l, C, r) from the database D is converted into a supervised learning example (X, y), with X an input matrix of size $k \times |V|$ corresponding to the set of k vectors $C = \{\mathbf{v_1}, \ldots, \mathbf{v_k}\}$, and y is a real vector of size $|\mathcal{O}^l|$ (number of local search operators), so that y is initialized with $g_\theta(C)$, the output vector of the neural network taking as input C, then its value $y[o^l]$ for the chosen operator o^l is replaced by the expected reward r: $y[o^l] = r$. Once this conversion of the training examples is done, g_θ is trained to minimize the mean square error computed over the $|\mathcal{O}^l|$ outputs (supervised learning) on this training dataset for 10 epochs with the Adam optimizer [16].

2.6 Insertion and Elite Solutions

Like HEAD [21], offspring solutions systematically replace the parents and elite solutions are used. Elites, which are high-quality solutions from previous generations, are added to the population after 10 generations. Thus, every 10 generations, the best individual encountered from the previous 11 to 20 generations is reintegrated into the population.

Furthermore, at each generation, the set-theoretic partition distance [24] between the solutions S_1 and S_2 is evaluated. This distance is defined as the least number of one-move operator changes for transforming S_1 to S_2. If the distance is 0, meaning that the two individuals are the same solution, the individuals of the population are randomly reinitialized. Note that when this happens, we do not reset the θ weights of the learning strategy in order to benefit from what the operator selector has learned since the start of the search.

3 Computational Experiments

This section is dedicated to a computational assessment of the proposed AHEAD algorithm for solving the k-coloring and the weighted vertex coloring problems, by making comparisons with state-of-the-art methods. We also discuss the impact of the different operator selection strategies on the results.

3.1 Experimental Settings

In these experiments, we consider 31 instances for the k-col and 48 instances for the WVCP, among the most challenging DIMACS benchmark instances and widely used in the experiments of many recent papers [21,22,28]. 20 independent runs per method and per instance are carried out for one hour with two CPU for the HEAD and AHEAD methods (four hours for the WVCP) and two hours with one CPU for the local search algorithms for the k-col (eight hours for the WVCP). For each k-col instance, and for each independent run, the smallest value of k for which the method is able to find a legal solution is reported.

The time spent, in seconds, in the local search during each generation in HEAD and AHEAD is $0.001 * |V|$ for the k-col and $0.04 * |V|$ for the WVCP, with $|V|$ being the number of vertices in the graph. These values were chosen following

tests on a range of values for each problem, which will not be presented here. To erase the impact of the training time of the neural network, for the versions with AHEAD, the methods perform as many generations in the memetic algorithm as the HEAD versions.

The experiments are run on a computer equipped with an Intel Xeon ES 2630, 2.66 GHz processor. All algorithms are coded in C++, compiled and optimized with the g++ 12.1 compiler. For the neural network implementation, the Pytorch 1.13 C++ library was used. The source code and complete tables of detailed results are available at https://github.com/Cyril-Grelier/gcp_ahead and https://github.com/Cyril-Grelier/wvcp_ahead.

3.2 Experimental Results for the *k*-Coloring Problem

In this section, we first analyze the general results obtained for the k-coloring problem. The different methods tested can be regrouped into three categories:

- The standalone local searches PartialCol (PC) [2] and TabuCol (TC) [15] (with optimizations from [21]).
- The memetic algorithm HEAD [21] combined with local search operators PartialCol (version HEAD + PC) and TabuCol (version HEAD + TC).
- The different proposed AHEAD versions with the 6 different operator selection strategies presented in Sect. 2.2. These versions are called AHEAD + the name of the selection strategy.

We first present a general comparison between all these methods, followed by detailed results on the different benchmark instances.

General Comparisons. Table 1 displays general performance comparisons between each pair of methods on the benchmark instances considered in this work for the k-col. Whenever the mean score of a method in the row for an instance is better than the mean score of a method in the column, and this difference is significant (non-parametric Wilcoxon signed rank test with p-value ≤ 0.001) the method in the row obtains one point. If the method in the row is better on more instances than the method in the column than the opposite, then the number of instances is in bold. As an example, we observe in Table 1 that TabuCol is significantly better on 14 instances when compared with PartialCol, while PartialCol is only better on 2 instances when compared with TabuCol. The last three columns of Table 1 gives the number of times a Best Known Score (BKS) from the literature is found by the method and the number of times the method reaches the best score and the best mean among the presented methods.

