Local Evaluation Functions and Global Evaluation Functions for Computational Evolution

Jing Han

SFI WORKING PAPER: 2003-09-048

SFI Working Papers contain accounts of scientific work of the author(s) and do not necessarily represent the views of the Santa Fe Institute. We accept papers intended for publication in peer-reviewed journals or proceedings volumes, but not papers that have already appeared in print. Except for papers by our external faculty, papers must be based on work done at SFI, inspired by an invited visit to or collaboration at SFI, or funded by an SFI grant.

©NOTICE: This working paper is included by permission of the contributing author(s) as a means to ensure timely distribution of the scholarly and technical work on a non-commercial basis. Copyright and all rights therein are maintained by the author(s). It is understood that all persons copying this information will adhere to the terms and constraints invoked by each author's copyright. These works may be reposted only with the explicit permission of the copyright holder.

www.santafe.edu
Local Evaluation Functions and Global Evaluation Functions for Computational Evolution

HAN Jing
The Santa Fe Institute, NM, USA. 87501
Institute of Systems Science, AMSS, China. 100080
hanjing@santafe.edu

Abstract

This paper presents a new look on computational evolution from the aspect of the locality and globality of evaluation functions for solving classical combinatorial problem: the $k$-coloring Problem (decision problem) and the Minimum Coloring Problem (optimization problem). We first review current algorithms and model the coloring problem as a multi-agent system. We then show that the essential difference between traditional algorithms (Local Search, such as Simulated Annealing) and distributed algorithms (such as the Alife&AER model) lies in the evaluation function: Simulated Annealing uses global information to evaluate the whole system state, which is called the Global Evaluation Function (GEF) method; the Alife&AER model uses local information to evaluate the state of a single agent, which is called the Local Evaluation Function (LEF) method. We compare the performances of LEF and GEF methods for solving the $k$-coloring Problems and the Minimum Coloring Problem. The computer experimental results show that the LEF is comparable to GEF methods (Simulated Annealing and Greedy), in many problem instances the LEF beats GEF methods. At the same time, we analyze the relationship between GEF and LEF: consistency and inconsistency. The Consistency Theorem shows that Nash Equilibria of an LEF is identical to local optima of a GEF when the LEF is consistent with the GEF. This theorem partly explains why the LEF can lead the system to a global goal. Some rules for constructing a consistent LEF are proposed. In addition to consistency, we give a picture of the LEF and GEF and show their differences in exploration heuristics.

Key Words

Local Evaluation Function (LEF), Global Evaluation Function (GEF), Combinatorial problem, Coloring problem, Alife&AER Model, Local search, Simulated Annealing, Nash Equilibrium, Local Optimum, Consistency, Computational Evolution

1. INTRODUCTION

The purpose of this paper is to show how people use Local Evaluation Functions and Global Evaluation Functions to direct computational evolution and what the differences between them are. We will focus on the evolution of solution for Combinatorial problems. We limit our discussion on the evolution of multi-agent systems that are composed of homogeneous agents with limited states, the same as the evolution of multi-particle systems in physics, such as spin glass systems [6]. Note that here agents will not evolve their strategies during evolution.

Evolution is not pure random processing. There always exists an explicit or implicit Evaluation Function (EF) which directs the evolution. In different contexts, the evaluation function is also called the objective function, penalty function, fitness function, cost function, energy...
2 Jing HAN

$E$, etc. It has the form $f: \mathbf{S} \in \Omega \rightarrow \mathbb{R}$, where $\Omega$ is the state (configuration) space. The evaluation function calculates how good a state of the whole system (or a single agent) is, then the selection for the next step is based on the evaluation value. Obviously, the evaluation function plays a key role in evolution and it is one of the fundamental problems in evolution, including combinatorial problems.

Generally speaking, the system state related to the minimal evaluation value is the optimum of the system, $S_{opt} = \arg \min f(S)$. For example, the Hamiltonian is used to evaluate the energy of a configuration of the spin glass system in statistical physics, so that the Hamiltonian is the evaluation function and the ground-state is related to the minimum of the Hamiltonian. In combinatorial problems, such as the SAT problem [38], the evaluation function is the total number of unsatisfied clauses of the current configuration. So the evaluation value for a solution for the SAT problem is zero. In biological systems, the evaluation function is the fitness function.

Different evaluation functions typically guide the system to different final states, but sometimes guide the system to the same state. Therefore, for a specified system people are able to construct different evaluation functions to reach the same goal state. As the main theme of this paper, we investigate two ways to construct evaluation functions:

(1) **Global Evaluation Function (GEF)**. GEF uses global information to evaluate the whole system state. This is the traditional way to construct the evaluation function and is widely used in many systems, such as the Hamiltonian in spin glass, and the evaluation function that returns the total number of unsatisfied clauses in the SAT problem. The Global Evaluation Function embodies the idea of centralized control.

(2) **Local Evaluation Function (LEF)**. LEF uses the local (limited) information to evaluate the single agent state and guides the agent to evolve. As the study of Complex Systems [39] gets more and more attention, the idea of Local Evaluation Function has been applied to different distributed systems, especially in computer simulations for local interacting systems such as **Cellular Automata** [40], **Boid** [42], **Game of Life** [7], **Escape Panic model** [28], **Sandpile model** [41], etc.

<table>
<thead>
<tr>
<th>Information</th>
<th>Object (Agent)</th>
<th>Agent (Individual)</th>
<th>Whole system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local</td>
<td>Local Evaluation Function (Section 3.1.2)</td>
<td>Consistent Local Evaluation Function (Section 3.2.1)</td>
<td>Global Evaluation Function (Section 3.1.1)</td>
</tr>
</tbody>
</table>

| Table 1. Global Evaluation Function and Local Evaluation Function: the evaluated object and how much information is used for evaluation. |

It is a simple and natural idea to use a Global Evaluation Function to guide the whole system to evolve to a global goal. But can we use a Local Evaluation Function to guide each agent so that the whole system yields to the global goal in the end? In other words, if each agent makes its own decisions according to local information, can the agents get to the system state which is reached by evolution guided by the Global Evaluation Function? If ‘yes’, we say emergence happens in this system. Emergence is one of the significant properties of Complex Systems. Emergent complex behaviors from local interactions are found in natural and man-made systems [7][42][5][28].

---

1 Here the evolution is trying to maximize the fitness value (evaluation value).
Local and Global Evaluation Function for Evolution

Among the algorithms used for solving combinatorial problems, most traditional ones use a Global Evaluation Function such as Simulated Annealing [17], and some of them use Local Evaluation Function such as Extremal Optimization [18] and the Alife&AER model [1][2]. Our earlier paper [3] investigated emergent intelligence in solving Constraint Satisfaction Problems (CSPs, also called decision problems) [8] by modeling a CSP as a multi-agent system and then using Local Evaluation Function methods to find the solution.

This paper will study the relationship and essential differences between Local Evaluation Functions and Global Evaluation Functions for evolution of combinatorial problems, including decision problems and optimization problems. We use the example of the Coloring problem to demonstrate our ideas because it has two forms: the k-coloring Problem which is a decision problem and the Minimum Coloring Problem which is an optimization problem. We found that some Local Evaluation Functions are consistent with some Global Evaluation Functions while some are not. So Local Evaluation Function methods and Global Evaluation Function methods differ in two layers: Consistency and Exploration. We also compare the performances of Local Evaluation Function and Global Evaluation Function methods using computer experiments on 36 benchmarks of k-Coloring Problems and Minimum Coloring Problems.

