Journal of Combinatorial Optimization, 2009, 18(2): 99-123.
(The original publication is available at www.springerlink.com by DOI: 10.1007/s10878-008-9140-6)

Graph Coloring by Multiagent Fusion Search

Xiao-Feng Xie, Jiming Liu

Department of Computer Science, Hong Kong Baptist University,
Kowloon Tong, Hong Kong, China
xiexf@ieee.org, jiming@comp.hkbu.edu.hk

Abstract. A multiagent fusion search is presented for the graph coloring problem. In this method, each of agents performs the fusion search, involving a local search working in a primary exploitation role and a recombination search in a navigation role, with extremely limited memory and interacts with others through a decentralized protocol, thus agents are able to explore in parallel as well as to achieve a collective performance. As the knowledge components implemented with available structural information and in formalized forms, the Quasi-Tabu local search and grouping-based recombination rules are especially useful in addressing neutrality and ruggedness of the problem landscape. The new method has been tested on some hard benchmark graphs, and has been shown competitive in comparison with several existing algorithms. In addition, the method provides new lower bound solutions when applied to two large graphs. Some search characteristics of the proposed method are also discussed.

Keywords: Graph Coloring, Global Optimization, Multiagent System.

1. Introduction

Let $G=(\underline{V}, \underline{E})$ be an undirected graph, where $\underline{V}$ is a set of vertices and $\underline{E}$ is a set of edges, the graph coloring problem (GCP) is to partition $\underline{V}$ into K *color classes*, where each is a subset of $\underline{V}$ labeled with same color. For a *proper* coloring, each color class forms an *independent set*, which has no adjacent vertices. GCP is one of the most notorious models in graph theory: to compute the exact χ of an arbitrary graph requires the time $O(2.4422^{|V|})$ [21], and to color a graph with $\chi+2\cdot\left\lfloor\frac{\chi}{3}\right\rfloor-1$ colors is still *NP-hard* [44], where χ is the chromatic number. It also has various applications, such as timetabling [22], register allocation [62], and some others [4, 32].

The *landscape* paradigm has been used for search in general [58]. Formally, global structural information of the optimization task is represented as a landscape [58] containing two essential ingredients, i.e., the *representation space* S_R and the *cost function* $f(\vec{s})$. Each *state* $\vec{s}\in S_R$ is associated with a potential solution of the task. The function $f(\vec{s})$, which is to be minimized, is used for measuring the *quality* of each $\vec{s}$. The rationale of problem solving is then to find the state(s) with better quality by moving in the landscape with the search strategies utilizing structural information.

The GCP landscape can be studied from a point of view on its geometric properties [58] under specified neighborhood structure(s), focusing on the *ruggedness*, i.e., the

distribution of local minima, and the *neutrality*, i.e., the existence of *plateaus*, where each plateau is a cluster of the neighboring states in the same quality. The strategies based on local structural information may be strongly fooled by local minima [10]. For GCP, the existence of giant plateaus has been shown [52]. The neutrality becomes significant when some *benches* tend to be very large [26], as studied in Satisfiability Problem [60], where each bench is a plateau but not a local minimum.

Local search (LS) [30, 60, 64], which improves each incumbent state by neighborhood moves, has been applied to solve various problems successfully. A search strategy is defined as *stable* if it only allows the moves from one state to another with $\Delta f \leq 0$. Both the *greedy* ($\Delta f < 0$) and *plateau* moves ($\Delta f \equiv 0$) are stable. Each stable LS strategy is stuck into the local minimum that it first encounters. Vertex Descent strategy [34] is a simple example of a stable LS strategy.

Noise strategies [61], which are not stable, allow the LS to make occasional *uphill* moves ($\Delta f > 0$) to explore a rugged landscape. Typical examples include Random Walk [11, 47], Tabu Search [38], Simulated Annealing [41], etc. Noises may turn an incomplete LS strategy into *probabilistically approximately complete* (PAC) [39] which achieves an optimum state with a probability one as the run-time approaches infinity. However, for searching efficiently under a reasonable cutoff time, a strategy is preferable to exploit problem structural information rather than to perturb blindly.

Recombination search (XS), which generates a state by combining the positive clues from two source states, utilizes the difference between two source states and leads to an adaptive leaping [51]. The Graph-Adapted Recombination [24] was proposed by hybridizing *averaging* [60] with *min-conflicts*. For purpose of addressing *permutation symmetry* [34, 70], grouping-based XS methods [23], such as Greedy Partition Crossover [29], Union of Independent Sets [19], etc [22], were proposed.

Fusion search (FS), a concept borrowed from Multi-Step Crossover Fusion [57], is defined as a chained combination between an XS and an LS. Although FS has the same interface as XS, the two components in FS have different search roles, where the XS finds a promising state as an incumbent state of the LS, while the LS locally improves this state. The idea of FS has been used in practice [24, 29, 34, 60].

Various metaheuristic frameworks have been applied in solving GCP, which use LS, XS, or even FS as their search components. The examples include Ant Systems [9], Adaptive Memory [31], Scatter Search [37], Immune Algorithms [16], Genetic Algorithms [3, 19, 24, 34, 54], etc. In a framework, it is important to manage the source information efficiently for its search components by facilitating the emergence of the positive clues as well as maintaining the diversity of information.

Autonomy oriented computing [48] stresses modeling the flexible autonomy of entities and the self-organization of them for a specific goal. It is possible to preserve the diversity of the positive clues in the system [15] with a local diffusion effect [68], especially when each entity possesses its private memory [69]. Moreover, allowing agents to use memorized information to adjust their behaviors enable us to study more intelligent agents [46]. Specifically, the compact multiagent optimization framework [67], which supports the cooperative search by multiple compact agents, has been applied in solving hard computational problems, such as Numerical Optimization Problem [67] and Traveling Salesman Problem [68]. As a simple multiagent system [63], it may have the potentials of parallelism, robustness, and scalability.

In this paper, a multiagent fusion search (MAFS) for GCP is presented. In Section 2, a multiagent optimization framework in general is described to support the cooperative search by the multiple agents under the law of *socially biased individual learning* (SBIL) [28, 69]. In Section 3, MAFS is realized in a simplified way of the framework, where MAFS not only supports the fusion search for each agent based on extremely limited declarative knowledge, but also works in a decentralized way. In Section 4, the knowledge components of MAFS are implemented to address neutrality and ruggedness of the GCP landscape by utilizing available structural information. In Section 5, the characteristics of MAFS are studied by performing experiments on some hard graphs [24, 42]. Finally, this paper is concluded in Section 6.

2. Multiagent Optimization Framework

In order to achieve the goal of finding solution(s) with at least reasonable quality, the multiagent optimization framework is organized with autonomous entities [49, 69] that self-organize by manipulating certain knowledge components which are realized according the *internal representation* (IR) of task and related world knowledge [55].

The framework consists of N_P active entities, called compact *agents*, and a daemon entity, called *environment* (ENV) [66]. For simplicity, the agents are homogenous in the sense that they have the same organization structure. Each agent has an ability to generate new states in S_R by manipulating available knowledge based on simple rules [69]. Moreover, agents achieve a collective performance through interacting with each other according to the *interaction protocol* (IP) under the support of ENV.

2.1 Basic Concepts

The internal representation (IR) encapsulates the primary knowledge related to the optimization task, which contains global structural information, i.e., the landscape [58], and related local structural information. Using the landscape is quite general in solving hard computational problems [58]. Local structural information may reduce the local computation efforts and may lead the search into the promising directions.

The general problem solving capability emerges from an interaction of declarative and procedural knowledge [1, 55]. Declarative knowledge is represented in symbol structures called *chunks*, and procedural knowledge is represented in elementary information processes called *rules*. All required knowledge elements are instantiated according to the IR and related world knowledge, where only the knowledge elements using local structural information are regarded to be strongly problem-dependent.

Each chunk is a certain declarative data structure containing a meshed set of patterns [20, 35], and the content of a chunk is designated by its name. Each rule is represented as $R_{\text{I\_KEY}}^{\boxed{\text{I\_NAME}}}$ [68]. The subscript I_KEY designates a high-level interface used for handling with specific input/output parameters, where each parameter is either a knowledge element or a simple data type, and the superscript I_NAME designates the low-level realization. Each knowledge element may have specific setting parameters, where the value of each parameter can be tuned before a run.

2.2 Compact Agent

A compact agent is a socially situated autonomous entity [49, 69] capable of making decisions for itself which is subject to limitations of available knowledge.

Each agent has two declarative knowledge sources. Firstly, it possesses a long-term memory called M_A for supporting individual learning. Moreover, at least one of the chunks in M_A is publicly accessed by the external world [49]. Secondly, it refers to an equivalent *social memory* (M_{SE}), which is owned by ENV and contains the chunks for achieving socially biased learning [25, 28]. Moreover, the agent possesses a private buffer memory called M_G for temporarily holding the newly generated chunks.

Each memory is defined by the chunks it possesses, where each chunk aggregates certain particularities of the landscape. Moreover, each chunk in memory is updated [35] by its owner only. Typical examples of a chunk in a memory include a state in S_R, a state set [29, 68], or a special data structure, such as an ensemble of independent sets [31], a pheromone matrix [7], a state in priority space [43], etc.

The search capability in the landscape is achieved by the *generate-and-test* rule (R_{GT}) [18, 69], which contains a *generating* part (R_G), a *testing* part (R_T), and a *solution-extracting* part (R_S) [68]. The law of behavior is socially biased individual learning (SBIL) [25, 28, 69], as a fast-and-frugal heuristic in bounded rationality [33], which is adopted by many species for adapting in the real world with limited time and resource and is a mix of reinforced practice of individual experience and social influence. First, according to the current chunks in M_A and M_{SE}, the R_G generates new chunks and stores them into M_G immediately. Afterwards, based on the chunks in M_G, the R_T updates M_A, and the R_S extracts inclusive valid states and exports them to ENV, respectively. The R_S has no influence on the solving process and is thus realized in the simplest way. If without loss of generality, the R_T only produces reflex behaviors that may determine certain nontrivial properties of the chunk(s) in M_A [67].

2.3 Environment

The environment (ENV) [48] is a daemon entity providing background supports for the cooperation among the agents by encapsulating available resources, even may include the physical infrastructure [66], if necessarily. Here ENV plays two roles.

First, it holds a *solution-depositing* module, which is simply realized by storing the best-quality state of all the states that are exported by the R_S of all agents.