We observe in Table 1 that using the memetic framework HEAD with PartialCol or TabuCol (version HEAD + TC, and HEAD + PC) improves the results over the methods using the corresponding local search alone.

Overall, TabuCol is more effective than PartialCol, that is why HEAD+TC keeps good results against some versions of AHEAD with less elitist operator selection such as Random, Roulette or UCB. However, the other versions of

Table 1. Comparison of each method for the k-col, the value is the number of instances where the method in the row is significantly better than the method in the column. The last three columns are a summary of the number of best scores.

/31 Instances	PartialCol	TabuCol	HEAD+PC	HEAD+TC	AHEAD+Random	AHEAD+Roulette	AHEAD+Deleter	AHEAD+UCB	AHEAD+Pursuit	AHEAD+NN	# BKS	# Best Score	# Best Mean
PartialCol	-	2	3	2	1	2	1	2	2	2	5	8	11
TabuCol	**14**	-	**11**	2	2	1	0	2	0	1	8	14	7
HEAD+PC	**8**	6	-	1	0	0	1	0	0	0	6	10	7
HEAD+TC	**18**	**12**	**20**	-	**4**	**2**	1	**2**	2	2	7	17	15
AHEAD+Random	**17**	**11**	**19**	1	-	0	1	1	0	0	9	17	9
AHEAD+Roulette	**17**	**11**	**19**	1	0	-	0	0	0	0	11	19	12
AHEAD+Deleter	**19**	**15**	**20**	**5**	**8**	**3**	-	**5**	**1**	**1**	**13**	**24**	**20**
AHEAD+UCB	**19**	**11**	**20**	1	1	0	0	-	0	0	10	18	10
AHEAD+Pursuit	**19**	**13**	**20**	**3**	**5**	**2**	0	**1**	-	0	11	20	14
AHEAD+NN	**19**	**12**	**20**	2	**4**	0	0	0	0	-	12	23	16

AHEAD, using Deleter, Pursuit and the neural network (NN), obtained overall better results than the memetic algorithm HEAD+TC without operator selection. It highlights the interest of dynamically choosing the best operator to apply for each given instance.

The Random selection policy is less effective in comparison with the other operator selection strategies, especially against Deleter, Pursuit and NN, which have a stronger bias on selecting the best operators. Surprisingly, the simplest but most elitist selection strategy, Deleter, achieves the best results, indicating that for this problem, once the best operator has been identified for each specific instance, there is generally no need to change it for the rest of the search.

Detailed Results. Table 2 shows, for each instance, the Best Known Score (BKS) in the literature[1] with a star if it is optimal, then, for each method, the best score, the mean score and the average time to reach the best scores over the 20 executions. Bold values indicate the best scores among the studied methods. The average score is not shown if equal to the best score. Due to space limitations, only a selection of methods is shown in the various tables, and not all instances studied are shown. Complete tables are available on the github repository.

First, those results confirm that the TabuCol local search is more often better than PartialCol, but that the latter can give better results for some

[1] Achieving these BKS for k-col, especially for the largest instances, is a very difficult task. Some have only been found by few algorithms under particular conditions (hyperparameter tuning, extended execution times of several days to a month).

Table 2. Results of the main methods for the GCP.