This paper is organized as follows. In Section 2, computer algorithms for solving Coloring Problems are reviewed, and then the way to model a Coloring Problem as a multi-agent system is described. Section 3 is about Global Evaluation Functions and Local Evaluation Functions. We first show how to construct a Global Evaluation Function and a Local Evaluation Function for solving the Coloring Problem (3.1), and then show the differences between the Local Evaluation Function and Global Evaluation Function from aspects of consistency (3.2.1) and exploration heuristics (3.2.2). Computer experiments (3.3) show that the Local Evaluation Function (Alife&AER Model) and Global Evaluation Function (Simulated Annealing and Greedy) are good at solving different instances of Coloring Problem. The conclusions and discussions are presented in Section 4.

From now on, we use ‘LEF’ for Local Evaluation Function, ‘GEF’ for Global Evaluation Function and ‘EF’ for Evaluation Function. We use ‘assignment’ for a single variable’s assigned value, and ‘configuration’ for complete assignments for all variables.

2. COLORING PROBLEMS

The Coloring Problem is an important and classical combinatorial problem. It is an NP-complete problem and is useful in a variety of applications, such as time-tabling and scheduling, frequency assignment, register allocation [8], etc. It is simple and can be mapped directly onto a model of an anti-ferromagnetic system [29]. Moreover, it has obvious network structure so that it is a good example for studying network dynamics [47]. It is good for this paper because it can be treated in decision problem (CSPs) form or optimization problem form.

K-coloring Problem:

Given a graph $G = (V, e)$, where $V$ is the set of $n$ vertices and $e$ is the set of edges, we need to color each vertex using a color from a set of $k$ colors. It is called a ‘conflict’ if a pair of linked vertices has the same color. The objective of the problem is either to find a non-conflict coloring configuration for all vertices, or to prove that it is impossible to color this graph.

![Figure 1: A solution for a 3-coloring problem](image-url)
4 Jing HAN

without any conflict using \( k \) colors. Therefore, the \( k \)-coloring Problem is a decision problem. This problem has \( n \) variables for the color of \( n \) vertices of the graph, and the possible value of each variable is the given set of \( k \) colors. For each pair of nodes linked by an edge, there is a binary constraint between the corresponding variables that disallows identical assignments. Fig. 1 is an example:

\[
\text{Variable set } X = \{X_1, X_2, X_3, X_4\}, \\
\text{Variable domain } D_1 = D_2 = D_3 = D_4 = \{\text{green, red, blue}\}, \\
\text{Constraint set } R = \{X_1 \neq X_2, X_1 \neq X_3, X_1 \neq X_4, X_2 \neq X_3\}.
\]

The \( k \)-Coloring problem \( (k \geq 3) \) is NP-complete, which means it might be impossible to find an algorithm to solve the \( k \)-Coloring problem in polynomial time.

**Minimum Coloring Problem:**

The definition is almost the same as the \( k \)-coloring Problem except that \( k \) is unknown. It requires us to find the minimal number of colors that can color the graph without any conflict. The minimal possible number of colors of \( G \) is called the chromatic number and denoted by \( \chi(G) \). So the domain size of each variable is unknown. For the graph shown in Fig.1, \( \chi(G) = 3 \).

Obviously, the Minimum Coloring Problem is more difficult to solve than the \( k \)-coloring Problem, and it is also NP-complete.

### 2.1. ALGORITHMS

There are several basic general algorithms for solving combinatorial problems. Of course they can also be used to solve the Coloring Problem. Although we focus on the Coloring Problem, the purpose of this paper is to demonstrate the ideas of LEF and GEF methods for all combinatorial problems. What follows is a brief review of all general algorithms for the combinatorial problem even though some of them might not have been applied to the Coloring Problem. The focus is only on the basic ones, so variants of the basic algorithm will not be discussed because its main idea remains the same as the basic one.

![Classification of algorithms](FIGURE 2)

**FIGURE 2** Classifications of basic algorithms for combinatorial problems. This paper focuses on GT-SS algorithms, the intersection of single-solution and Generate-test (the gray area).

Fig. 2 is our classification of current algorithms. According to how many (approximate) solutions the system is seeking concurrently, they can be divided into **Single-solution Algorithms** and **Multi-solution Algorithms**. According to the style of search, there are two types: **Systematic**
Local and Global Evaluation Function for Evolution

This paper will focus on GT-SS algorithms: the intersection of Single-solution Algorithms and the Generate-test (the gray area in Fig.2), because the process of finding a solution using GT-SS algorithms can be regarded as the evolution of a multi-agent system.

**Single-solution Algorithms (SS).** The system is searching for only one configuration at a time. Some examples are Backtracking [10], Local Search [11-17, 21-22], Extremal Optimization (EO)[18], and Alife&AER model[1][2].

**Multi-solution Algorithms (MS).** The system is concurrently searching for several configurations, which means that there are several copies of the system. Examples of this are Genetic Algorithms [4], Ant Colony Optimization [5] and Particle Swarm Optimization [42]. They all are population-based optimization paradigms. The system is initialized with a population of random configuration and searches for optima by updating generations based on the interaction between configurations.

**Backtracking** (depth-first search) assigns colors to vertices sequentially and then checks for conflicts in each colored vertex. If conflicts exist, it will go back to the most recently colored vertex and perform the process again. Many heuristics have been developed to improve the performance. For the Coloring Problem, several important algorithms are SEQ[9], RLF[9] and DSATUR [30].

**Generate—test (GT)** is a much bigger and more popular family than the Backtracking paradigm. GT generates a possible configuration of all variables (colors all vertices) and then checks for conflicts, (i.e., checks if it is a solution). If it is not a solution, it will modify the current coloring configuration and check it again until a solution is found. The simplest but non-efficient way to generate a configuration is to select a value for each variable randomly. Smarter configuration generators have been proposed, such as hill climbing[21] or min-conflict [12]. They move to a new configuration with a better evaluation value chosen from the current configuration’s neighborhood (see detail in 3.1) until a solution is found. This style of searching is called Local Search (or neighborhood search). For many large-scale combinatorial problems, Local search always gives better results than the systematic Backtracking paradigm. To avoid being trapped in the local-optima, it sometimes performs stop-and-restart, random walk [22] and Tabu search [15-16, 31]. Simulated Annealing [17], a famous heuristic of local search is inspired by the roughly analogous physical process of heating and then slowly cooling a substance to obtain a strong crystalline structure. People proposed Simulated Annealing schedules to solve the Coloring Problem and showed that it can dominate backtracking diagrams on certain types of graphs [9].

Some algorithms that use the complex systems ideas are Genetic Algorithms, Particle Swarm Optimization, Extremal Optimization and Alife&AER model.

Extremal Optimization and Alife&AER model also generate a random initial configuration, and then change one variable’s value in each improvement until a solution is found or the maximal tried number is reached. Note that Extremal Optimization and the Alife&AER model improve the variable assignment based on Local Evaluation Functions, which is the main difference from Simulated Annealing.

Both the Genetic Algorithms and Particle Swarm Optimization have many different configurations at a time. Each configuration can be regarded as an agent (it is called a ‘chromosome’ in Genetic Algorithms). The fitness function for each agent is to evaluate how good the configuration represented by the agent is, so the evolution of the system is based on a GEF².