Secondly, it manages resources and services [66] for all the agents. Such as, a) it constructs the initial contents of chunks in M_A of all the agents with the *memory-constructing* rule (R_{INI}); and b) it organizes the corresponding M_{SE} for all agents based on the available declarative knowledge through the interaction protocol (IP).

2.4 Working Process

The framework is initialized at t=0. All the N_P agents are constructed, and the R_{INI} rule is executed to construct the chunks in M_A for all agents. Then states are extracted from such chunks by R_S and are submitted to the solution-depositing module in ENV. Then all the knowledge components, including the IP in ENV, are instantiated.

The framework runs in iterative learning cycles. By running in a Markov chain, the system behavior in the tth ($t \in [1, T_{MAX}]_{\mathbb{Z}}$) cycle only depends on the system status in the (t-1)th cycle, where T_{MAX} is the maximal number of cycles. The learning process is terminated as if the proper solution is found, or if the condition $t \equiv T_{MAX}$ is satisfied.

Moreover, each cycle contains two sequential clock steps: the C_RUN step and the C_POST step. The R_G rule is executed at the C_RUN step, and the R_T and R_S rules are executed at the C_POST step. The using of the two synchronizing steps simply ensures the environment being unchanged during a generating process for all agents.

At each cycle (t>0), all the agents are activated in turn. The socially biased learning process by the ith activated agent in the tth cycle can be represented as:

$$M_{A(i)}^{(t)},\ M_{SE(i)}^{(t)} \xrightarrow[\text{(C\_RUN)}]{R_G} M_{G(i)}^{(t)} \begin{cases} \xrightarrow[\text{(C\_POST)}]{R_T} M_{A(i)}^{(t+1)} \\ \xrightarrow[\text{(C\_POST)}]{R_S} \{\vec{s}\} \rightarrow \text{ENV} \end{cases} \tag{1}$$

where each $M_{SE(i)}^{(t)}$ is organized by the IP in ENV. Moreover, the chunks in $M_{G(i)}^{(t)}$ will be cleared at the end of such a learning process.

At the end of each cycle, if necessarily, the information related to ENV is updated.

2.5 Summary

In summary, the framework is represented as a tuple, i.e., <IR, T_{MAX}, N_P, M_A, M_G, M_{SE}, R_G, R_T, R_S, R_{INI}, IP>, where both T_{MAX} and N_P are simple parameters.

Given a known IR, the three types of memories, including M_A, M_G, and M_{SE}, can be specified in advance according to the names of chunks that each memory possesses, although the contents of chunks have to be varied during the runtime.

The other knowledge components can be specified rather independently through using different memories. For each agent, R_{INI} simply uses M_A; R_S only depends on M_G; R_T works on both M_A and M_G; and R_G employs all the three available memories.

For ENV, IP accesses the chunks in M_A shared by all agents, and then organizes M_{SE} for every agent. Hence, IP may be realized in a quite sophisticated way, if necessarily. However, simple implementations are often considered, if possible.

The number of setting parameters in such a framework is not necessarily large since many of the knowledge components may have none or few parameters. In addition, in order to focus our studies on certain interesting components, we may fix many components in the simple forms. Those components with no variety, e.g., the solution-depositing module in ENV, are out of our further concentration.

3. Multiagent Fusion Search (MAFS)

The MAFS is an optimization system realized by using simple forms of knowledge components in the multiagent framework. It has three main features.

Firstly, memories are specified with extremely limited declarative chunks. M_A and M_G both contain one state in S_R, which are called $\vec{s}_A^{(t)}$ and $\vec{s}_G^{(t)}$, respectively, where $\vec{s}_A^{(t)}$ is publicly accessed by ENV since it is the only one chunk in M_A. M_{SE} holds an equivalent state set called $\underline{X}_{SE}^{(t)}$, which refers to a set of states.

Secondly, a decentralized IP is considered for supporting the interactions between the agents. This is important, as in the real world, animals may observe neighbors for achieving socially biased learning [25, 28], which leads to a cumulative evolution of knowledge that no single individual could invent on its own [2]. This IP employs a directed network topology model (IP_{NET}). Each agent is associated with one node in the network, and the node stores a reference of the publicly opened knowledge in M_A of the agent. A directed connection from the node A to B indicates that the agent B can use the referenced knowledge of the node A. For each agent, $\underline{X}_{SE}^{(t)}$ contains all the referenced chunks of the nodes connected to the node associated with it, which is a subset of $\underline{X}_{S}^{(t)} = \{ \vec{s}_{A(i)}^{(t)} \mid i \in [1, N_P]_{\mathbb{Z}} \}$ and may be different for different agents.

Thirdly, in order to formalize some well-studied LS and XS strategies and to study certain novel strategies, R_G is realized in a tuple $<R_{SP}, R_{FS}>$, where $R_{FS}=R_{XS}+R_{LS}$ is the fusion search (FS), and the *state-picking* rule (R_{SP}) serves as a simple *knowledge lens* [20] to choose one state from $\underline{X}_{SE}^{(t)}$ as the input information of R_{FS}. The *chaining operator* ('+') indicates that the recombination search rule (R_{XS}) and the local search rule (R_{LS}) are *chained*, i.e., the output of the former rule R_{XS} is exactly the input of the latter rule R_{LS}. The fusion is a concept borrowed from Multi-Step Crossover Fusion [57], which is actually an XS strategy designed with an extension of a LS strategy. Moreover, R_{FS} may be understood as a special R_{XS} in consideration of the same input/output parameters of them, which possesses two components in different search roles: the R_{LS} rule playing a primary exploitation role and the R_{XS} rule working in a navigation role to find a promising state as the incumbent state for the R_{LS}.

To provide a straightforward understanding, Figure 1 shows the pseudo code of MAFS from the viewpoint of a population-based optimization algorithm, where all the knowledge publicly accessed by the IP_{NET} in ENV, i.e., $\underline{X}_{S}^{(t)} = \{ \vec{s}_{A(i)}^{(t)} \mid i \in [1, N_P]_{\mathbb{Z}} \}$, corresponds to a virtual population of states, although $\vec{s}_{A(i)}^{(t)}$ is actually located in M_A of the *i*th agent. For each agent, $\underline{X}_{S}^{(t)}$ is at least transparent to it, since it simply refers to $\underline{X}_{SE}^{(t)}$, which is organized by the IP_{NET} model. Moreover, because each agent shares all its declarative knowledge in M_A with the external world, $\underline{X}_{S}^{(0)}$ can represent the knowledge to be constructed by R_{INI} into the memory M_A of all agents.

At the C_RUN step, the R_G rule of each agent works in the following steps: 1) two states $\vec{s}_{base}^{(t)}$ and $\vec{s}_{ref}^{(t)}$, are chosen from the input information, where $\vec{s}_{base}^{(t)} = \vec{s}_{A}^{(t)}$ and $\vec{s}_{ref}^{(t)}$ is a state picked by the R_{SP} from $\underline{X}_{SE}^{(t)}$; b) the R_{XS} part of R_{FS} generates one child state called $\vec{s}_{inc}^{(t)}$ by using both $\vec{s}_{base}^{(t)}$ and $\vec{s}_{ref}^{(t)}$ as the parent states; and c) the R_{LS} part of R_{FS} further improves $\vec{s}_{inc}^{(t)}$ and finally stores it as $\vec{s}_{G}^{(t)}$ in M_G.

At the C_RUN step, The R_T rule of each agent replaces $\vec{s}_{A}^{(t)}$ by $\vec{s}_{G}^{(t)}$ according to a specific criterion. The R_S rule is not mentioned in Figure 1 since it simply exports $\vec{s}_{G}^{(t)}$ to the solution-depositing module in ENV.

In MAFS, the two parent states of R_{FS} serve different roles, especially in consideration of the multiple cycles in a run. Under the law of SBIL, the parent $\vec{s}_{base}^{(t)}$ always uses the $\vec{s}_{A}^{(t)}$ in M_A of each agent as its input and the state generated by R_{FS} is always the candidate of $\vec{s}_{A}^{(t+1)}$, while the parent $\vec{s}_{ref}^{(t)}$ uses a state from $\underline{X}_{SE}^{(t)}$ as its input in a stochastic way. Hence, R_{FS} may be interpreted from a viewpoint of a guided local search process, where $\vec{s}_{A}^{(t)}$ serves as an incumbent state to be improved and the state which is picked from $\underline{X}_{SE}^{(t)}$ serves as the guiding information.

In the multiagent framework, since each agent possesses its own long-term declarative memory, it is possible to preserve the diversity of positive clues in the system. By utilizing their individual experiences, the agents are able to explore in parallel, which may significantly increase the probability of escaping from local minima in the rugged GCP landscape. With R_{XS}, the agents are facilitated by the social influence of $\mathrm{IP}_{\mathrm{NET}}$, thus achieve a collective performance searching faster than they work independently. In addition, usage of R_{LS} in the FS is important in obtaining good states, especially when some benches in the GCP landscape are huge.

Data: IR(G, K) /* graph: $G=(\underline{V}, \underline{E})$, color number: K */
% N_P, T_{MAX} /* simple setting parameters */
% R_{SP}, R_{XS}, R_{LS}, R_T, R_{INI}, $\mathrm{IP}_{\mathrm{NET}}$ /* knowledge elements: instantiated by IR */
Result: $\vec{s}^*$ (the best state found) /* held by solution-depositing module in ENV */
begin
  t=0 /* initialization stage */
  $\underline{X}_S^{(0)} = R_{INI}(\mathrm{IR})$ /* $\underline{X}_S^{(t)} = \{ \vec{s}_{A(i)}^{(t)} \mid i \in [1, N_P]_{\mathbb{Z}} \}$, $\vec{s}_{A(i)}^{(t)}$ belongs to the ith agent */
  $\vec{s}^* = best(\mathrm{IR}, \underline{X}_S^{(0)})$ /* $best(\mathrm{IR}, \underline{X}_S^{(0)})$: returns the best state in $\underline{X}_S^{(0)}$ */
  while ($t<T_{MAX}$ **and** $f(\vec{s}^*)>0$) **do** /* termination criteria */
    $t=t+1$ /* iterative cycles */
    for i=1 to N_P, **do** /* C_RUN step, for the ith agent */
      $\underline{X}_{SE(i)}^{(t)} = \mathrm{IP}_{\mathrm{NET}}(\mathrm{IR}, \underline{X}_S^{(t)})$ /* organizes $\underline{X}_{SE(i)}^{(t)}$ by $\mathrm{IP}_{\mathrm{NET}}$ */
      $\vec{s}_{base}^{(t)} = \vec{s}_{A(i)}^{(t)}$, $\vec{s}_{ref}^{(t)} = R_{SP}(\mathrm{IR}, \underline{X}_{SE(i)}^{(t)})$ /* filters input information */
      $\vec{s}_{inc}^{(t)} = R_{XS}(\mathrm{IR}, \vec{s}_{base}^{(t)}, \vec{s}_{ref}^{(t)})$ /* performs recombination search */
      $\vec{s}_{G(i)}^{(t)} = R_{LS}(\mathrm{IR}, \vec{s}_{inc}^{(t)})$ /* refines $\vec{s}_{inc}^{(t)}$ and stores it as $\vec{s}_{G(i)}^{(t)}$ in M_G */
    for i=1 to N_P, **do** /* C_POST step, for the ith agent */
      $\vec{s}_{A(i)}^{(t+1)} = R_T(\mathrm{IR}, \vec{s}_{A(i)}^{(t)}, \vec{s}_{G(i)}^{(t)})$ /* determines which state is $\vec{s}_{A(i)}^{(t+1)}$ */
      if ($f(\vec{s}_{G(i)}^{(t)}) < f(\vec{s}^*)$) **then** $\vec{s}^* = \vec{s}_{G(i)}^{(t)}$ /* stores $\vec{s}_{G(i)}^{(t)}$ as $\vec{s}^*$ if it is better */
end

Figure 1. Pseudo code of MAFS from the viewpoint of a population-based algorithm.