instance	BKS	PartialCol			TabuCol			HEAD+TC			AHEAD+Random			AHEAD+Deleter		
		best	mean	time	best	mean	time	best	mean	time	best	mean	time	best	mean	time
C2000.5	145	164	165.2	5313	162	162.8	4628	**148**	**149.2**	3330	150	150.7	3101	149	150.7	3152
C2000.9	408	420	420.8	5171	411	412.5	4786	405	406.4	2328	405	407.7	2956	**404**	**405.6**	2988
C4000.5	259	304	305.6	6690	303	304.2	5567	**278**	**279.6**	3580	280	281.6	3651	279	280.8	3404
DSJC500.1	12	**12**		128	**12**		75	**12**		86	**12**		80	**12**		56
DSJC500.5	47	50	50.1	2227	49		460	**48**		819	**48**		1258	**48**		850
DSJC500.9	126	128		975	**126**	126.3	2988	**126**		1027	**126**	126.1	1379	**126**		632
DSJC1000.1	20	21		1	21		0	21		0	21		1	**20**	**20.9**	2391
DSJC1000.5	82	90	90.5	3516	88		1760	**83**	**83.3**	2290	**83**	83.5	2372	**83**	83.5	2511
DSJC1000.9	222	227	228.4	3630	224	224.9	3345	**223**	224	1616	**223**	224.2	2734	**223**	**223.8**	1589
DSJR500.5	122*	125	126.2	1666	124	127	1155	**123**	124	1766	**123**	124.2	2245	**123**	**123.8**	2289
flat300_28_0	28*	**28**		896	**28**	29.5	3220	30	30.8	1916	**28**	28.5	702	**28**	30.4	5
flat1000_50_0	50*	**50**		44	**50**		69	**50**		28	**50**		8	**50**		8
flat1000_60_0	60*	**60**		213	**60**		233	**60**		54	**60**		28	**60**		29
flat1000_76_0	76*	89	89.1	2845	86	87	3096	**82**	**82.3**	1905	**82**	82.8	2775	**82**	82.8	1969
latin_square_10	97	107	110.2	4875	100	100.8	4377	102	103.7	93	103	103.8	1996	**99**	**100.7**	1729
le450_25c	25*	27		69	26		0	26		0	**25**	25.9	1407	**25**	**25.3**	1022
le450_25d	25*	27		50	26		0	26		0	26		0	**25**	**25.3**	1537
queen11_11	11*	**11**	**11.9**	1303	12		0	12		0	12		0	12		0
queen12_12	12*	**13**		4	**13**		0	**13**		0	**13**		0	**13**		1
queen13_13	13*	**14**		20	**14**		0	**14**		1	**14**		1	**14**		1
queen14_14	14*	**15**		585	**15**		20	**15**		9	**15**		18	**15**		16
r250.5	65*	67		134	66	67.2	462	**65**	66	3378	**65**	66	1638	66		549
r1000.1c	98	141	149.1	61	134	155.2	77	**100**	101.6	264	**100**	101.6	1674	**100**	101.6	1621
r1000.5	234	247	248.1	5638	**244**	245.6	3622	246	247.6	1479	246	247.4	2134	245	**245.5**	2009
wap01a	41*	42		1088	42	43	2160	42		137	42		143	**41**	**42**	1958
wap02a	40*	41	41.7	4275	**40**	41.1	6499	41		15	41		15	**40**	**40.8**	1634
wap03a	43	44		91	44	45.9	4342	45		261	45		87	**43**	44.3	2387
wap04a	41	43		61	**42**	43.1	4869	43		880	43		1186	43		293
wap06a	40*	41		98	**40**	41.3	4248	**40**		909	**40**	40.8	1549	**40**		246
wap07a	41	44		41	**41**	42.3	5046	42	42.1	1771	42	43	2526	42	42.1	494
wap08a	40*	43	43.2	2750	**41**	**41.5**	2967	42		48	42		365	**41**	41.9	2146
#BKS		5/31			8/31			7/31			9/31			13/31		
#Best		8/31			14/31			17/31			17/31			24/31		
#Best Avg		11/31			7/31			15/31			9/31			20/31		

instances such as queen11_11 and can be faster for solving other instances such as flat300_28_0 or wap03a. It shows to some extent that these two local searches can be complementary.

Second, we obviously confirm that when TabuCol is integrated within the HEAD memetic framework and combined with the GPX crossover (version HEAD +TC), it can generally improve the results significantly, but it does not improve the results for all instances. For example, for the instances flat300_28_0, r1000.5, and some wap, it is actually better to use the local search alone for better intensification. Using crossovers for these instances can actually disrupt the search too early, preventing the local search from significantly improving its results. Note that the results of HEAD in [21] can differ from our results with HEAD+TabuCol using exactly the same operators, because we did not perform a fine-tuning of the number of iterations spent in local search for each given instance, unlike it was done in the original article.

Third, the algorithm AHEAD + Random can find the best results for more instances than the HEAD+TabuCol version whose choice of operators does not change during the search. For example, AHEAD + Random finds the optimal coloring with 28 colors for the instance flat300_28_0. This is due to the fact that TabuCol is not able to reach the chromatic number of the graph in a short amount time compared to PartialCol which can reach it systematically more than three times faster. Therefore, AHEAD + Random benefits from having a chance at each generation to select the right local search for each given instance.