² Some implementations of Particle Swarm Optimization can use GEF or LEF. Since Particle Swarm Optimization is not a single-solution algorithm, it is not discussed in this paper.
All the above algorithms have their advantages and drawbacks, and no algorithm can beat all other algorithms on all combinatorial problems. Although Generate-test algorithms always beat Backtracking on large-scale problems, they are not complete algorithms; that is, they can not prove there is no solution for decision problems and sometimes they can not find a solution even though solutions exist.

2.2. MULTI-AGENT MODELING

We have already proposed the method of modeling a CSP in the multi-agent framework [3]. Here we briefly review some basic concepts again and translate the Coloring Problem in the language of multi-agent systems.

The concept of agent (denoted by \( a \)) is a computational entity: It is autonomous; it is able to act with other agents and get information from the system; it is driven by some objectives and has some behavioral strategies based on information it collects.

Multi-agent system (MAS) is a computational system that consists of a set of agents \( A = \{ a_1, a_2, \ldots, a_n \} \), in which agents interact or work together to reach goals [27]. Agents in these systems may be homogeneous or heterogeneous, and may have common or different goals. Multi-agent systems focus on the complexity of a group of agents, especially the local interactions among them.

It is straightforward to construct a Multi-agent System for a Coloring Problem:

<table>
<thead>
<tr>
<th>Coloring Problem ⇔ Multi-agent System</th>
</tr>
</thead>
<tbody>
<tr>
<td>• ( n ) Vertices ( { V_1, V_2, \ldots, V_n } ) ( \rightarrow ) ( n ) Agents ( { a_1, a_2, \ldots, a_n } )</td>
</tr>
<tr>
<td>• Possible colors of ( V_i ) ( \rightarrow ) Possible states of agent ( a_i )</td>
</tr>
<tr>
<td>• Colors of ( V_i ) is Red ( \iff ) ( X_i = \text{red} ) ( \rightarrow ) State of agent ( a_i ) is ( a_i.\text{state} = \text{red} )</td>
</tr>
<tr>
<td>• Link between two vertices ( \rightarrow ) Interaction between two agents</td>
</tr>
<tr>
<td>• Graph of the problem ( \rightarrow ) Interaction network in MAS</td>
</tr>
<tr>
<td>• Configuration ( S = \langle X_1, X_2, \ldots, X_n \rangle ) ( \rightarrow ) a system state</td>
</tr>
<tr>
<td>• Solution ( S_{sol} ) ( \rightarrow ) Goal system state</td>
</tr>
<tr>
<td>• Process of finding a solution by GT-SS algorithms ( \rightarrow ) Evolution of MAS</td>
</tr>
<tr>
<td>• Evaluation function ( \rightarrow ) Evolution direction of MAS</td>
</tr>
</tbody>
</table>

**FIGURE 3** A Multi-agent System (right) for a 3-coloring problem (left). The table with 3 lattices beside each agent represents all possible states (R-red, G-green, B-blue), and the circle indicates the current agent state (color). The current assignment \( S = \langle G, R, B, B \rangle \) is a solution.
3. GEF & LEF

3.1. EVOLUTION

Once we model the Coloring Problem as a multi-agent system, we need to design a set of rules for its evolution, which is called the ‘algorithm’ in computer science. The core of the algorithm is how the whole system (configuration) changes from one state to another state, i.e., how variables change their assigned values.

For example, if the system is initialized as shown in Fig.4, \( S = <G, R, B, G> \). Obviously it is not a solution, because there is a conflict between \( V_1 \) and \( V_4 \) which are linked to each other and have the same color. Thus, the system needs to evolve to reach the solution state.

The process of finding a solution can be regarded as the evolution of the MAS; the first step of Generate-test algorithms is to generate the initial configuration of all variables, all agents in the multi-agent system will get their initial color; then in the following time steps, one or several agents change their states according to their strategies, until a solution \( S_{sol} = <G, R, B, B> \) is found (see Fig. 4(2)).

How does the system evolve to the solution state? This is the core of problem solving and MAS. As we mentioned above, GT-SS algorithms (Local search-Simulated Annealing, Extremal Optimization and Alife&AER model, see Fig.2) are schedules for evolution of the MAS. In the language of MAS, those algorithms all share the same framework as the one in Fig.5.

The essential difference between the LEF and GEF lies in the process of Evolution(\( S \)) in step 2 and it separates them into two groups:

**GEF methods**—use the global evaluation function to select the best system state in its current neighborhood, such as Local Search (Simulated Annealing);

**LEF methods**—use the local evaluation function to determine the movement of a single agent (change of variable assignment), such as Extremal Optimization and the Alife&AER model.
1. Initialization ($t=0$): each agent picks a random state, get $S(0)$.

2. For each time step ($t$): according to some schedule and heuristics, one or several agents change their state, so that $S(t+1) = \text{Evolution}(S(t))$; $t \leftarrow t+1$.

3. Repeat step 2 until a solution is found or reaches the maximum tries.

4. Output the current state $S(t)$ as the (approximate) solution.

**FIGURE 5.** The framework of GT-SS algorithms.

### 3.1.1. Traditional Methods: Global Evaluation Functions

$\text{Evolution}(S)$ of step 2 of GEF methods can be described as:

| Compute the GEF values of all (or some, or one) neighbors of the current system state $S(t)$, and then select and return a neighboring state $S(t)'$ to update the system; |

*Neighborhood* in GEF methods is the search scope for the current system state. For example, we can define the neighbor structure of a configuration $S \in \Omega$ based on the concept of Hamming distance:

$$N_S^d = \{ S' | \text{Hamming-distance}(S, S') = d \},$$

where $\text{Hamming-distance}(S, S') = \sum_{x_i \in S, x_i \in S'} \delta(x_i, y_i)$, with $\delta$ denoting the Kronecker function $\delta(x, y) = 1$ if $x = y$ and otherwise $\delta(x, y) = 0$.

If $d=1$, e.g., the search space of GEF is $N_S^1$, in the $k$-coloring Problem, there are $n(k-1)$ neighbors of each system state. Fig.4(2) is a neighbor of Fig.4(1). In Fig.6, (b) and (c) are neighbors of (a). If $d$ is larger, the neighborhood size is bigger, that means the search scope is larger, the cost of searching all these neighbors increases, but there will probably be more improvement in each time step. So the neighbor structure will affect the efficiency of the algorithm [25]. In the following discussions, we always use $N_S^1$ ($d=1$) as the neighborhood structure, since the neighborhood structure is not in the scope of this paper. Note that *Simulated Annealing* just randomly generates a neighbor and accepts it if it is a better state, or with a specified probability to accept a worse state. Here ‘better’ and ‘worse’ are defined by the evaluation value.

**GEF (Global Evaluation Function)** evaluates how good the current system state $S$ is. In *Simulated Annealing*, GEF is the energy or cost function of the current system state. It guides search. Obviously, *evaluation function* plays an important role in the algorithm. There are varieties to define the *evaluation function* $E$, and a suitable definition of GEF for a specified problem will get a good performance.

- **A GEF for the $k$-coloring Problem:**

  One naïve GEF definition for the $k$-coloring Problem is to count the number of conflicts in the current configuration. For simplicity, formula (1) counts *twice* the number of conflicts:
Local and Global Evaluation Function for Evolution

\[ E_{GEF-k}(S) = \sum_{i=1}^{n} \sum_{j \neq i} \delta(X_i, X_j) \]  \hspace{1cm} (1)

where \( S_i=\{X_1|<i,j>\in e\} \) denotes the set of variables that link to \( V_i \).