In summary, MAFS can be represented as a tuple, i.e., <IR, T_{MAX}, N_P, R_{SP}, R_{XS}+R_{LS}, R_T, R_{INI}, $\mathrm{IP}_{\mathrm{NET}}$>. All its components can be realized in a rather decoupled way since the memory specification is known. Moreover, many of its components may be not strongly problem-dependent if they do not use any local structural information. For example, the $\mathrm{IP}_{\mathrm{NET}}$ model may not utilize any structural information, or both R_{SP} and R_T are suggested to only use (or even not use) the global structural information of IR.

4. The Implementation for GCP

The implementation of the knowledge components in MAFS is especially focused on R_{XS} and R_{LS}, because they play the major roles in tacking with neutrality and ruggedness of the GCP landscape in the multiple cycles of a run, although R_{INI}, which constructs the totally N_P states at t=0, may also utilize local structural information to facilitate the search process through providing a good starting status, if necessarily.

Formalized forms are used to realize the rules of R_{LS} and R_{XS}, which is important not only in stressing the difference between various realizations, but also in leaving certain flexibility to develop novel variants locally.

4.1 Internal Representation (IR)

The primary input information of the graph coloring problem (GCP) contains both the graph, i.e., $G=(\underline{V}, \underline{E})$, and the number of available colors, i.e., K.

Normally, a preliminary data structure, i.e., the string-based *assignment* ($\hat{s}$) [23], is considered. Each $\hat{s}$ has $|\underline{V}|$ elements, where each element corresponds to a vertex and can be assigned a color value. An assigned vertex is called *critical* [30] if its *violation number* (*vio*), i.e., the number of vertices within the same color class that are adjacent to the vertex, is larger than 0. The number of assigned vertices is called V_A. A *configuration* is then defined as a *complete assignment* with $V_A \equiv |\underline{V}|$.

For the GCP landscape, each configuration is a state $\vec{s} \in S_R$, where S_R is an integer representation space with $s_{(j)} \in [1, K]_{\mathbb{Z}}$ for $\forall j \in [1, |\underline{V}|]_{\mathbb{Z}}$, $s_{(j)}$ is the color of the jth vertex of $\vec{s}$. The cost function is $f(\vec{s}) = \sum_{j=1}^{|\underline{V}|} vio(\vec{s}, j) \big/ 2$, where $vio(\vec{s}, j) \geq 0$ is the violation number of the jth vertex. Then an optimal solution is a state $\vec{s}^*$ that satisfies $f(\vec{s}^*) \equiv 0$, which means all its vertices are not assigned in the critical status.

The local structural information relies on the *adjacency matrix*. Any of edge in $\underline{E}$ which has two adjacent vertices j_a and j_b are described with the TRUE values at the two corresponding entries (j_a, j_b) and (j_b, j_a) in the Boolean $|\underline{V}| \times |\underline{V}|$ matrix.

An assignment $\hat{s}$ can be simply constructed with some heuristics which utilize the local structural information from the adjacency matrix involving the distribution of node degrees [65], most-constrained vertices [45], etc. Examples of these heuristics include *DSatur* [8], *XRLF* [41], *lmXRLF* [45], etc.

Moreover, each assignment $\hat{s}$ with $V_A < |\underline{V}|$ can be constructed into a state $\vec{s} \in S_R$ by a *vertices-assigning* rule (R_{VA}). Each unassigned vertex is assigned a randomly chosen color [29] by the *randomizing* R_{VA} rule ($R_{VA}^{[R]}$) and a color with the minimal violation number by the *min-conflicts* R_{VA} rule ($R_{VA}^{[MC]}$) [24, 37].

4.2 Local Search

A local search strategy (LS) tries to achieve improvement on an incumbent state with certain neighborhood moves. For GCP, as one of the representative models of Constraint Satisfaction Problems [47], LS strategies based on 1-*moves* [30] are often considered, since 1-*moves* can be significantly speeded up by associating each state $\vec{s}$ with a violation table [47]. Here a 1-*move* changes the color of a single vertex in the incumbent state. In addition, 1-*moves* possess the *connectivity* property [56], i.e., there exists a finite sequence of such moves to achieve the optimum solution from any valid state. Many sophisticated moves, such as *Kempe chain* [41], *Shuffle* [27], etc., can be represented by a finite sequence of 1-*moves*.

For GCP, the violation table is simplified as a $|\underline{V}| \times K$ violation matrix γ_V, in which each entry $\gamma_V(j,k) \geq 0$ ($j \in [1, |\underline{V}|]_{\mathbb{Z}}$, $k \in [1, K]_{\mathbb{Z}}$) is the number of vertices within the

*k*th color class of $\vec{s}$ adjacent to the *j*th vertex. The initialization of such a matrix takes the time complexity $O(|\underline{V}|\cdot K)$. Each delta value $\Delta f = \gamma_V(j, s_{(j)}) - \gamma_V(j, k_x)$ can be obtained in constant time before the color of the *j*th vertex changes from $s_{(j)}$ to k_x. If a 1-*move* is actually performed, both columns $s_{(j)}$ and k_x of the matrix are updated, where the updating takes $O(|\underline{V}|)$. The matrix γ_V is not the same as the matrix Δ [24], in which entry $\Delta(j,k)$ represents the effect of changing the color of node *j* to the color *k*, where the initialization takes $O(|\underline{V}|^2 \cdot K)$ for and each 1-*move* takes $O(|\underline{V}|\cdot K)$.

For the purpose of representing various R_{LS} strategies in formalized forms, three hierarchical levels are used, i.e., a) the *local* level (R_{LSL}), b) the *round* level (R_{LSR}), and c) the *meta* level, if necessarily. A basic R_{LS} strategy can be achieved by a tuple $<R_{LSL}, R_{LSR}>$. A *meta* R_{LS} strategy can then be achieved by chaining certain R_{LS} strategies being on the same incumbent state. For a basic R_{LS}, it is stable if its R_{LSL} only allows stable moves. For a *meta* R_{LS}, it is stable if all its component R_{LS} strategies are stable.

For convenience, the best state found so far by the R_{LS} rule is called $\vec{s}_*$, which is recorded only when a LS strategy is not stable.

The *local* level. The R_{LSL} decides a destination color for 1-*move* at each vertex. Here a *color list* ($\underline{\Gamma}$) contains certain colors. For the *j*th vertex, the color is randomly selected from a *candidate* color list corresponding to the *input* $\underline{\Gamma}_{(j)}$ with the current color $s_{(j)}$ excluded. The vertex is defined as *fixed* if the candidate color list is empty.

With the violation matrix γ_V, one simple way is to define an input color list is by utilizing $\gamma_V(j,k)$ values. For example, the *least-violation* $\underline{\Gamma}_{(j)}$, i.e., $\underline{\Gamma}_{LV(j)}$, contains all colors with the minimum violation value of the *j*th vertex.

The 1-*moves* can be further guided by using a $|\underline{V}|\times K$ *tabu matrix* [38]. If a 1-*move* leading to a state no better than $\vec{s}_*$ is performed, then its original color is declared as *tabu* for a certain number of such 1-*moves* (called *tabu tenure*). The tabu tenure is calculated as $U_{\mathbb{Z}}(A) + \alpha \cdot V_C$ [29], where $U_{\mathbb{Z}}(A)$ returns an integer value selected in [0, *A*-1]$_{\mathbb{Z}}$ at random and V_C is the number of the critical vertices in the current $\vec{s}$. The default values of *A* and α are 10 and 0.6, respectively [29]. Here the second part, i.e., $\alpha \cdot V_C$, provides a self-adaptive scheduled neighborhood selection; and the first part, i.e., $U_{\mathbb{Z}}(A)$, introduces certain fluctuation into such scheduled process.

For the input $\underline{\Gamma}_{(j)}$, the *random walk* R_{LSL} ($R_{LSL}^{\boxed{RW}}$) uses the list of all possible colors; the *least* R_{LSL} ($R_{LSL}^{\boxed{L}}$) uses the list $\underline{\Gamma}_{LV(j)}$; and the *Quasi-Tabu* R_{LSL} ($R_{LSL}^{\boxed{QT}}$) uses the list $\underline{\Gamma}_{LV(j)} \cap \underline{\Gamma}_{NT(j)}$, where the *non-Tabu* $\underline{\Gamma}_{(j)}$ ($\underline{\Gamma}_{NT(j)}$) is defined as all the colors that are not in tabu status of the *j*th vertex. Both $R_{LSL}^{\boxed{L}}$ and $R_{LSL}^{\boxed{QT}}$ only allow stable moves.

The *round* level. The R_{LSR} executes R_{LSL} on selected vertices in a specified order during a round. The *minimal-critical* R_{LSR} ($R_{LSR}^{\boxed{MC}}$) selects a 1-*move* with the maximum deduction of $f(\vec{s})$ by examining all critical vertices [29]. The *systematic* R_{LSR} ($R_{LSR}^{\boxed{SYS}}$) [34] takes each unfixed vertex in turn, and performs each 1-*move* based on a specified R_{LSL} rule. The *probabilitistic* R_{LSR} ($R_{LSR}^{\boxed{P}}$) takes each vertex in turn, and then performs the 1-*move* on the vertex selected with a probability of $V_{RW}/|\underline{V}|$ ($V_{RW} \in (0, |\underline{V}|]_{\mathbb{R}}$).