Fourth, from Table 2, we observe that using an elitist strategy in local search and crossover selection can significantly improve the results compared to the random selection strategy. In particular, as shown in this table, the version AHEAD + Deleter can find a new best coloring with $k = 404$ for the instance C2000.9 that has never been reported in the literature. This new best score is also found by the versions AHEAD + Pursuit and AHEAD + NN. In general, these versions of AHEAD with elitist operator selection strategies obtain the best results for a wide variety of instances of different types.

Figure 2 shows the average cumulative selections for each pair of operators, performed by the different selection criteria for the DSJC500.1 and queen12_12 instances. In the plots, TabuCol and PartialCol selections are indicated by red and blue lines, respectively, and a higher contrast indicates a more conservative crossover. When we look at the frequencies of the local search operators selected by AHEAD, we see that TabuCol is selected most often in comparison with PartialCol, which is no surprise, as TabuCol is already better on its own for a greater number of instances. However, when it comes to crossovers, we observe a balanced choice between the three GPX variants, with a bias toward the more conservative crossover GPX-9 for geometric graphs (e.g., DSJR500.5) and sparse graphs (e.g., wap instances), for which local optima are very distant in the search space, while the GPX crossover is more often preferred for random and dense graphs (e.g., DSJC1000.9), for which there is often larger backbones of solutions shared by the high-quality solutions (as shown in [10]).

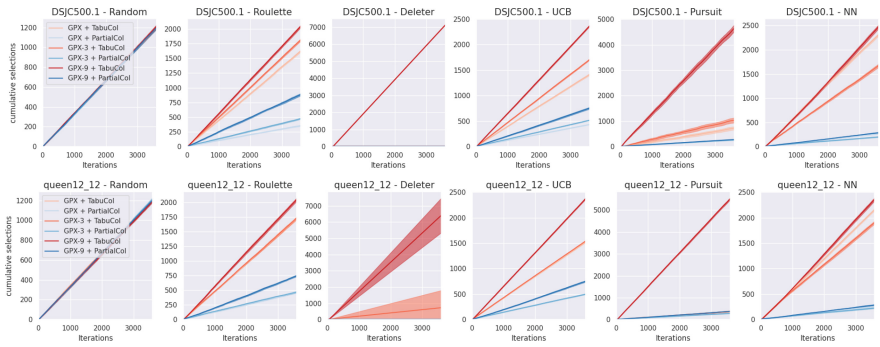

Fig. 2. Average cumulative selections, along with error bars, for each pair of operators based on different criteria on the DSJC500.1 and queen12_12 instances.

3.3 Experimental Results for WVCP

Now, we analyze the general results obtained for the WVCP. We first present a general comparison between the methods, followed by detailed results on the different benchmark instances. The studied methods are the following:

- The local searches, RedLS [29] and ILS-TS [22] (8h runs on 1 CPU).
- The memetic algorithm HEAD [21] with RedLS or ILS-TS (HEAD + RedLS /ILS-TS) and the crossover GPX (4h runs on 2 CPUs).
- The different proposed AHEAD versions with the 6 different operator selection strategies presented in Sect. 2.2 with the two local searches (RedLS and ILS-TS) and the three variants of GPX crossovers (4h runs on 2 CPUs).

General Comparisons. As seen in Table 3, using HEAD with RedLS (HEAD + RedLS) improves the results for 26 instances but the standalone local search RedLS stays better in 9 instances. On the other hand, for ILS-TS, using the HEAD framework is better only for 6 instances, while ILS-TS remains better for 19 instances. We observe that the use of crossovers in combination with the ILS-TS local search procedure does not improve the results of ILS-TS. This can be explained by the fact that ILS-TS is already a method incorporating a strong perturbation strategy for search diversification, making the use of crossovers somewhat superfluous. Regarding the results of AHEAD, we confirm what we have observed for the k-col, even if it's less pronounced. The AHEAD versions are more often better than the other methods, and the most elitist operator selection strategies (Deleter and Pursuit) obtain the best results.

Table 3. Comparison of each method for the WVCP, the value is the number of instances where the method in the row is significantly better than the method in the column.