Formula (1) is called a GEF because it considers the whole system including all agents. For the system state shown in Fig.4(1), the evaluation value is \( 1 \times 2 = 2 \). Fig.4(2) shows a solution state, so \( E_{GEF}(S_{sol}) = 0 \). Minimizing the evaluation value is the goal of the solving process, although a state with a better evaluation value does not always mean that it is closer to the solution state.

- **A GEF for the Minimum Coloring Problem:**

There are several definitions [9][35][36] of evaluation function for finding the chromatic number \( \chi(G) \). We modified the evaluation function version given by Johnson [9] to be:

\[ E_{GEF-O}(S) = m \sum_{i=1}^{n} |C_i|^2 + n \sum_{i=1}^{n} \sum_{X_e \in S_i} \delta(X_i, X_j) \]  \hspace{1cm} (2)

where \( m \) is the current total number of colors the system uses, \( C_i \) is a set of variables that are colored by the \( i \)th color. We assign each color \( s \) an index number \( \text{Index}(s) \), say ‘red’ is the 1st color: \( \text{Index}(‘red’)=1 \), ‘green’ is the 2nd color: \( \text{Index}(‘green’)=2 \), ‘blue’ is the 3rd color: \( \text{Index}(‘blue’)=3 \), and so on. So

\[ C_i = \{X|\text{Index}(X_j) = i, j \in 1..n\} \].

Part (b) of evaluation function (2) counts twice the number of conflicts in the current configuration; Part (a) of the evaluation function favors using fewer colors. \( n \) in part (b) is a weight value to balance part (a) and part (b), which makes all the local optima in the evaluation function (2) correspond to a non-conflict solution, e.g., a feasible solution.

For example, in Fig.6 (a), \( C_1=\{X_2, X_3\} \), \( C_2=\{\emptyset\} \), \( C_3=\{X_1, X_3\} \), so \( E_{GEF-O}(S_a)=-(2^2+2^2)+4 \times 1 \times 2 = -6 \). Using the same evaluation function (2), we can get the evaluation value of its two neighbors: \( E_{GEF-O}(S_b)=2 \) and \( E_{GEF-O}(S_c)=6 \). From \( E_{GEF-O}(S_b)>E_{GEF-O}(S_a) \), we know \( S_b \) is a worse state than \( S_a \) because it uses one more color, but it still has one conflict as does \( S_a \). And \( E_{GEF-O}(S_c)<E_{GEF-O}(S_a) \), \( S_c \) is a better state because it can remove the conflict in \( S_a \) although it uses one more color. Actually \( S_c \) is the optimal solution because its evaluation value is the global minimum.

**FIGURE 6.** An example for using GEFs for a coloring problem. \( S_a \) is the initial state. \( S_b \) and \( S_c \) are two of \( S_a \)'s neighbors. As a 3-coloring problem, based on formula (1), \( E_{GEF-k}(S_a)=2 \), \( E_{GEF-k}(S_b)=2 \), \( E_{GEF-k}(S_c)=0 \). So \( S_a \) is the solution. As a minimum Coloring Problem based on formula (2), \( E_{GEF-O}(S_a)=0 \), \( E_{GEF-O}(S_b)=2 \), \( E_{GEF-O}(S_c)=6 \), so \( S_c \) is the best one (also the optimal solution).
So we can see that the GEF methods has two main features: (1) it evaluates the whole system state, not a single agent’s state; (2) it uses all information of the system, formula (1) and (2) consider all agents’ states. Centralized control in GEF methods is obvious: in each step, the system checks all (or one) neighbors and selects the one that has the best evaluation value as the new system state. Therefore, no agents here are autonomous. They all obey the choice of the centralized controller.

3.1.2. Local Evaluation Function Methods

Evolution $S$ of step 2 (of Fig.5) of LEF methods can be described as:

For each agent $a_i$, do the following sequentially$^3$:

1. Compute the evaluation values of all/some states of $a_i$ by $a_i$’s LEF, and then select a state for $a_i$ to update itself.

Here the search is distributed because agents make their own decisions. The search space for each agent is its possible states. In order to decide which state should be a better one, each agent uses an LEF to evaluate its possible states. So LEF aims at a single agent, unlike GEF which aims at the whole system. And LEF uses local information: when it evaluates a color (state $s$) of a single agent ($a_i$), it only considers agents which have constraints (links) with $a_i$. In the Coloring Problem, they are vertices that link to $a_i$. Therefore, each agent has its own LEF and it wants to minimize the evaluation value (i.e., maximize its own profit) without considering the whole system (the overall profit of the system).

After obtaining the evaluation values for all states of an agent, it will update its state using some strategies, such as Least-move, Better-move, and Random-move(random walk). Least-move means the agent will select the best state and change to that state; Random-move means the agent will change its state randomly (for more details, please see [2]). These strategies are part of the exploration heuristics (see section 3.2.2).

- AN LEF for the $k$-coloring Problem:

Like the GEF, there are varieties of definitions of the LEF for a problem. Formula (3) [2] is one possible definition of the LEF for each agent $a_i$ in the $k$-coloring Problem. It calculates how many conflicts each state ($s$) of agent $a_i$ receives based on the current assignment of $a_i$’s linked agents. Obviously, if for all $i \in 1..n$ we have $E_{LEF-k}^i(S_{-i}, s) = 0$, then $S$ is a solution (Fig.7-(c)).

$$E_{LEF-k}^i(S_{-i}, s) = \sum_{X_j \in X_i} \delta(X_j, s) \quad ---- (3)$$

For example, in Fig.6(a), agent $a_1$ links to $a_2$, $a_3$ and $a_4$, so we get:

$$E_{LEF-k}^1(S_{a_1-1}, R) = (\delta(X_2, R) + \delta(X_3, R) + \delta(X_4, R)) = 2$$
$$E_{LEF-k}^1(S_{a_1-1}, G) = (\delta(X_2, G) + \delta(X_3, G) + \delta(X_4, G)) = 0$$
$$E_{LEF-k}^1(S_{a_1-1}, X_1) = E_{LEF-k}^4(S_{a_1-1}, B) = (\delta(X_2, B) + \delta(X_3, B) + \delta(X_4, B)) = 1.$$ 

Since the current state’s evaluation value is higher than state ‘G’, $a_1$ will most probably change to state ‘G’. Agent $a_4$ only links to $a_1$, so it only considers $a_1$’s state:

---

$^3$ Or simultaneously in some cases, as in Cellular Automata. This is one of our future projects.
Local and Global Evaluation Function for Evolution

\[ E_{LEF-k}^a (S_{a,-4}, X_i) = \delta(X_i, R) = 0 \]
\[ E_{LEF-k}^4 (S_{a,-4}, G) = \delta(X_i, G) = 0 \]
\[ E_{LEF-k}^4 (S_{a,-4}, B) = \delta(X_i, B) = 1. \]

Similarly, \( E_{LEF-k}^a \) for Agent \( a_2 \) only considers \( a_1 \) and \( a_3 \); \( E_{LEF-k}^a \) for Agent \( a_3 \) only considers \( a_1 \) and \( a_2 \). Continuing this way, we can get evaluation values (Fig. 7.) for all states of each agent.

\[
\begin{align*}
E_{LEF-O}^1 (S_a, s) &= Index(s) + n \sum_{X_j \in S_i} \delta(X_j, s) \\
E_{LEF-O}^2 (S_a, s) &= Index(s) + n \sum_{X_j \in S_i} \delta(X_j, s)
\end{align*}
\]

where \( Index(s) \) is described in section 3.1.1, returns the index number of color \( s \).