The *meta* level. This level manages one or more R_{LS} strategies into a meta LS strategy. One simple way is to combine different R_{LS} strategies by using the chaining operator.

In addition, the *local cutoff criterion* (R_{CCL}) is often considered, where one R_{LS} rule is executed in multiple rounds. Generally, the execution of R_{CCL} is always terminated if all vertices are fixed during a round. Specifically, the *deterministic* R_{CCL} ($R_{CCL}^{[D]}$) also terminates the search in exactly L_C rounds [29]; while the *improvement-based* R_{CCL} ($R_{CCL}^{[I]}$) also terminates if no further improvement on the $\vec{s}_*$ occurs for L_I rounds [11, 34]. In the case that large plateaus exist in the GCP landscape, it is difficult in assuring if a local minimum is actually reached even as L_C>1 or L_I>1.

The Instances. The *random walk* strategy ($R_{LS}^{[RW]}$) can be represented by a tuple < $R_{LSL}^{[RW]}$, $R_{LSR}^{[P]}$ >. The *Vertex Descent* strategy (VDS) [34] can be represented as << $R_{LSL}^{[L]}$, $R_{LSR}^{[SYS]}$ >, $R_{CCL}^{[I]}$ >. The *Quasi-Tabu* strategy ($R_{LS}^{[QT]}$) is defined as << $R_{LSL}^{[QT]}$, $R_{LSR}^{[SYS]}$ >, $R_{CCL}^{[I]}$ >.

As an intermediate version between VDS [34] and *Tabucol* [38], $R_{LS}^{[QT]}$ is not completely new since it inherits the traits from both of them. However, as an essential difference from *Tabucol*, $R_{LS}^{[QT]}$ is a stable strategy, which may be terminated early if all its vertices are fixed during a round due to the restriction of the tabu matrix. The tabu matrix is updated only while 1-*move* is searching in a plateau. As same as VDS, it cannot escape from the local minimum it first encounters. But it may be more efficient in finding the exits from benches by utilizing the tabu matrix.

4.3 Recombination Search

The *grouping-based* R_{XS} ($R_{XS:G}$) is a R_{XS} rule based on the grouping method [19, 23, 29, 37]. Here a *group set* (H) is defined as a set of K *groups*, where each group contains a set of vertices. By using groups, the *permutation symmetry* [34, 70], which has massive redundancy ($\equiv K!$) for labeling the colors, can be broken naturally.

The numbers of the total and the distinct vertices in an H are called V_{HT} and V_{HD}, respectively. Then an H with $V_{HT} \equiv V_{HD}$ is defined as a *simplex H*, where each of its vertices only exists in one group. Each assignment, and hence each configuration state, has an *equivalent* simplex H by simply taking each color class as a group. A *stable H* is defined as an H that each of its groups is an independent set (IS).

For the rugged GCP landscape, it has been suggested that good states may contain a fairly robust '*core*' [34]. The exact core, or called *backbone* [70], may not exist in a meaningful size due to the existence of giant plateaus which contain the majority of solutions [52]. The "big valley" hypothesis [5, 57, 70], which has been validated in many hard computational problems, suggests that better local minima tend to have smaller distance from the closest optimum by sharing common partial structures. For GCP, such partial structures may be associated with groups in a certain way.

The concept of *complex core* (*c-core*) is introduced here: each H has one exactly c-core, i.e., a stable group set defined as a subset of the H where all its critical vertices are excluded. For each H, its *c-core size* (V_{CC}) is the V_{HD} of its c-core and is not larger than the V_{HD} of the H. The c-core of each stable H is exactly the stable H itself.

In order to navigate in the rugged landscape, the basic principle of realizing a $R_{XS:G}$ rule is to combine the positive partial structures associated with the parent states as well as to allow the adaptive leaps into new local valleys. Formally, there is

$R_{XS:G}$=<R_{GPP}, R_{GGP}, R_{GVR}, R_{VA}>, which contains four parts working in sequential steps. First, two source states $\vec{s}_{base}^{(t)}$ and $\vec{s}_{ref}^{(t)}$ are translated into two equivalent group sets H_{base} and H_{ref}, respectively. The three early parts, i.e., *preprocessing* (R_{GPP}), *group-picking* (R_{GGP}), and *vertices-removing* (R_{GVR}), which generates a *simplex H* called H_{inc} by operating on the two parent group sets, i.e., H_{base} and H_{ref}. Afterwards, the H_{inc} is translated back into an equivalent assignment. In the last step, the assignment is constructed into a state $\vec{s}_{inc}^{(t)}$, by a *vertices-assigning* (R_{VA}) rule (see Section 4.1).

Preprocessing. The R_{GPP} preprocesses each input H of H_{base} and H_{ref} into a group set containing suitable positive clues. The *equivalent* R_{GPP} ($R_{GPP}^{\boxed{E}}$) returns the original input H [23, 29]. The *IS* R_{GPP} ($R_{GPP}^{\boxed{IS}}$) reduces each group in the H into an independent set (IS) by removing each critical vertex with a maximum number of neighbors in the group [19, 37]. The *MIS* R_{GPP} ($R_{GPP}^{\boxed{MIS}}$) further expands each IS into a maximum IS [21] by inserting each of the vertices with a minimal number of neighbors in the IS [31].

Either $R_{GPP}^{\boxed{IS}}$ or $R_{GPP}^{\boxed{MIS}}$ transforms each input H into a stable one. Moreover, the H outputted by $R_{GPP}^{\boxed{IS}}$ is a subset of the input H and a subset of the H outputted by $R_{GPP}^{\boxed{MIS}}$.

The V_{CC} size of the group set outputted by $R_{GPP}^{\boxed{IS}}$ is not smaller than that by $R_{GPP}^{\boxed{E}}$ since certain vertices in the input H may no longer be critical as other critical vertices are removed, and the V_{CC} size by $R_{GPP}^{\boxed{MIS}}$ is obviously not smaller than that by $R_{GPP}^{\boxed{IS}}$.

Group-Picking. The R_{GGP} generates the group set H_M by picking out K groups from the both parent group sets, i.e., H_{base} and H_{ref}. The *alternate-greedy* R_{GGP} ($R_{GGP}^{\boxed{AG}}$) for picking out each $H_{M(k)}$ is achieved by two steps. The first step is to select one parent as the target, called H_T. Here it is achieved by selecting one of them alternately [29]. The second step is to pick out a group in H_T as the $H_{M(k)}$. Here the element $H_{T(x)}$ with maximal size of $|(H_{M(1)} \cup ... \cup H_{M(k-1)}) \cup H_{T(x)}|$ is picked out as the $H_{M(k)}$ [29].

$R_{GGP}^{\boxed{AG}}$ aims at achieving a H_M with two features: a) the H_M has an enough distance from the both parent group sets H_{base} and H_{ref}, thus it allows adaptive leaps into new local valleys, which has been used by algorithms [51] in exploring the "big valley" [5, 57, 70] in a rugged landscape; and b) the H_M has a large V_{HD} size. If both parents are stable group sets, H_M is also a stable H, thus the V_{CC} size of H_M is also large.

Vertices-Removing. The R_{GVR} achieves a simplex H, called H_{inc}, by removing all redundant vertices in the H_M, where H_{inc} is a subset of H_M and has a same V_{HD} size as H_M. For each vertex existing in multiple groups, the *first-keeping* R_{GVR} ($R_{GVR}^{\boxed{KF}}$) only keeps the vertex in the $H_{M(k)}$ with the smallest k value [29]; while the *random-keeping* R_{GVR} ($R_{GVR}^{\boxed{KR}}$) keeps the vertex in a group selected at random.

It is rational that H_{inc} has a large V_{CC} size, thus H_{inc} will benefit given H_M has a large V_{CC} size. If H_M is a stable H, then every potential H_{inc} has the same V_{CC} size.

The Instances. The grouping-based method itself and its parts have been studied in recent years, especially the greedy partition crossover (GPX) [29] and its variants. For example, GPX can be represented as < $R_{GPP}^{\boxed{E}}$, $R_{GGP}^{\boxed{AG}}$, $R_{GVR}^{\boxed{KF}}$, $R_{VA}^{\boxed{R}}$ >. Moreover, $R_{GPP}^{\boxed{MIS}}$ has been applied to the independent sets in the adaptive memory [31], and both $R_{GPP}^{\boxed{IS}}$ and $R_{VA}^{\boxed{MC}}$ have been considered in another generalized version [37].

The standard version of grouping-based recombination ($R_{XS:G}^{[STD]}$) is defined as < $R_{GPP}^{[MIS]}$, $R_{GGP}^{[AG]}$, $R_{GVR}^{[KR]}$, $R_{VA}^{[MC]}$ >, which differs from the previous methods in at least two parts and the part $R_{GVR}^{[KR]}$ is a novel one. Together with $R_{GPP}^{[MIS]}$, $R_{GGP}^{[AG]}$ and $R_{GVR}^{[KR]}$ work on stable group sets to generate a simplex group set with a large c-core size in an unbiased way.

4.4 Standard MAFS Version

Formally, the standard version of MAFS, called #STD, can be represented as a tuple, i.e., <IR, T_{MAX}, N_P, $R_{SP}^{[R]}$, $R_{XS:G}^{[STD]} + R_{LS}^{[QT]}$, $R_T^{[D]}$, $R_{INI}^{[DS]}$, $G_{(N_P,N_L)}$ >. Other MAFS versions are then defined by applying the corresponding modification(s) to #STD.

$G_{(N_P,N_L)}$ is defined to describe a static IP_{NET} model: each node has N_L nodes connecting to it, where the N_L nodes are randomly selected at t=0 and all the connections are directed and static in a run. In the case that $N_L \equiv N_P$, because the topology is fully-connected, $G_{(N_P,N_L)}$ is equivalent to the centralized memory [68].

The *randomized* R_{SP} rule ($R_{SP}^{[R]}$) picks out a state from $\underline{X}_{SE}^{(t)}$ at random. The *directly* R_T rule ($R_T^{[D]}$) replaces $\vec{s}_A^{(t)}$ by $\vec{s}_G^{(t)}$ directly, thus the $\vec{s}_A^{(t)}$ in M_A of an agent is the most recently state generated by the agent itself.