/48 Instances	RedLS	ILS-TS	HEAD+RedLS	HEAD+ILS-TS	AHEAD+Random	AHEAD+Roulette	AHEAD+Deleter	AHEAD+UCB	AHEAD+Pursuit	AHEAD+NN	# BKS	# Best Score	# Best Mean
RedLS	-	10	9	14	9	9	9	9	9	9	15	24	11
ILS-TS	**27**	-	8	**19**	3	6	5	4	1	3	23	25	21
HEAD+RedLS	**26**	**15**	-	**25**	1	1	0	0	1	0	19	19	11
HEAD+ILS-TS	**20**	6	5	-	0	0	0	0	0	0	18	19	13
AHEAD+Random	**27**	**20**	**10**	**25**	-	0	0	0	0	0	21	22	19
AHEAD+Roulette	**26**	**20**	**9**	**26**	0	-	0	0	0	0	22	22	17
AHEAD+Deleter	**26**	**19**	**9**	**26**	**3**	0	-	0	0	0	**24**	**28**	19
AHEAD+UCB	**26**	**20**	**9**	**26**	**1**	**1**	0	-	0	0	23	23	19
AHEAD+Pursuit	**26**	**23**	**11**	**26**	**1**	0	0	0	-	0	**24**	26	**22**
AHEAD+NN	**27**	**21**	**10**	**27**	0	**1**	0	0	0	-	21	23	19

Table 4. Results of the best methods for the WVCP.