Part (b) of the evaluation function (4) counts how many conflicts \( a_i \) will receive from its linked agents if \( a_i \) takes the color \( s \); Part (a) of (4) favors using fewer colors, like putting pressure on each agent to try to get a color with a smaller index number. \( n \) in part (b) is a weight value to balance part (a) and part (b), which makes all the local optima in evaluation function (4) correspond to a non-conflict state to the agent. Obviously, if for all \( i \in n \) we have \( E_{LEF-O}^i (S_a, X_i) \leq n, S = <X_1, X_2, ..., X_n> \) is a feasible solution, but not always the optimal solution.

We can evaluate each state of \( a_1 \) in Fig. 6(a):
\[ E_{LEF-O}^1 (S_a, R) = Index(R) + n \times (\delta(X_2, R) + \delta(X_3, R) + \delta(X_4, R)) = 1 + 4 \times 2 = 9 \]
\[ E_{LEF-O}^1 (S_a, G) = Index(G) + n \times (\delta(X_2, G) + \delta(X_3, G) + \delta(X_4, G)) = 2 + 4 \times 0 = 2 \]
Since the current state’s evaluation value is higher than state ‘G’, \( a_1 \) will most probably perform a \textit{least-move} to change from ‘B’ to state ‘G’, and get to an optimal solution \( S_c \). Or, with lower probability \( a_1 \) will perform a \textit{random-move} to change to state ‘Y’ and get to another feasible solution (but not an optimal one, since it uses 4 colors); or \( a_4 \) changes to ‘G’ and gets to \( S_b \).

\[
\begin{align*}
E_{\text{LEF-O}}^i(S_{a-1}, X_i) &= E_{\text{LEF-O}}^i(S_{a-1}, B) \\
&= \text{Index}(B) + n \times (\delta(X_2, B) + \delta(X_3, B) + \delta(X_4, B)) = 3 + 4 \times 1 = 7.
\end{align*}
\]

So we can see that the \textit{LEF methods have two main features:} (1) it evaluates the state of a single agent, not the whole system; (2) it uses local information of the system, formula (3) and (4) only considers its linked agents’ states (e.g., only knows the information of its related subgraph).

\textit{Decentralized control} in LEF methods is also obvious: in each step, the system dispatches all agents sequentially (in the \textit{Extremal Optimization} algorithm, the system dispatches the agent who is in the worst state according to the LEF). All agents are autonomous and decide their behaviors based on the \textit{LEF} that maximize their own profit. Therefore, if the whole system can achieve a global solution based on all selfish agents which are using the LEF, we call this \textit{emergence} and the system \textit{self-organizes} towards a global goal because there is no centralized control.

### 3.2. ESSENTIAL DIFFERENCES BETWEEN LEF AND GEF

#### 3.2.1. Consistency and Inconsistency

In section 3.1 we studied the LEF and GEF by using the Coloring Problem. Now we know there are two aspects we should consider while constructing an evaluation function:

1. What object is being evaluated? In the GEF, the whole system is considered, while in the LEF a single agent is considered;
2. How much information is used in the evaluation? The GEF considers the whole system state, while the LEF only considers the agent’s linked agents.

This fits the concepts we mentioned in Table 1. But what if the evaluation functions consider the whole system and only use local information (see Table 1. the gray one)? What is the relationship between the LEF and GEF?
Local and Global Evaluation Function for Evolution

Consistency between LEF and GEF

In the following, we use \( S' = \text{replace}(S, i, s, s') \), \( s, s' \in D_i \), to denote \( S' \) is achieved by replacing the state \( s \) of \( X_i \) in \( S \) to be \( s' \). So, all variables share the same state in \( S \) and \( S' \) except \( X_i: X_i = s \) while \( X'_i = s' \). Obviously, \( S' \in N^i_S \).

**Definition 1** LEF is **consistent** with GEF if it is true for \( \forall S \in \Omega \) and \( \forall S' \in N^i_S \) (\( S' = \text{replace}(S, i, s, s') \)) that

\[
\text{sgn}(E_{\text{GEF}}(S') - E_{\text{GEF}}(S)) = \text{sgn}(E^i_{\text{LEF}}(S \setminus i, s') - E^i_{\text{LEF}}(S \setminus i, s)),
\]

where \( \text{sgn}(x) \) is defined as: \( \text{sgn}(x) = 1 \) when \( x > 0 \), \( \text{sgn}(x) = -1 \) when \( x < 0 \) and \( \text{sgn}(0) = 0 \).

For convenience, sometimes we just simply say \( \text{sgn}(\Delta E_{\text{GEF}}) = \text{sgn}(\Delta E^i_{\text{LEF}}) \) for (5).

**Definition 2** LEF is **restrict-consistent** with GEF if (I) LEF is consistent with GEF and (II) \( \exists \alpha \in \mathbb{R}^+ \) such that it is true for all \( S \in \Omega \) that

\[
E_{\text{GEF}}(S) = \alpha \sum_{i=1}^{n} E^i_{\text{LEF}}(S \setminus i, X_i).
\]

So restrict-consistent is the subset concept of consistent.

It is easy to prove that **LEF-formula (3) is restrict-consistent with GEF-formula (1) in the k-Coloring Problem**. For example in the 3-Coloring Problem shown in Fig.6, agent \( a_1 \) uses the LEF formula (3) to change to be ‘G’ which can decrease its valuation value (from 1 to 0). This change is also a good change for the whole system, because it decreases the evaluation value of the whole system based on a GEF-formula (1) (from \( E_{\text{GEF}}(S) = 2 \) to \( E_{\text{GEF}}(S) = 0 \)). Suppose we use Simulated Annealing. The current system state is \( S \) and we randomly generate a neighbor \( S' = \text{replace}(S, j, s, s') \). Then we need to calculate \( \Delta = E_{\text{GEF}}(S') - E_{\text{GEF}}(S) \) to decide whether we accept \( S' \) or not. In the consistent case, it is not necessary to consider all agents’ states and \( \Delta E_{\text{GEF}} \) can be calculated by \( \Delta E^i_{\text{LEF}} \):

\[
E_{\text{GEF}}(S') - E_{\text{GEF}}(S) = 2 \Delta E^i_{\text{LEF}}.
\]

What does consistency mean? A good agent decision based on the LEF that can decrease the value of \( E^i_{\text{LEF}} \), is also a good decision for the whole system based on the GEF that can decrease the value of \( E_{\text{GEF}} \). In the k-Coloring Problem, while all agents get to a state that is evaluated by formula (3) of zero, the evaluation value of formula (1) of the whole system is also zero which means it is a solution. This makes it easier for us to understand why the LEF can guide agents to a global solution. When LEF is consistent with the GEF, LEF and GEF indicate the same equilibrium points. For a certain configuration \( S \), if it is the local optimum of GEF, then it is the Nash Equilibrium [44] of LEF, and vice versa.

**Definition 3** If \( E_{\text{GEF}}(S') \geq E_{\text{GEF}}(S) \) is true for \( S \in \Omega \) and \( \forall S' \in N^i_S \), then \( S \) is called the **local optimum** of the GEF using \( N^i_S \) neighborhood structure in problem P.
The set of all local optima is denoted as \( L(P, E_{GEF}) \).

Optimal solutions of optimization problems and feasible solutions in decision problems belong to \( L(P) \) and have the smallest evaluation value.