The $R_{INI}^{[DS]}$ rule constructs the total N_P states by three steps: a) to construct an template assignment [29], called $\hat{s}_T$, corresponding to the first K color classes found by *Dsatur* [8]; b) to generate each $\vec{s}$ with $R_{VR}^{[R]}$ base on the $\hat{s}_T$ [29]; and c) to improve each generated $\vec{s}$ immediately with $R_{LS}^{[QT]}$.

By default, the parameters are N_P=25 and L_I=50 for the $R_{CCL}^{[I]}$ of $R_{LS}^{[QT]}$, respectively. T_{MAX} is fixed as 500. In addition, there is $N_L \equiv N_P$ for the $G_{(N_P,N_L)}$ by default.

5. Experimental Results and Discussions

The characteristics of MAFS are investigated by the experiments on the hard graphs.

There are two main indices for measuring the performance of an algorithm. The first is the solution quality, which is the probability to find one solution for given K, called p_s ($p_s \in [0,1]_{\mathbb{R}}$). The larger is the p_s, the better the performance. The p_s can be estimated with N_S/N_R, where N_S is the number of satisfied trials that achieve proper coloring, and N_R is the number of trials. All average results are evaluated with the satisfied trials. The second index is the computational cost. In the comparison of the algorithms across different platforms and reducing unnecessarily impacts caused by the low-level details, it is often preferable to use *representative operation counts*, or called *run-length*, is preferred to be used for reducing unnecessarily impacts caused by the low-level details rather than the CPU time, as the first is a more platform-independent measure of the computational cost of an algorithm [40]. R_{LS} and R_{XS} both execute major computations in multiple cycles. In this paper, T_m ($\times 10^6$) is the count of 1-*moves* and $N_X = T_R \cdot N_P$ is the count of R_{XS} operations, where T_R is the cycles taken to reach the last improvement. The smaller are the T_m and N_X, the better the performance. In solving GCP, the run-length is characterized by T_m because 1-*moves*

consume much more computational time than R_{XS} operations, where T_m is huge. Therefore, the performance indices can be simplified as a tuple, < p_s, T_m>.

5.1 Basic Performance

The random graphs with a density d=0.5 is a traditional class of benchmark instances. Table 1 lists the mean results of 10 trials by #STD for both the 100-node (gcol01-gcol20) and the 300-node (gcol21-gcol30) instances, which are available from the OR-Library (http://people.brunel.ac.uk/~mastjjb/jeb/orlib/colourinfo.html). For each instance, p_s, T_m and T_R are reported for the color K. The value $\bar{K}$ denotes the smallest number of colors needed for which each instance can be colored without a failure. The results show that all the 100-node instances can be solved with short T_R values.

Table 1. Mean results on random graph instances with d=0.5 by #STD of MAFS.

Graph	K	p_s	T_m	T_R	Graph	K	p_s	T_m	T_R	Graph	$\bar{K}$	K	p_s	T_m	T_R
gcol01	15	1.00	0.011	0.6	gcol11	15	1.00	0.009	0.1	gcol21	33	32	0.70	3.867	186.6
gcol02	15	1.00	0.011	0.8	gcol12	15	1.00	0.017	2.1	gcol22	33	33	1.00	2.018	72.0
gcol03	15	1.00	0.020	2.7	gcol13	15	1.00	0.011	0.1	gcol23	33	32	0.70	2.611	104.7
gcol04	15	1.00	0.017	1.4	gcol14	15	1.00	0.038	7.8	gcol24	33	32	0.20	3.896	160.5
gcol05	15	1.00	0.011	0.0	gcol15	15	1.00	0.022	3.6	gcol25	32	32	1.00	3.858	143.7
gcol06	15	1.00	0.017	2.2	gcol16	15	1.00	0.012	0.4	gcol26	33	32	0.10	1.275	84.0
gcol07	15	1.00	0.014	0.5	gcol17	15	1.00	0.026	5.2	gcol27	32	32	1.00	3.087	126.5
gcol08	15	1.00	0.012	0.2	gcol18	15	1.00	0.012	0.5	gcol28	33	32	0.10	4.613	225.0
gcol09	15	1.00	0.009	0.1	gcol19	15	1.00	0.031	5.3	gcol29	33	32	0.10	2.873	130.0
gcol10	15	1.00	0.018	2.4	gcol20	14	1.00	0.028	6.6	gcol30	33	33	1.00	2.030	69.7

Table 2. Average results on random graphs with d=0.5 by *Tabucol* [38], GLS [24] and #STD.

$\|V\|$	Number of Graphs	$\tilde{\chi}$	K_M from [38]	K_M from [24]	$\bar{K}$ from [24]	K_M	$\bar{K}$	K
100	20	16	16	15	14.95	15	14.95	14.95
300	10	35	35	34	33.5	**33**	**32.8**	**32.2**

Table 2 summarizes the average results reported by *Tabucol* [38], by the genetic algorithm hybridized with a local search (GLS) [24], as well as those obtained by #STD, where $\tilde{\chi}$ is a probabilistic estimation of the chromatic number of a group of graphs [24]. The value K_M denotes the smallest number of colors for which all graphs of the same $|V|$ can be colored with p_s=100%. Both the algorithm from [24] and #STD find smaller K_M than *Tabucol* for the 100-node graphs. Moreover, for the 300-node graphs, #STD achieves better results than the both previous algorithms [24, 38] for both K_M and $\bar{K}$, while the average K is even smaller than the average $\bar{K}$.

For further demonstration of the performance of MAFS, totally 20 representative challenging instances are selected from a mixed set of both DIMACS [42] and COLOR04 (http://mat.gsia.cmu.edu/COLOR04/) graph instances, where C2000.5 is a large graph from the clique part of the DIMACS Challenge. Some easy graphs are excluded, such as: a) the graphs that can be reduced into trivially [10, 11], such as

games120, Book (5 graphs), Miles (5 graphs), and MIZ graphs (4 graphs); or b) the graphs that can be solved efficiently by simple heuristics including *DSatur* [8] and *XRLF* [41], such as MYC (5 graphs), REG (14 graphs), CAR, and most Queen Graphs, which may due to 1-*perfect* [13] or the distribution in node degrees [65].

Table 3. Results on challenging graph instances by #STD and some existing algorithms.

Graph	$\|V\|$	d	K^*	K	p_s	T_m	T_R	*DS*	*XR*	*TC*	*IG*	*SI*	*MIPS*	*ILS*	*AMA*	*IA*	*ABA*
abb313GPIA	1557	0.04	9	**9**	1.00	19.3	22.0	11	12					**9**	11		**9**
ash958GPIA	1916	0.01	4	**4**	1.00	6.35	5.90	6	5					**4**	6		**4**
C2000.5	2000	0.50	162	***150***	0.60	70.9	410				188	165	**162**				
DSJC125.5	125	0.50	17	**17**	1.00	0.29	51.2	21	18		18	**17**	**17**	**17**	**17**	18	**17**
DSJC250.5	250	0.50	28	**28**	1.00	2.00	110	38	29	**28**	32	**28**	**28**	**28**	**28**	**28**	29
DSJC500.5	500	0.50	48	**48**	0.80	7.70	173	67	50	49	57	49	49	50	**48**		50
DSJC1000.5	1000	0.50	83	84	0.90	31.8	296	114	86	89	102	89	88	90	84		91
DSJC1000.9	1000	0.90	224	***223***	0.40	17.8	285	297	232				228	227	**224**		229
DSJR500.1c	500	0.97	85	**85**	1.00	1.77	8.70	87	91		**85**	**85**	**85**		86		**85**
flat300_26_0	300	0.48	26	**26**	1.00	0.10	20.2	41	33		36	**26**	**26**	**26**	**26**	27	**26**
flat300_28_0	300	0.48	31	**31**	1.00	2.49	103	41	33	32	35	**31**	**31**	**31**	**31**	32	**31**
flat1000_50_0	1000	0.49	50	**50**	1.00	0.86	53.5	112	84		**50**	**50**	**50**	88	**50**		**50**
flat1000_60_0	1000	0.49	60	**60**	0.50	1.97	222	113	87		100	**60**	**60**	89	**60**		**60**
flat1000_76_0	1000	0.49	83	**83**	0.90	27.0	280	114	87	87	102	89	87	89	84		84
latin_square_10	900	0.76	98	104	0.20	109	235	126	117		105	**98**	99	103	104		100
le450_15c	450	0.17	15	**15**	1.00	0.13	7.10	24	19	16	25	**15**	**15**	**15**	**15**	**15**	**15**
le450_25c	450	0.17	26	27	1.00	0.52	0.40	29	27	**26**			**26**	**26**	**26**		**26**
qg.order100	10000	0.02	100	**100**	1.00	0.49	0.00	103	**100**					**100**	**100**		
queen16_16	6320	0.19	17	18	1.00	0.03	0.00	21	**17**					18	**17**		18
school1_nsh	352	0.31	14	**14**	1.00	0.08	0.20	15	19		**14**	20	**14**		**14**	15	**14**

Table 3 summaries the results on the challenging graph instances by #STD of MAFS and existing algorithms. For each graph, $|V|$ is the number of vertices, d is the density, and K^* is the best-known color size. For #STD, N_R=10 trials are run, then the mean results of p_s, T_m, and T_R are reported for the K. It also summaries the best color sizes achieved by some existing algorithms, including *DSatur* (DS) [8] tested in [31], *XRLF* (*XR*) [41] tested in [11], *Tabucol* (TC) [38] tested in [29], iterated greedy algorithm (*IG*) [14], S-IMPASSE (*SI*) [53], iterated local search (*ILS*) [11], minimal-state processing search (*MIPS*) [27], adaptive memory algorithm (*AMA*) [31], immune algorithm (*IA*) [16], and ant-based algorithm (*ABA*) [9]. Bold face indicates that the color size is not worse than K^*. It shows that #STD is competitive to the state-of-the-art algorithms in achieving K^*. It is impressive that #STD obtains new K results for two large graphs in high densities, i.e., C2000.5 and DSJC1000.9. It also shows that qg.order100 can be solved only by the stable LS in the stage of initialization. For latin_square_10, MAFS is not very efficient, which may due to the additional symmetry that all the vertices are in the same degree.

For all the following experiments, N_R=100 trials are run for each case so as to achieve more reliable statistics of the performance indices. In consideration of the limited available computational resources, we will focus on a small subset of the challenging graph instances, which includes: a) four random graphs, DSJC250.5,

DSJC500.5, DSJC1000.5, and DSJC1000.9; b) two flat graphs, flat300_28_0 and flat1000_76_0; and c) two structural Leighton graphs, le450_15c and le450_25c.

Table 4. The mean results achieved by HCA [29] and #STD.