instance	BKS	RedLS			ILS-TS			HEAD+RedLS			AHEAD+Random			AHEAD+Deleter		
		best	mean	time	best	mean	time	best	mean	time	best	mean	time	best	mean	time
C2000.5	2144	**2131**	**2155.7**	18367	2244	2264.4	6423	2244	2257.9	7453	2220	2236.8	12962	2218	2236.3	1782
C2000.9	5477	**5439**	**5455.1**	23137	5847	5910.1	23014	5732	5748.2	12980	5732	5783.9	12491	5717	5758.8	12327
DSJC1000.1	300	303	306.9	5839	305	306.2	5819	304	305.6	7380	302	303.8	9348	**300**	**302.2**	12874
DSJC1000.5	1185	**1190**	**1206.9**	12204	1241	1267.7	21935	1225	1229.7	7011	1222	1228.2	5371	1224	1230.5	1476
DSJC1000.9	2836	**2828**	**2841.8**	22796	3004	3035.9	25345	2909	2926.5	820	2911	2928.7	12633	2907	2926.8	2379
DSJC500.1	184	187	194	702	185	187.3	7107	186	186.9	6594	185	186.5	10290	**184**	185.9	8022
DSJC500.5	685	707	712.5	27147	711	721.2	9150	709	712.6	2534	**706**	**711.5**	12516	709	713.5	5838
DSJC500.9	1662	**1667**	**1671**	9925	1709	1725.3	24351	1680	1683.5	4053	1678	1684.2	12644	1676	1682.8	8149
DSJC250.1	127	129	131.4	56	**127**	127.1	11901	**127**		4516	**127**		3729	**127**	127.2	3235
DSJC250.5	392	399	400.8	2602	**392**	**393.9**	10722	395	396.2	8349	393	395.2	9592	**392**	396.6	6028
DSJC250.9	934*	**934**	935	9679	**934**	935.1	14740	**934**	935.1	6741	**934**	**934.2**	8097	**934**	935	5011
DSJC125.5gb	240	243	252.7	0	**240**		132	**240**	240.9	4098	**240**		222	**240**		152
DSJC125.5g	71	72		1063	**71**		64	**71**		1609	**71**		86	**71**		104
DSJC125.9gb	604*	**604**		2	**604**		125	**604**		4	**604**		13	**604**		12
DSJC125.9g	169*	**169**		0	**169**		320	**169**		0	**169**		6	**169**		9
flat1000_50_0	924	**1152**	**1165.7**	6259	1213	1230.5	570	1181	1187.7	7544	1179	1186.3	4428	1180	1186.8	2952
flat1000_60_0	1162	**1196**	**1204.8**	1877	1247	1263.8	25765	1216	1227.2	10824	1213	1223.7	11726	1217	1224.5	9840
flat1000_76_0	1165	**1163**	**1183.2**	28084	1228	1242.2	16513	1192	1204	2214	1187	1203	10742	1196	1204	8938
latin_square_10	1480	**1505**	**1515.3**	14189	1555	1575	18924	1523	1532.5	11286	1510	1526.2	13987	1517	1527.8	8732
le450_15a	212	213	215.4	54	**211**	213.6	11684	**212**	212.8	6777	**212**	212.8	8819	**211**	**212.4**	10557
le450_15b	216	218	219.9	41	217	217.1	10346	**216**	217	3204	**216**	217.1	2736	**215**	**216.5**	11124
le450_15c	275	282	285.4	82	279	281.7	16288	277	279.4	8360	277	**278.8**	7220	278	279.4	4788
le450_15d	272	277	280.6	325	275	277.6	8456	274	276.1	6004	274	275.6	8759	**273**	275.2	13299
le450_25a	306	**306**	306.6	2881	**306**		142	**306**		161	**306**		169	**306**		131
le450_25b	307*	**307**	307.6	95	**307**		23	**307**		53	**307**		28	**307**		19
le450_25c	342	348	352.8	583	348	349.1	16413	347	348.1	180	**346**	**347.8**	5652	**346**	348	588
le450_25d	330	335	339.4	232	337	338.7	14212	333	334.4	5904	333	334.2	6282	333	334.2	9648
queen10_10	162	**162**	164.8	865	**162**		20	**162**		51	**162**		32	**162**		27
queen10_10gb	164	165	168.7	4790	**164**		172	**164**	164.4	4850	**164**		227	**164**		314
queen10_10g	43*	**43**	43.1	12	**43**		7	**43**		11	**43**		7	**43**		9
queen11_11	172	174	178	28766	**172**		6983	**172**	172.7	5668	**172**	172.1	3108	**172**		2207
queen11_11gb	176	177	178.6	1329	**176**		187	**176**		1583	**176**		436	**176**		396
queen11_11g	47	**47**	47.9	669	**47**		154	**47**		276	**47**		144	**47**		151
queen12_12	185	188	189.9	61	**185**	**185.2**	13770	186	186.3	6201	**185**	185.6	5997	**185**	185.4	7066
queen12_12gb	191	192	197.8	150	**191**		5019	**191**	191.3	5174	**191**		1521	**191**	191.1	3380
queen12_12g	50	**50**	51.5	986	**50**		1214	**50**		1464	**50**		1533	**50**		865
queen13_13	194	**194**	199.9	8	**194**	194.8	11188	**194**	194.2	5243	**194**		1560	**194**	194.1	1480
queen14_14	215	218	223.8	568	**215**	216.4	9862	216	216.6	7956	**215**	216.2	6384	**214**	**215.3**	8624
queen15_15	223	228	229.7	5806	225	226.5	15730	**224**	225.5	13260	**224**	225.1	10180	**224**	225.7	7200
queen16_16	234	237	240.8	17	237	238.3	15114	235	236.4	913	235	236	5610	235	236.4	11836
wap01a	545	557	577	995	**547**	**550.1**	20531	552	559.1	8178	549	553.6	14094	549	552.8	8874
wap02a	538	554	572.1	16183	**536**	**541**	21912	550	557.1	13884	541	546.1	7654	541	545.5	12994
wap03a	562	**569**	575.5	17878	572	575.5	22637	577	579.7	6992	573	576.3	8096	573	575.9	2944
wap04a	563	**567**	578.9	13939	**567**	**570.5**	7346	573	575.6	3152	570	573.2	1970	569	572.5	13790
wap05a	541	**542**	543.8	7719	**542**	**542.2**	11809	**542**	542.9	4471	**542**	543	12056	**542**	543.2	2772
wap06a	516	519	526.1	1575	**516**	**519.5**	6264	519	520.7	12180	518	521	9100	520	521.2	5978
wap07a	555	**554**	573	8460	565	569.2	16299	557	559.4	3360	558	559.8	12040	557	**559.2**	12460
wap08a	529	**536**	543.7	19557	543	546.9	19271	539	540.8	7452	539	541.2	1800	538	**540.1**	10608
#BKS		15/48			23/48			19/48			21/48			24/48		
#Best		24/48			25/48			19/48			22/48			28/48		
#Best Avg		11/48			21/48			11/48			19/48			19/48		

Detailed Results. First, we see in Table 4 that with the help of the eight hours of computation, RedLS is capable of finding five new scores (underlined score). ILS-TS is also able to find two new upper bounds with this longer execution time.