**Definition 4** A Nash equilibrium of an LEF in a problem \( P \) is defined as \( S = \langle s_1, s_2, \ldots, s_n \rangle \), \( S \in \Omega \), such that

\[
E^i_{LEF}(S_{-i}, s_i) \leq E^i_{LEF}(S_{-i}, s_i') \quad \text{for } \forall s_i' \in D_i
\]

holds for all \( i = 1, 2, \ldots, n \).

The set of all Nash equilibria of the \( P \) and the LEF is denoted as \( N(P, E_{LEF}) \).

**Consistency theorem.** Given a problem \( P \), a GEF, and an LEF: LEF is consistent with GEF (i.e. relation (5) holds between GEF and LEF) is a necessary and sufficient condition for

\[
(\forall S \in \Omega) \quad S \in L(P, E_{GEF}) \iff S \in N(P, E_{LEF}).
\]

It follows that \( L(P, E_{GEF}) = N(P, E_{LEF}) \).

**Proof.**

(I) Relation (5) is the sufficient condition of relation (7).

If \( S \in L(P, E_{GEF}) \), \( S \) is a local optimum, i.e., \( E_{GEF}(S') \geq E_{GEF}(S) \) is true for \( \forall S' \in N^1_S \)

\[
\Rightarrow \forall i=1, 2, \ldots, n, \forall x \in D_i, E_{GEF}(\text{replace}(S, i, s, x)) - E_{GEF}(S) \geq 0.
\]

Because \( \text{sgn}(E_{GEF}(\text{replace}(S, i, s, x)) - E_{GEF}(S)) = \text{sgn}(E^i_{LEF}(S_{-i}, x) - E^i_{LEF}(S_{-i}, s)) \)

\[
\Rightarrow \forall i=1, 2, \ldots, n, \forall x \in D_i, E^i_{LEF}(S_{-i}, x) - E^i_{LEF}(S_{-i}, s) \geq 0
\]

\[
\Rightarrow \forall i=1, 2, \ldots, n, \forall x \in D_i, E^i_{LEF}(S_{-i}, x) \geq E^i_{LEF}(S_{-i}, s)
\]

\( S \) is a Nash equilibrium, \( S \in N(P, E_{LEF}) \).

If \( S \in N(P, E_{LEF}) \), \( S \) is a Nash equilibrium, then we have

\[
\forall i=1, 2, \ldots, n, \forall x \in D_i, E^i_{LEF}(S_{-i}, x) \geq E^i_{LEF}(S_{-i}, s).
\]

Because \( \text{sgn}(E^i_{LEF}(S_{-i}, x) - E^i_{LEF}(S_{-i}, s)) = \text{sgn}(E_{GEF}(\text{replace}(S, i, s, x)) - E_{GEF}(S)) \),

\[
\Rightarrow \forall i=1, 2, \ldots, n, \forall x \in D_i, E_{GEF}(\text{replace}(S, i, s, x)) - E_{GEF}(S) \geq 0
\]

\( \Rightarrow \forall S' \in N^1_S, E_{GEF}(S') \geq E_{GEF}(S) \rightarrow S \) is a local optimum, \( S \in L(P, E_{GEF}) \).

(II) Relation (5) is the necessary condition of relation (7).

Table 2 shows all possible combinations of \( \Delta E_{GEF} \) and \( \Delta E_{LEF} \) for any \( S \in \Omega \) and any \( S' \in N^1_S \). Combinations marked by a ‘Y’ are those allowed by relation (5). Obviously, combinations marked by an ‘N’ are not allowed if we want to hold relation (7). Neither are those combinations marked by
Local and Global Evaluation Function for Evolution

a ‘?’. Because if \( S \) and \( S' \) are both local optima (\( \Delta E_{\text{GEF}} = 0 \)) and \( \Delta E_{\text{LEF}} > 0 \), it implies \( S' \) is not a Nash equilibrium; if \( S \) and \( S' \) are both Nash equilibria (\( \Delta E_{\text{LEF}} = 0 \)), and \( \Delta E_{\text{GEF}} > 0 \), it implies \( S' \) is not a local optimum. So ‘?’ should be replaced by ‘N’ in table 2, which means relation (5) is the necessary condition of relation (7).

**End of Proof.**

**Inference 1.** Given a problem \( P \), a GEF, and an LEF: if the LEF is consistent with the GEF, \( S \) is a solution and \( S \) is a local optimum of the GEF, then \( S \) is also a Nash equilibrium of the LEF.

Inference 1 helps us to understand why the LEF can solve problems when the consistent GEF can solve: because their equilibrium points are the same! They have the same phase transition, solution numbers and structure according to the spin glass theory [6]. The main difference between a consistent pair of LEF and GEF lies in their exploration heuristics, such as dispatch methods and strategies of agents. Simulated Annealing dispatches a random agent in a time step, and the Alife&AER model dispatches all agents in a time step. We will discuss this in section 3.2.2.

So can we construct any other LEF or GEF that can make an essential difference? Is there any case for an LEF that is not consistent with a GEF?

- **Inconsistency between LEF and GEF**

  The LEF-formula (4) can be used to solve the \( k \)-coloring Problem (the experimental result in section 3.3.1 shows that LEF-formula (4) works for the \( k \)-Coloring Problems). LEF-formula (4) and GEF-formula (1) are not consistent with each other. Suppose the current configuration is \( S \) and its neighboring configuration \( S' = \text{replace}(S, i, s, s') \), then

  \[
  \Delta E_{\text{LEF}-i} = \frac{n}{2} \Delta E_{\text{GEF}} + \text{Index}(s') - \text{Index}(s). \tag{8}
  \]

  If \( \Delta E_{\text{GEF}} \neq 0 \):

  Because \( \text{Index}(s) \leq n \) for all \( s \), so \( |\frac{n}{2} \Delta E_{\text{GEF}}| > |\text{Index}(s') - \text{Index}(s)| \) holds for all \( s' \) and \( s \), we will get \( \text{sgn}(\Delta E_{\text{GEF}}) = \text{sgn}(\Delta E_{\text{LEF}-i}) \).

  If \( \Delta E_{\text{GEF}} = 0 \):

  \( \Delta E_{\text{LEF}-i} \) can be larger than, equal to or smaller than zero depending on \( \text{Index}(s') - \text{Index}(s) \). There exists \( S \) and \( S' \in N_S^1 \) such that \( \text{sgn}(\Delta E_{\text{GEF}}) \neq \text{sgn}(\Delta E_{\text{LEF}-i}) \).

  So the assertion (5) is true only when \( \Delta E_{\text{GEF}} \neq 0 \). LEF-formula (4) is inconsistent with GEF-formula (1) in the \( k \)-Coloring Problem.

**Definition 5** LEF is inconsistent with GEF if LEF is not consistent with GEF.
Definition 6 LEF is weak-inconsistent with GEF iff \( \forall S \in \Omega \) and \( \forall S' \in N_1^G \), relation (5) is true only when \( \Delta E_{GEF} \neq 0 \); GEF is weak-inconsistent with LEF iff \( \forall S \in \Omega \) and \( \forall S' \in N_1^G \), relation (5) is true only when \( \Delta E_{LEF} \neq 0 \) (see Table 3).

So weak-inconsistency is not a symmetrical relation and it is a subset concept of inconsistency. LEF-formula (4) is weak-inconsistent with GEF-formula (1) in k-Coloring Problem.