Graph	K_S	HCA [29]					#STD				
		L_C	N_R	p_s	T_m	T_R	p_s	T_m	T_R	$r_{q/p}$	$\Delta\tilde{p}_S$
DSJC250.5	28	2000	10	0.90	0.49	235	0.90	1.87	102	0.026	-0.0998
DSJC500.5	48	5600	10	0.50	4.90	865	0.75	9.64	210	0.019	0.0056
DSJC1000.5	84	16000	5	0.60	20.7	1283	0.94	27.5	271	0.017	0.2357
flat300_28_0	31	2000	10	0.60	0.64	790	1.00	3.31	123	0.022	0.0086
flat1000_76_0	83	16000	5	0.80	17.5	1008	0.93	28.5	286	0.017	0.0028
le450_15c	15	5600	10	0.60	0.19	24	1.00	0.14	7.91	0.172	0.5193
le450_25c	27	4000	10	1.00	0.09	18	1.00	0.49	0.31	0.003	-

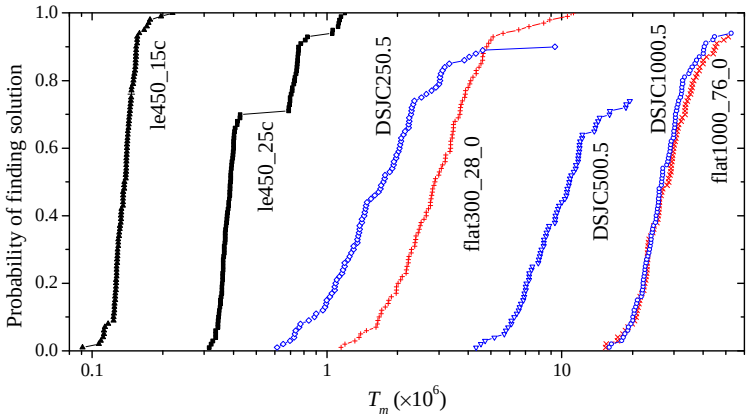

Figure 2. Run-length distribution (RLD) for #STD on the graphs.

In Table 4, #STD and the hybrid coloring algorithm (HCA) [29] are compared on the K_S color series (K=84 for DSJC1000.5 and K=27 for le450_25c). HCA [29] maintains a state set supporting a FS, i.e., GPX+*Tabucol*, which performs the FS only once at each cycle. In addition, HCA can be considered as the anterior version of AMA [31] (cf. Table 3). In HCA, the size of the state set is fixed as 10, and the LS chain length (L_C) values of *Tabucol* have to be adjusted for different instances. We can use $\Delta\tilde{p}_S = p_S - \tilde{p}_S^*$, where $\tilde{p}_S^* = 1-(1-p_S^*)^{T_m/T_m^*}$, to achieve an approximate comparison between the algorithm in $< p_S, T_m >$ and the reference algorithm in $< p_S^*, T_m^* >$, under the condition that $p_S^* < 1$. In Table 3, HCA is chosen as the reference algorithm, and the results of the seven graphs indicate that #STD achieves positive $\Delta\tilde{p}_S$ values over HCA on all the graphs except for DSJC250.5 and le450_25c. The better performance of HCA on DSJC250.5 and le450_25c might be due to the usage of *Tabucol* [38]. Both DSJC250.5 and le450_25c can be solved by *Tabucol*, and *Tabucol* is much more efficient than HCA in solving le450_25c [29].

Figure 2 gives the empirical run-length distribution (RLD) for the GCP instances solved by #STD, where RLD provides adequate information to describe the behavior

of an algorithm [40]. It can be seen that the steepness in every case is quite well, where above 50% trials have achieved the optimum in a T_m within one order of magnitude. Moreover, abrupt changes and heavy tails [36] are found in some cases, such as le450_25c and DSJC250.5, which may result from the stagnation in certain large benches or local minima. Such abrupt slowdowns appear in the later search stage, which may be improved through running MAFS multiple times [36, 40].

In Table 5, #STD.L and the GPB algorithm [34] are compared on the K* color series. Here #STD.L is defined as #STD with different N_P and L_I values, where each parameter is set as a value that is not larger than that of GPB and is not less than that of #STD. GPB [34] is a generational genetic algorithm manipulating a FS strategy, i.e., GPX+VDS. For GPB, only N_R=3 trials were run for each graph instance. In order to evaluate each performance index of an algorithm over multiple instances, we define a r_v value for each performance index of the algorithm as follows: a) choose the reference algorithm; b) compute the ratio of each performance index between the algorithm and the reference algorithm for each instance; and c) calculate the geometric mean value of all the ratios over all the instances. For p_s, r_v>1 is preferable; for T_m or N_X, r_v<1 is preferable. The results in Table 5 show that #STD.L produces a dominating performance over GPB. By taking GPB as the reference algorithm, the r_v values of #STD.L are r_v=1.045 for p_s, r_v=0.227 for T_m, and r_v=0.212 for N_X.

Table 5. The mean results achieved by GPB [34] and #STD.L.

Graph	K^*	GPB [34]					#STD.L				
		N_P	L_I	p_s	T_m	T_R	N_P	L_I	p_s	T_m	T_R
DSJC250.5	28	100	100	1.00	11.7	118	100	100	1.00	6.73	81.8
DSJC500.5	48	100	500	1.00	485	686	100	500	1.00	122	172
DSJC1000.5	83	500	100	0.33	690	239	500	50	0.45	508	244
flat300_28_0	31	100	100	1.00	52.7	435	25	100	1.00	4.28	117
flat1000_76_0	83	100	200	1.00	177	305	100	50	1.00	83.2	195
le450_15c	15	100	100	1.00	1.9	11	25	100	1.00	0.16	7.74
le450_25c	26	100	500	1.00	2341	1571	25	500	1.00	211	89.1

The standard MAFS version has two main parameters, N_P and L_I. The larger is the N_P, the more agents the system has. The larger is the L_I for a stable R_{LS}, the more powerful capability in finding the exits from benches the strategy has. By comparing #STD.L and #STD, usage of large N_P and/or L_I has two implications for MAFS, which, in one hand, leads to a better solution quality, as demonstrated on DSJC1000.5 and le450_25c where the K^* is achieved, but in another hand, makes MFAS require more computational cost, as shown by the other instances.

The performance of MAFS may be further enhanced through employing one of more advanced strategies in its R_{LS} that not only explores other local valleys but also increases the diversity of the newly generated information [29], such as *Tabucol* [38], ERA [47], Neural Network [17], Extremal Optimization [6], etc. In addition, the knowledge components of MAFS may be further improved by utilizing certain structural information and the related knowledge, if necessarily. For example, R_T may use the quality information in a landscape [67], and may also combine it with certain

auxiliary methods, such as a Boltzmann acceptance criteria in Simulated Annealing [41]. Moreover, both R_{FS} and R_{SP} may be turned to be more intelligent by utilizing certain population information of $\underline{X}_{SE}$, such as Kullback entropy [16].

5.2 Search Characteristics

Although a stable R_{LS} cannot tackle with any kinds of ruggedness, i.e., local minima in a landscape, it can lead to better states by finding the exits from benches [26], where each bench is a plateau but not a local minimum in a landscape. Hence when a stable R_{LS} is used in MAFS, such as VDS or $R_{LS}^{\boxed{QT}}$, the neutrality in a landscape is mainly exploited by the stable R_{LS} rule, while the ruggedness is mainly explored by the R_{XS} rule under the management of the multiagent framework.

Table 6 lists the results of the MAFS versions with different components in R_{FS}, which are applied to the K_S color series. The version #LS.VD is defined by using the VDS as the R_{LS}. Then three MAFS versions are realized, where each uses a different component for $R_{XS:G}^{\boxed{STD}}$: a) #GPP.E, which uses $R_{GPP}^{\boxed{E}}$ for the R_{GPP}; b) #GPP.IS, which uses $R_{GPP}^{\boxed{IS}}$ for the R_{GPP}; and c) #GVR.KF, which uses $R_{GVR}^{\boxed{KF}}$ for the R_{GVR}. Moreover, the reference algorithm for calculating the r_v values is #STD (cf. Table 4).

Table 6. The mean results by MAFS versions in different LS and XS rules.

Graph	K_S	#LS.VD			#GPP.E		#GPP.IS		#GVR.KF	
		p_s	T_m	$r_{g/p}$	p_s	T_m	p_s	T_m	p_s	T_m
DSJC250.5	28	0.82	3.82	0.010	0.68	3.70	0.97	2.20	0.78	2.59
DSJC500.5	48	0.55	15.9	0.009	0.34	14.9	0.88	12.3	0.31	7.64
DSJC1000.5	84	0.56	46.1	0.008	0.35	38.5	0.96	33.4	0.25	21.2
flat300_28_0	31	0.98	5.38	0.010	0.75	6.49	1.00	4.13	0.93	2.98
flat1000_76_0	83	0.51	41.8	0.009	0.22	37.7	0.99	33.6	0.20	23.1
le450_15c	15	1.00	0.52	0.038	0.89	0.22	1.00	0.14	0.95	0.13
le450_25c	27	1.00	0.55	0.003	1.00	0.52	1.00	0.60	1.00	0.56
r_v	-	0.80	1.78	0.463	0.57	1.51	1.05	1.19	0.56	0.94

Local Search. The advantage of long LS chains is shown in the results by #STD.L (cf. Table 5) and #STD (cf. Table 4), where #STD.L simply uses a larger L_I value for solving le450_25c, which is able to obtain better K than #STD does. For $R_{LS}^{\boxed{QT}}$, L_I>1 simply means it walks on a plateau at the last round, since it takes stable moves for all the unfixed vertices with its $R_{LSR}^{\boxed{SYS}}$. The advantage of the stable LS using a larger L_I value clearly indicates the significance of the search on plateaus. Moreover, it implies that some benches are quite large, thus the local search hardly finds the exits.