The RedLS local search alone remains better on large instances (e.g., C2000, latin_square and flat) than the different memetic versions using this local search procedure (HEAD + RedLS, and all AHEAD variants). Contrary to what was observed for the k-col, this shows that for very large WVCP instances, using the GPX crossover is not very beneficial. This can be explained by the fact that for the WVCP, only the maximum weight of each color group affects the score. Thus, for large instances, many different groupings of vertices are possible without impacting the score, which generally results in a very high distance between the two solutions S_1 and S_2 of the population. However, as observed in [21], the solution quality of an offspring built with the GPX crossover is poorer (higher fitness) if the individuals are too distant in the search space.

However, for medium-sized instances, such as le450_15a/b and queen14_14, the AHEAD memetic framework with an elitist operator selection (AHEAD + Deleter) significantly improves results and yields three new best upper bounds.

Regarding the operators selected by AHEAD, the two local search operators RedLS and ILS-TS are almost equally preferred, with a choice depending on the type of instance. On the other hand, unlike the k-col, the choice of crossover is almost exclusively oriented towards the most conservative crossover, GPX-9, particularly in combination with the local search ILS-TS. As mentioned above, this is due to the large distance between individuals in the population in the case of the WVCP.

4 Conclusion

The proposed AHEAD (Adaptive HEAD) framework is based on a population of two individuals and uses learning-driven operator selectors to determine a pair of local search and crossover to apply during the search process for solving a given instance of the k-coloring and weight vertex coloring problems. For both problems, the proposed approach shows advantages over versions without automatic selection of low-level operators. In the course of these experiments, we obtained three new best scores for the WVCP with the proposed AHEAD method, as well as a new best coloring with 404 colors for the very large and dense graph C2000.9.

AQ1

The work could be extended by considering a wider variety of complementary crossover procedures and local searches to be chosen by the high level operator selection strategy. Future work could also involve coupling the choice of operators with the setting of critical hyperparameters involved in these operators, such as the number of local search iterations to be performed at each generation, or the size of the tabu list.

Acknowledgment. We would like to thank Dr. Yiyuan Wang, [30] and Pr. Bruno Nogueira [22] for sharing their codes. This work was granted access to the HPC resources of IDRIS (Grant No. AD010611887R1) from GENCI and the Centre Régional de Calcul Intensif des Pays de la Loire (CCIPL). We are grateful to the reviewers for their comments.