Inference 2 Given a problem \( P \), a GEF and an LEF: that the LEF is weak-inconsistent with the GEF implies \( N(P, E_{LEF}) \subset L(P, E_{GEF}) \). That the GEF is weak-inconsistent with the LEF implies \( L(P, E_{GEF}) \subset N(P, E_{LEF}) \).

GEF-formula (2) and LEF-formula (4) in the Minimum Coloring Problem is an example of inconsistency but not weak-inconsistent. The minimal number of colors \( \chi(G) \) is a global variable. We can see that the linear relationship between formula (2) and formula (4) only exists between part (b), not between part (a). It is easy to prove the inconsistency: suppose a variable \( X_k \) changes its color from the \( l \)-th color to the \( h \)-th color. If this will not cause a change of conflict numbers (conflicts between \( X_k \) and its linked vertices keep the same number, part (b) remains the same), then

\[
\Delta E_{GEF-O} = 2(|C_k|-|C_l|-1) \quad \text{and} \quad \Delta E_{LEF-O} = h-l.
\]

If \( h>l \) and \( |C_k|>|C_l|+1 \), we will get \( \Delta E_{GEF-O} < 0 \) and \( \Delta E_{LEF-O} > 0 \);

If \( h<l \) and \( |C_k|>|C_l|+1 \), we will get \( \Delta E_{GEF-O} > 0 \) and \( \Delta E_{LEF-O} < 0 \). In the LEF, it is more likely that \( h<l \) because the agent favors colors which have smaller index numbers according to the formula (4). So when \( h<l \) and \( C_h \) has no fewer vertices than \( C_l \), a good choice for an agent (from \( l \)-th color to \( h \)-th color) based on the LEF will be a bad one for the whole system based on the GEF. This occurs because part (a) of formula (2) favors the unbalanced distribution of colors.

Figure 9 is the example for the Minimum Coloring Problem. \( S_a = <B,G,G,B> \) and \( S_b = <B,B,G,B> \) are two neighboring system states such that \( S_a \notin N^1_{S_a} \) and \( S_b \notin N^1_{S_b} \). From the LEF judgement, \( E^2_{LEF-O}(S_a,G) - E^2_{LEF-O}(S_b,B) = 9 - 8 = 1 > 0 \), indicating \( S_a \) is better than \( S_b \). From the GEF judgement, \( E_{GEF,0}(S_a) = - (2^2 + 2^2) + 6 \times 2 = 4 > E_{GEF,0}(S_b) = - (1^2 + 3^2) + 6 \times 2 = 2 \), indicating \( S_b \) is better than \( S_a \). So, using the LEF, \( \alpha_l \) will not change from ‘G’ to ‘B’ while using GEF system will change from \( S_a \) to \( S_b \). This indicates that the LEF-formula (4) and the GEF-formula (2) will lead the system to different states.

<table>
<thead>
<tr>
<th>( \Delta E_{LEF} )</th>
<th>( \Delta E_{GEF} )</th>
<th>( &gt;0 )</th>
<th>( =0 )</th>
<th>( &lt;0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( &gt;0 )</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>( =0 )</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>( &lt;0 )</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td></td>
</tr>
</tbody>
</table>

(1) LEF is weak-inconsistent with GEF

<table>
<thead>
<tr>
<th>( \Delta E_{LEF} )</th>
<th>( \Delta E_{GEF} )</th>
<th>( &gt;0 )</th>
<th>( =0 )</th>
<th>( &lt;0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( &gt;0 )</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>( =0 )</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>( &lt;0 )</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td></td>
</tr>
</tbody>
</table>

(2) GEF is weak-inconsistent with LEF

Table 3. Relations between LEF and GEF in Weak-inconsistency.

‘Y’ means allowed, ‘N’ means not allowed.
Inconsistency means the individual’s profit (defined by an LEF) will (sometimes) conflict with the profit (defined by a GEF) of the whole system. Does this mean if an LEF is inconsistent with a GEF, the LEF is also inconsistent with other GEFs? Actually, we can find the consistent GEF for LEF--formula (4) by simply defining the GEF as the sum of all LEF--formula (4):

\[
E_{\text{GEF}} = \sum E_{\text{LEF}}
\]

It is easy to prove that LEF-formula (4) is consistent with GEF-formula (9). And we can also say that the GEF-formula (9) is inconsistent with LEF-formula (4) if for any two neighboring configurations \( S_1 \) and \( S_2 \), \( \text{sgn}(\Delta E_{\text{LEF}}) = \text{sgn}(\Delta E_{\text{GEF}}) \). So LEF-formula (4) and GEF-formula (2) are inconsistent with each other: the LEF prefers \( S_1 \) while the GEF prefers \( S_2 \).

Inconsistency means the individual’s profit (defined by an LEF) will (sometimes) conflict with the profit (defined by a GEF) of the whole system. Does this mean if an LEF is inconsistent with a GEF, the LEF is also inconsistent with other GEFs? Actually, we can find the consistent GEF for LEF--formula (4) by simply defining the GEF as the sum of all LEF--formula (4):

\[
E_{\text{GEF}}(S) = \sum_{i=1}^{n} \text{Index}(X_i) + \sum_{i=1}^{n} \sum_{X_j \in S_i} \delta(X_i, X_j).
\]

It is easy to prove that LEF-formula (4) is consistent with GEF-formula (9). And we can also say that the GEF-formula (9) is inconsistent with LEF-formula (4) if for any two neighboring configurations \( S_1 \) and \( S_2 \), \( \text{sgn}(\Delta E_{\text{LEF}}) = \text{sgn}(\Delta E_{\text{GEF}}) \).

**Consistency between \( E_{\text{LEF}} \) and \( E_{\text{GEF}} = \sum E_{\text{LEF}} \)**

**Definition 7** If an LEF \( E_{\text{LEF}} \) is consistent with the GEF \( E_{\text{GEF}} = \sum E_{\text{LEF}} \), \( E_{\text{LEF}} \) is called consistent LEF, otherwise it is called inconsistent LEF.

Is it true that any LEF \( E_{\text{LEF}} \) is always consistent with \( E_{\text{GEF}} = \sum E_{\text{LEF}} \)? Are all LEFs consistent LEFs? Our answer is No. For example in the 2-Coloring Problem for a graph that only consists of two linked nodes \( i \) and \( j \), we have a total of four coloring configurations. If we define the evaluation value of the LEF of each node in each configuration as in table 4, we will find inconsistency between the LEF and GEF =\( \sum E_{\text{LEF}} \).
Figure 10. The LEF is inconsistent with the GEF = ∑ E_{LEF} in a 2-node-linked graph of a 2-Coloring Problem. (b), (c), (d) and (e) are LEF values (upper) and GEF values (lower) of each configuration for nodes $i$ and $j$. ‘$\rightarrow$’ is their tendency to change color given the other node’s color according to the LEF judgement. In (d) and (c), the LEF and the GEF lead the system to evolve to different configurations. So they are inconsistent even though GEF is the sum of LEFs of all agents. If we see ‘Green’ as ‘Cooperative’ and ‘Red’ as ‘Defect’, actually (a) is the payoff matrix of the Prisoner Dilemma problem [46] which is the most interesting problem in game theory, and the LEF of each agent is the utility function of each player.

Here we are trying to give some rules for constructing a consistent LEF under some limitations:

I. We limit our discussion on binary constraint problems and same domains for all variables. This means all constraints are only between two variables, and $D_1=D_2=\ldots=D_n=D$. The Coloring Problem is a binary-constraint problem and all domains are the same.