By comparing the results between #LS.VD (cf. Table 6) and #STD (cf. Table 4) which are realized under two stable R_{LS} rules, i.e., VDS and $R_{LS}^{\boxed{QT}}$, respectively, two facts are indicated: a) most 1-*moves* are spent on plateaus, maybe in different levels, according to the quite small $r_{g/p}$ values (<0.05) in Table 4 and 6, where each $r_{g/p}$ gives the ratio of the number of greedy 1-*moves* (Δf <0) over that of plateau 1-*moves* ($\Delta f \equiv 0$); and b) VDS is less efficient in the plateau search than $R_{LS}^{\boxed{QT}}$, since #LS.VD obtains worse p_s and higher T_m by using a larger ratio of plateau 1-*moves* (according

to that r_v=0.463 for $r_{g/p}$). Compared with VDS, $R_{LS}^{\boxed{QT}}$ uses an additional tabu matrix to restrict the input color list at the local level, which may forbid many unpromising plateau moves. For LS strategies, there are three intuitive measures of effectiveness [59]: *depth*, *mobility*, and *coverage*. $R_{LS}^{\boxed{QT}}$ with the tabu matrix may be efficient in leading the search to a better *coverage* than VDS in consideration of that the *depth* and *mobility* of both strategies are same on each plateau, where *coverage* measures how systematically the search explores the entire plateau [59].

Recombination Search. The R_{XS} rule, under the management of the multiagent framework, plays a major role in navigating the search in a rugged landscape, especially when a stable R_{LS} is employed as the low-level strategy in the fusion search. Specifically, for GCP, the grouping-based R_{XS} rule ($R_{XS:G}$) works on group sets.

The MAFS versions #GPP.E, #GPP.IS, and #STD are realized by simply using the different R_{GPP} rules, i.e., $R_{GPP}^{\boxed{E}}$, $R_{GPP}^{\boxed{IS}}$, and $R_{GPP}^{\boxed{MIS}}$, respectively. The results in Table 6 show that #GPP.D achieves a much worse performance while #GPP.IS achieves a similar performance, when they are compared with #STD.

Together with $R_{GPP}^{\boxed{E}}$, $R_{GGP}^{\boxed{AG}}$ simply leads to a H_M in a large V_{HD} size, and then the R_{GVR} rule can achieve a H_{inc} in the same V_{HD} size of the H_M.

Together with either $R_{GPP}^{\boxed{IS}}$ or $R_{GPP}^{\boxed{MIS}}$, $R_{GGP}^{\boxed{AG}}$ leads to a stable H_M in a large c-core size (V_{CC}). Firstly, the c-core of stable group set achieved by either $R_{GPP}^{\boxed{IS}}$ or $R_{GPP}^{\boxed{MIS}}$ is a superset of that by $R_{GPP}^{\boxed{E}}$. Secondly, $R_{GGP}^{\boxed{AG}}$ can achieve a stable H_M with a larger V_{CC} size if the misleading information from the critical vertices on the group sizes is excluded. Then the R_{GVR} rule can achieve a H_{inc} in the same V_{CC} size of the stable H_M.

The $R_{XS:G}$ rule may prefer to obtain a H_M in a large V_{CC} size instead of in a large V_{HD} size, so as to achieve a H_{inc} in a large V_{CC} size. Moreover, each parent group set associated with a high-quality state is preferable to have a large V_{CC} size.

Some advanced methods for improving independent sets, such as those studied in the *column generation approach* [50] and *lmXRLF* [45], may also be used to enhance the c-core size of a stable group set by applying to certain groups in the group set.

The MAFS versions #GVR.KF and #STD are realized by using the different R_{GVR} rules, i.e., $R_{GVR}^{\boxed{KF}}$ and $R_{GVR}^{\boxed{KR}}$, respectively. The results in Table 6 indicate that using of $R_{GVR}^{\boxed{KR}}$ can achieve a better performance than $R_{GVR}^{\boxed{KF}}$. Removing a few vertices from a group allows the group to be evolved locally through adding the new vertices performed by the R_{VA} rule. Although each group extracted from a high-quality state may be rather stable, a large group is not necessarily a superset of certain proper color class. Hence, the advantage of using the unbiased $R_{GVR}^{\boxed{KR}}$ might result from that $R_{GVR}^{\boxed{KR}}$ allows all the related groups to be evolved, even the large groups.

Decentralized Interaction. Table 7 summaries the results on the K_S color series by four MAFS versions with a decentralized $G_{(N_P,N_L)}$ model at different density values, where N_L/N_P values are 0.2, 0.4, 0.6, and 0.8, respectively. A larger N_L value means both a faster diffusion of the local information and a slower diffusion of the global information. Here the reference algorithm in calculating the r_v values is #STD, where N_L/N_P is 1.0. Table 7 shows that the r_v values of p_s and T_m for each case are closed to 1 when N_L/N_P varies from 1.0 to 0.2, meaning the performances of these MAFS

versions are quite robust, which may be due to the good balance between the diffusions of the local and global information in the networks.

Table 7. The results by MAFS versions for IP_{NET} in different N_L/N_P values (from 0.2 to 0.8).

Graph	K_S	N_L/N_P=0.2		N_L/N_P=0.4		N_L/N_P=0.6		N_L/N_P=0.8	
		p_s	T_m	p_s	T_m	p_s	T_m	p_s	T_m
DSJC250.5	28	0.96	1.84	0.91	2.05	0.87	2.15	0.94	1.88
DSJC500.5	48	0.65	11.3	0.80	9.18	0.76	10.8	0.76	10.6
DSJC1000.5	84	0.93	27.5	0.94	26.7	0.93	28.2	0.93	28.0
flat300_28_0	31	0.99	3.32	1.00	3.00	1.00	3.31	1.00	3.21
flat1000_76_0	83	0.94	30.4	0.92	28.7	0.92	26.9	0.94	25.7
le450_15c	15	0.96	0.14	0.97	0.14	0.98	0.14	0.99	0.14
le450_25c	27	1.00	0.50	1.00	0.55	1.00	0.48	1.00	0.52
r_v	-	0.98	1.03	1.00	1.01	0.99	1.03	1.01	1.01

As a multiagent model, MAFS may have an advantage in the robust parallel computing, as agents may be designed to locate at different processors in a network. This may be useful in practice since solving large graphs is quite time-consuming.

Firstly, MAFS stores declarative knowledge by the M_A of agents and distributed the knowledge to the nodes, which make the algorithm very robust because there is not any key node(s) in this case, meaning the nodes will not fail occasionally during a run.. The centralized memory is a typical example of a key node, which is required by many frameworks, such as Immune Algorithms [16], Scatter Search [37], Genetic Algorithms [24, 34, 54], and Adaptive Memory Programming [31], etc.

Secondly, the communication of each agent is simply spent on accessing one (or a few) state(s) memorized by other nodes during each learning cycle. Hence, the communication cost may be neglected in comparison with the computation cost.

Thirdly, MAFS may still work well even if the interaction network is sparse, as indicated in Table 7, MAFS possesses a robust performance in the topologies with varied N_L/N_P values. In a real world, the mode of networks is often partially connected, i.e., some connections may be invalid under certain physical network conditions.

Of course, in order to make MAFS support a communication between the agents in the real world, more efforts must be employed on the physical infrastructure. In addition, the current IP_{NET} model is quite artificial, thus more experiments should be performed to evaluate its robustness. Another two interesting issues are to analyze the cooperative solving features of MAFS under various topologies [12], and to study if some of the features may boost the performance. Moreover, non-uniform models [65] such as *small-world*, *ultrametric*, *power-law* models, etc., including certain dynamic topologies, may be worthy further consideration.

PAC Property. MAFS is not necessarily probabilistically approximately complete (PAC) [39], according to the law of bounded rationality [33] it follows.

However, MAFS can achieve PAC in a simple way. Firstly, MAFS is PAC if the R_{FS} of any agent is PAC. Secondly, the R_{FS} rule is PAC if: a) its R_{LS} rule is PAC; or b) its $R_{XS:G}$ rule is PAC and its R_{LS} rule preserves the best state ever found. Thirdly, the PAC may be also achieved by some meta R_{LS} rules. For Example, a meta R_{LS} rule is PAC if it is chained by both a R_{LS} rule in PAC and a stable R_{LS} rule.

In fact, to find a R_{LS} rule in PAC is no difficult. For example, it is easy to prove that the random walk strategy ($R_{LS}^{\boxed{RW}}$), i.e., < $R_{LSL}^{\boxed{RW}}$, $R_{LSR}^{\boxed{P}}$ >, is PAC.

At each round, the probability of determining whether a vertex is moved by $R_{LSL}^{\boxed{RW}}$ is $V_{RW} / |\underline{V}|$, and the probability of assigning the vertex with each color by $R_{LSR}^{\boxed{P}}$ is 1/*K*. Hence the probability for assigning each vertex with each color is $V_{RW} /(|\underline{V}| \cdot K)$. For any incumbent state, the probability of achieving a proper coloring is $(V_{RW} /(|\underline{V}| \cdot K))^{|\underline{V}|}$, which is larger than zero since $V_{RW}>0$ and both $|\underline{V}|$ and *K* are finite values. Hence $R_{LS}^{\boxed{RW}}$ is PAC if $t \to \infty$ according to Theorem 2 in Ref. [39], i.e., *any algorithm which, for any incumbent state, executes a random walk with a probability of at least larger than 0 at any given time, is PAC.*

6. Conclusions

In this paper, a multiagent fusion search (MAFS) is presented as a realization of a multiagent optimization framework to solve the graph coloring problem (GCP). A fusion search includes a recombination search (XS) working in a navigation role and a local search (LS) in an exploitation role. In MAFS, each of agents performs the fusion search with extremely limited knowledge in its personal declarative memory and cooperates with others through a decentralized interaction protocol in the environment, thus the agents are able not only to explore in parallel but also to achieve a collective performance under the law of socially biased individual learning.

Compared with some state-of-the-art coloring algorithms, MAFS is competitive in both the solution quality and the computational cost when applied to some hard graphs. In addition, MAFS improves the best known results of two large graphs.

In addition, we have investigated the search characteristics of the components of MAFS. The experimental results show that the Quasi-Tabu LS and grouping-based XS strategies are especially useful for tackling with neutrality and ruggedness in the GCP landscape. A simple analysis indicates that MAFS can achieve probabilistically approximately complete in an easy way. The potential advantage of the decentralized interaction protocol in a robust parallel computing is discussed as well.

Future research is suggested to: a) achieve a scalable performance by the agents with adaptive strategies; b) explore certain cooperative problem solving features by investigating the performances with real-world interaction protocols; and c) study MAFS with suitable components to solve other hard computational problems.