References

1. Auer, P.: Using confidence bounds for exploitation-exploration trade-offs. J. Mach. Learn. Res. **3**(Nov), 397–422 (2002)
2. Blöchliger, I., Zufferey, N.: A graph coloring heuristic using partial solutions and a reactive tabu scheme. Comput. Oper. Res. **35**(3), 960–975 (2008)
3. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R.: Hyper-heuristics: a survey of the state of the art. J. Oper. Res. Soc. **64**(12), 1695–1724 (2013)
4. Drake, J.H., Kheiri, A., Özcan, E., Burke, E.K.: Recent advances in selection hyper-heuristics. Eur. J. Oper. Res. **285**(2), 405–428 (2020)
5. Elhag, A., Özcan, E.: A grouping hyper-heuristic framework: application on graph colouring. Expert Syst. Appl. **42**(13), 5491–5507 (2015)
6. Galinier, P., Hamiez, J.P., Hao, J.K., Porumbel, D.: Recent advances in graph vertex coloring. In: Zelinka, I., Snášel, V., Abraham, A. (eds.) Handbook of Optimization. Intelligent Systems Reference Library, vol. 38, pp. 505–528. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-30504-7_20
7. Galinier, P., Hao, J.K.: Hybrid evolutionary algorithms for graph coloring. J. Comb. Optim. **3**, 379–397 (1999)
8. Garey, M.R.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman. Fundamental (1997)
9. Goëffon, A., Lardeux, F., Saubion, F.: Simulating non-stationary operators in search algorithms. Appl. Soft Comput. **38**, 257–268 (2016)
10. Goudet, O., Grelier, C., Hao, J.K.: A deep learning guided memetic framework for graph coloring problems. Knowl.-Based Syst. **258**, 109986 (2022)
11. Goudet, O., Grelier, C., Lesaint, D.: New bounds and constraint programming models for the weighted vertex coloring problem. In: Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th–25th August 2023, Macao, SAR, China, pp. 1927–1934 (2023)
12. Grelier, C., Goudet, O., Hao, J.-K.: On Monte Carlo tree search for weighted vertex coloring. In: Pérez Cáceres, L., Verel, S. (eds.) EvoCOP 2022. LNCS, vol. 13222, pp. 1–16. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-04148-8_1
13. Grelier, C., Goudet, O., Hao, J.K.: Monte Carlo tree search with adaptive simulation: a case study on weighted vertex coloring. In: Pérez Cáceres, L., Stützle, T. (eds.) EvoCOP 2023. LNCS, vol. 13987, pp. 98–113. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-30035-6_7
14. Halldórsson, M.M., Shachnai, H.: Batch coloring flat graphs and thin. In: Gudmundsson, J. (ed.) SWAT 2008. LNCS, vol. 5124, pp. 198–209. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69903-3_19
15. Hertz, A., Werra, D.D.: Using tabu search techniques for graph coloring. Computing **39**(4), 345–351 (1987)
16. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
17. Liu, H., Beck, M., Huang, J.: Dynamic co-scheduling of distributed computation and replication. In: Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID 2006), vol. 1, pp. 9–pp. IEEE (2006)
18. Lü, Z., Hao, J.K.: A memetic algorithm for graph coloring. Eur. J. Oper. Res. **203**(1), 241–250 (2010)
19. Malaguti, E., Monaci, M., Toth, P.: An exact approach for the vertex coloring problem. Discret. Optim. **8**(2), 174–190 (2011)

20. Malaguti, E., Toth, P.: A survey on vertex coloring problems. Int. Trans. Oper. Res. **17**(1), 1–34 (2010)
21. Moalic, L., Gondran, A.: Variations on memetic algorithms for graph coloring problems. J. Heuristics **24**, 1–24 (2018)
22. Nogueira, B., Tavares, E., Maciel, P.: Iterated local search with tabu search for the weighted vertex coloring problem. Comput. Oper. Res. **125**, 105087 (2021)
23. Porumbel, D.C., Hao, J.K., Kuntz, P.: An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. Comput. Oper. Res. **37**(10), 1822–1832 (2010)
24. Porumbel, D.C., Hao, J.K., Kuntz, P.: An efficient algorithm for computing the distance between close partitions. Discret. Appl. Math. **159**(1), 53–59 (2011)
25. Prais, M., Ribeiro, C.C.: Reactive grasp: an application to a matrix decomposition problem in TDMA traffic assignment. INFORMS J. Comput. **12**(3), 164–176 (2000)
26. Sabar, N.R., Ayob, M., Qu, R., Kendall, G.: A graph coloring constructive hyper-heuristic for examination timetabling problems. Appl. Intell. **37**(1), 1–11 (2012)
27. Sghir, I., Hao, J.-K., Ben Jaafar, I., Ghédira, K.: A distributed hybrid algorithm for the graph coloring problem. In: Bonnevay, S., Legrand, P., Monmarché, N., Lutton, E., Schoenauer, M. (eds.) EA 2015. LNCS, vol. 9554, pp. 205–218. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-31471-6_16
28. Sun, W., Hao, J.K., Lai, X., Wu, Q.: Adaptive feasible and infeasible tabu search for weighted vertex coloring. Inf. Sci. **466**, 203–219 (2018)
29. Wang, Y., Cai, S., Pan, S., Li, X., Yin, M.: Reduction and local search for weighted graph coloring problem. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, pp. 2433–2441 (2020)
30. Wang, Y., Cai, S., Pan, S., Li, X., Yin, M.: Reduction and local search for weighted graph coloring problem. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, no. 0303, pp. 2433–2441 (2020)
31. Zaheer, M., Kottur, S., Ravanbakhsh, S., Póczos, B., Salakhutdinov, R., Smola, A.J.: Deep sets. In: Guyon, I., et al. (eds.) Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4–9 December 2017, Long Beach, CA, USA, pp. 3391–3401 (2017)