References

1. Anderson, J.R.: Human symbol manipulation within an integrated cognitive architecture. Cognitive Science 29(3): 313-341 (2005)
2. Bandura, A.: Social Learning Theory. Prentice Hall, Englewood Cliffs, NJ (1977)
3. Barbosa, V.C., Assis, C.A.G., do Nascimento, J.O.: Two novel evolutionary formulations of the graph coloring problem. Journal of Combinatorial Optimization 8(1): 41-63 (2004)
4. Barnier, N., Brisset, P.: Graph coloring for air traffic flow management. Annals of Operations Research 130(1-4): 163-178 (2004)

5. Boese, K.D., Kahng, A.B., Muddu, S.: A new adaptive multi-start technique for combinatorial global optimizations. Operation Research Letters 16: 101-113 (1994)
6. Boettcher, S., Percus, A.G.: Extremal optimization at the phase transition of the three-coloring problem. Physical Review E 69(6): Art. 066703 (2004)
7. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, UK (1999)
8. Brélaz, D.: New methods to color the vertices of a graph. Communications of the ACM 22(4): 251-256 (1979)
9. Bui, T.N., Nguyen, T.H., Patel, C.M., Phan, K.-A.T.: An ant-based algorithm for coloring graphs. Discrete Applied Mathematics 156(2): 190-200 (2008)
10. Cheeseman, P., Kanefsky, B., Taylor, W.M.: Where the really hard problems are. International Joint Conference on Artificial Intelligence, San Mateo, CA, 331-337 (1991)
11. Chiarandini, M.: Stochastic Local Search Methods for Highly Constrained Combinatorial Optimisation Problems. Ph.D. thesis, Darmstadt University of Technology, Germany (2005)
12. Cioffi-Revilla, C.: Invariance and universality in social agent-based simulations. PNAS 99(suppl. 3): 7314-7316 (2002)
13. Coudert, O.: Exact coloring of real-life graphs is easy. Design Automation Conference, San Francico, California, USA, 121-126 (1997)
14. Culberson, J.C., Luo, F.: Exploring the k-colorable landscape with iterated greedy. *In* [42]: 245-284 (1996)
15. Curran, D., O'Riordan, C.: Increasing population diversity through cultural learning. Adaptive Behavior 14(4): 315-338 (2006)
16. Cutello, V., Nicosia, G., Pavone, M.: An immune algorithm with stochastic aging and kullback entropy for the chromatic number problem. Journal of Combinatorial Optimization 14(1): 9-33 (2007)
17. Di Blas, A., Jagota, A., Hughey, R.: Energy function-based approaches to graph coloring. IEEE Transactions on Neural Networks 13(1): 81-91 (2002)
18. Dietterich, T.G.: Learning at the knowledge level. Machine Learning 1: 287-316 (1986)
19. Dorne, R., Hao, J.K.: A new genetic local search algorithm for graph coloring. International Conference on Parallel Problem Solving from Nature, Amsterdam, NL, 745-754 (1998)
20. Edgington, T., Choi, B., Henson, K., Raghu, T.S., Vinze, A.: Adopting ontology to facilitate knowledge sharing. Communications of the ACM 47(11): 85-90 (2004)
21. Eppstein, D.: Small maximal independent sets and faster exact graph coloring. Journal of Graph Algorithms and Applications 7(2): 131-140 (2003)
22. Erben, W.: Grouping genetic algorithm for graph colouring and exam timetabling. International Conference on Practice and Theory of Automated Timetabling, Konstanz, Germany, 132-156 (2000)
23. Falkenauer, E.: A hybrid grouping genetic algorithm for bin packing. Journal of Heuristics 2(1): 5-30 (1996)
24. Fleurent, C., Ferland, J.A.: Genetic and hybrid algorithms for graph coloring. Annals of Operations Research 63: 437-464 (1996)
25. Fragaszy, D., Visalberghi, E.: Socially biased learning in monkeys. Learning & Behavior 32(1): 24-35 (2004)
26. Frank, J., Cheeseman, P., Stutz, J.: When gravity fails: local search topology. Journal of Artificial Intelligence Research 7: 249-281 (1997)
27. Funabiki, N., Higashino, T.: A minimal-state processing search algorithm for graph coloring problems. IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences E83A(7): 1420-1430 (2000)
28. Galef, B.G.: Why behaviour patterns that animals learn socially are locally adaptive. Animal Behaviour 49(5): 1325-1334 (1995)
29. Galinier, P., Hao, J.-K.: Hybrid evolutionary algorithms for graph coloring. Journal of Combinatorial Optimization 3(4): 379-397 (1999)

30. Galinier, P., Hertz, A.: A survey of local search methods for graph coloring. Computers & Operations Research 33(9): 2547-2562 (2006)
31. Galinier, P., Hertz, A., Zufferey, N.: An adaptive memory algorithm for the k-colouring problem. Discrete Applied Mathematics 156(2): 267-279 (2008)
32. Gebremedhin, A.H., Manne, F., Pothen, A.: What color is your Jacobian? Graph coloring for computing derivatives. SIAM Review 47(4): 629-705 (2005)
33. Gigerenzer, G., Goldstein, D.G.: Reasoning the fast and frugal way: models of bounded rationality. Psychological Review 103(4): 650-669 (1996)
34. Glass, C.A., Prugel-Bennett, A.: Genetic algorithm for graph coloring: exploration of Galinier and Hao's algorithm. Journal of Combinatorial Optimization 7(3): 229-236 (2003)
35. Glenberg, A.M.: What memory is for. Behavioral and Brain Sciences 20(1): 1-55 (1997)
36. Gomes, C.P., Selman, B.: Algorithm portfolios. Artificial Intelligence 126(1-2): 43-62 (2001)
37. Hamiez, J.-P., Hao, J.-K.: Scatter search for graph coloring. International Conference on Artificial Evolution, Le Creusot, France, 168-179 (2001)
38. Hertz, A., de Werra, D.: Using tabu search techniques for graph coloring. Computing 39: 345-351 (1987)
39. Hoos, H.H.: On the run-time behaviour of stochastic local search algorithms for SAT. National Conference on Artificial Intelligence, Orlando, FL, 661-666 (1999)
40. Hoos, H.H., Stützle, T.: Evaluating Las Vegas algorithms - pitfalls and remedies. Conference on Uncertainty in Artificial Intelligence, Madison, WI, 238-245 (1998)
41. Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C.: Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning. Operations Research 39(3): 378-406 (1991)
42. Johnson, D.S., Trick, M.A., editors. Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge. American Mathematical Society, Providence, RI (1996)
43. Joslin, D.E., Clements, D.P.: "Squeaky wheel" optimization. Journal of Artificial Intelligence Research 10: 353-373 (1999)
44. Khanna, S., Linial, N., Safra, S.: On the hardness of approximating the chromatic number. Combinatorica 20(3): 393-415 (2000)
45. Kirovski, D.: Efficient coloring of a large spectrum of graphs. Design Automation Conference, San Francico, California, USA, 427-432 (1998)
46. Lerman, K., Galstyan, A.: Agent memory and adaptation in multi-agent systems. International Conference on Autonomous Agents and Multi-Agent Systems, Melbourne, Australia, 797-803 (2003)
47. Liu, J., Han, J., Tang, Y.Y.: Multi-agent oriented constraint satisfaction. Artificial Intelligence 136(1): 101-144 (2002)
48. Liu, J., Jin, X., Tsui, K.-C.: Autonomy Oriented Computing (AOC): From Problem Solving to Complex Systems Modeling. Kluwer Academic Publishers, Boston, MA (2005)
49. Liu, J., Tsui, K.-C.: Toward nature-inspired computing. Communications of the ACM 49(10): 59-64 (2006)
50. Mehrotra, A., Trick, M.: A column generation approach for graph coloring. INFORMS Journal on Computing 8(4): 344-354 (1996)
51. Merz, P., Freisleben, B.: Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. IEEE Transactions on Evolutionary Computation 4(4): 337-352 (2000)
52. Mezard, M., Palassini, M., Rivoire, O.: Landscape of solutions in constraint satisfaction problems. Physical Review Letters 95(20): Art. 200202 (2005)
53. Morgenstern, C.: Distributed coloration neighborhood search. *In* [42]: 335-358 (1996)
54. Mumford, C.L.: New order-based crossovers for the graph coloring problem. International Conference on Parallel Problem Solving from Nature, Reykjavik, Iceland, 880-889 (2006)
55. Newell, A., Simon, H.A.: Human Problem Solving. Prentice-Hall, NJ (1972)

56. Nowicki, E.: A fast tabu search algorithm for the permutation flow shop problem. European Journal of Operational Research 91: 160-175 (1996)
57. Reeves, C.R., Yamada, T.: Genetic algorithms, path relinking, and the flowshop sequencing problem. Evolutionary Computation 6(1): 45-60 (1998)
58. Reidys, C.M., Stadler, P.F.: Combinatorial landscapes. SIAM Review 44(1): 3-54 (2002)
59. Schuurmans, D., Southey, F.: Local search characteristics of incomplete SAT procedures. Artificial Intelligence 132(2): 121-150 (2001)
60. Selman, B., Kautz, H.A.: An empirical study of greedy local search for satisfiability testing. National Conference on Artificial Intelligence, Washington DC, USA, 46-51 (1993)
61. Selman, B., Kautz, H.A., Cohen, B.: Noise strategies for improving local search. National Conference on Artificial Intelligence, Seattle, WA, 337-343 (1994)
62. Smith, M.D., Ramsey, N., Holloway, G.: A generalized algorithm for graph-coloring register allocation. ACM SIGPLAN Notices 39(6): 277-288 (2004)
63. Stone, P., Veloso, M.: Multiagent Systems: A survey from a machine learning perspective. Autonomous Robots 8(3): 345-383 (2000)
64. Trick, M.A., Yildiz, H.: A large neighborhood search heuristic for graph coloring. International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Brussels, Belgium, 346-360 (2007)
65. Walsh, T.: Search on high degree graphs. International Joint Conference on Artificial Intelligence, Seattle, Washington, USA, 266-274 (2001)
66. Weyns, D., Holvoet, T.: On the role of environments in multiagent systems. Informatica 29: 409-421 (2005)
67. Xie, X.-F., Liu, J.: A compact multiagent system based on autonomy oriented computing. IEEE/WIC/ACM International Conference on Intelligent Agent Technology, Compiègne, France, 38-44 (2005)
68. Xie, X.-F., Liu, J.: How autonomy oriented computing (AOC) tackles a computationally hard optimization problem. International Joint Conference on Autonomous Agents and Multiagent Systems, Hakodate, Japan, 646-653 (2006)
69. Xie, X.-F., Zhang, W.-J.: SWAF: swarm algorithm framework for numerical optimization. Genetic and Evolutionary Computation Conference, Seattle, WA, 238-250 (2004)
70. Zhang, W.: Configuration landscape analysis and backbone guided local search. Artificial Intelligence 158(1): 1-26 (2004)