II. We only discuss the LEF which has the form

$$E_{LEF}(S_s,s) = f(s) + \beta \sum_{X_i \in S_s} g_{ij}(s, X_j), \quad \beta > 0 \text{ and } \beta \geq \max_{s',s} \{|f(s')-f(s)|\}. \quad \text{----(10)}$$

So the form of the GEF $E_{GEF}=\sum E_{LEF}$ is:

$$E_{GEF}(S) = \sum_i f(X_i) + \beta \sum_{X_i \in S_s} \sum_{X_j \in S_s} g_{ij}(X_i, X_j). \quad \text{----(11)}$$

The $f(s)$ is the preference for color (value) $s$ of the node (agent or variable). The second part is about constraints between two linked nodes, so $g_{ij}(X_i, X_j)$ is the penalty function of $X_i$ incurred by the constraint between $X_i$ and $X_j$. $\beta$ balances these two parts. Evaluation function (11) has the same form as Hamiltonian in Spin Glasses: the first part relates to the external field, and the second part relates to the interaction between two spins. Therefore, the form we defined here can cover a big class of LEFs for combinatorial problems, and the related GEFs have the same form as spin glass models. Evaluation functions (3) and (4) are of the form (10); evaluation functions (1) and (9) are of the form (11).

III. Only one form of static penalty function $g_{ij}(X_i, X_j)$ is considered here which satisfies $g_{ij}(X_i, X_j) = g_{ji}(X_j, X_i)$. We called this $g_{ij}(X_i, X_j)$ Symmetrical Penalty. $g_{ij}(X_i, X_j)$ in evaluation functions (1),

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>j</th>
<th>Green</th>
<th>Red</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>-3</td>
<td>-1</td>
<td>-3</td>
<td>-4</td>
</tr>
<tr>
<td>j</td>
<td>-3</td>
<td>-1</td>
<td>-2</td>
<td>-2</td>
</tr>
</tbody>
</table>

(a) LEF values for node $i$ (upper) and $j$ (lower). For example, if $i$ is Green and $j$ is Red, the LEF value of $i$ is -1, and the LEF value of $j$ is -4.
(2), (3), (4) and (9) is \( \delta(X_i, X_j) \) which is symmetrical penalty with \( x = y = 1 \) and \( r = z = 0 \). Obviously, the penalty function shown in Fig 10(a) is not symmetrical penalty. In the Coloring Problems, the GEF-formula (11) should guide the evolution of the system to \((s_l, s_h)\) or \((s_h, s_l)\), but not \((s_l, s_l)\) nor \((s_h, s_h)\) since two linked nodes should not have the same color. So,

Table 4 should satisfy \( x > r, x > z, y > r \) and \( y > z \).

Under the above limitations, the LEF can be the consistent LEF. That is what the following theorem states:

**Theorem 2** If \( \forall \langle i, j \rangle \in e, g_{ij}(X_i, X_j) = g_{ji}(X_j, X_i) \), then \( E_{LEF} \) which has the same form as formula (10) is consistent with \( E_{GEF} = \sum E_{LEF} \), e.g., \( E_{LEF} \) is a consistent LEF.

**Proof.**

For \( \forall S = \langle X_1, X_2, \ldots, X_n \rangle \in \Omega \) and \( \forall S' = \langle X'_1, X'_2, \ldots, X'_n \rangle \in N^1 \ (S' = \text{replace}(S, i, s, s')) \),

\[
\text{sgn}(E_{GEF}(S') - E_{GEF}(S)) = \text{sgn} \left( \sum_k f(X_k) + \beta \sum_{X_i \in S\setminus i} g_{ij}(X_i, X_j) \right) = \text{sgn} \left( f(X_i) - f(X_i) + \beta \sum_{X_i \in S\setminus i} (g_{ij}(X_i, X_j) - g_{ij}(X_i, X_j) + g_{ji}(X_j, X_i) - g_{ji}(X_j, X_i)) \right) = \text{sgn} \left( f(s) - f(s) + \beta \sum_{X_i \in S\setminus i} (g_{ij}(s, X_j) - g_{ij}(s, X_j) + g_{ji}(s, X_j) - g_{ji}(s, X_j)) \right)
\]

(because \( g_{ij}(X_i, X_j) = g_{ji}(X_j, X_i) \))

\[
= \text{sgn} \left( f(s) - f(s) + \beta \sum_{X_i \in S\setminus i} (g_{ij}(s, X_j) - g_{ij}(s, X_j)) \right) = \text{sgn} \left( f(s) - f(s) + 2\beta \sum_{X_i \in S\setminus i} (g_{ij}(s, X_j) - g_{ij}(s, X_j)) \right).
\]

And \( \text{sgn}(E_{LEF}(S_{-i}, s') - E_{LEF}(S_{-i}, s)) = \text{sgn} \left( f(s) - f(s) + \beta \sum_{X_i \in S\setminus i} (g_{ij}(s', X_j) - g_{ij}(s', X_j)) \right) \).

Because \( \beta > \max_{s', s} \{|f(s') - f(s)|\} \), we have

\[
\text{sgn}(f(s) - f(s) + 2\beta \sum_{X_i \in S\setminus i} (g_{ij}(s, X_j) - g_{ij}(s, X_j))) = \text{sgn}(f(s) - f(s) + \beta \sum_{X_i \in S\setminus i} (g_{ij}(s', X_j) - g_{ij}(s', X_j)) \rightarrow \text{sgn}(E_{GEF}(S') - E_{GEF}(S)) = \text{sgn} \left( E_{LEF}(S_{-i}, s') - E_{LEF}(S_{-i}, s) \right).
\]

**End of Proof.**
Theorem 2 only gives a sufficient condition but not the necessary condition for consistent LEF. Finding theorems for the necessary condition will be our future study because other \( g_{ij}(k, h) \) which are not Symmetrical Penalties might be good evaluation functions for efficient search.

So far, we know we can always find a GEF= \( \sum E_{\text{LEF}} \) for any LEF. Although not all GEFs constructed in this way are consistent with the LEF, which means the LEF and the GEF will direct the system to evolve to different configurations. However, on the other hand, not all GEFs can be decomposed to be \( n \) LEFs, such as in Minimum Coloring Problems, if we construct a GEF as:

\[
E_{\text{GEF-1.O}}(S) = m + n \sum_{i=1}^{n} \sum_{s, s_i \in S} \delta(s, s_i)
\]  ----(12)

where \( m \) is the current number of colors. This is a naïve definition and reflects the nature of the Minimum Coloring Problem: without looking at the whole system, a single agent won’t know how many colors have been used to color the graph. Since the LEF is not allowed to use the global information, which means that an agent only knows its linked agent and does not know the others, it seems it is difficult for a single agent to make a good decision for the whole system. One approximate solution here is the LEF-formula (4). If we compare its consistent GEF-formula (9) with GEF-formula (12), we will see formula (12) is more ‘rational’ (rationality is discussed in section 3.2.2) than formula (9) because formula (12) gives a ‘fair’ evaluation on all solution states while formula (9) doesn’t. We will discuss ‘rational’ and ‘partial rational’ evaluation functions in our next paper. The experiments in Section 3.3.2 show that the ‘rational’ GEF formula (12) works much worse than the ‘partial rational’ GEF formula (12).

3.2.2. Exploration of LEF & GEF

If an LEF is consistent with a GEF, what is the main difference between them? Is there any picture that can give some insight into understanding the exploration ways of the LEF and GEF?

To understand the behavior of the LEF and GEF, first of all we should understand the role of evaluation function in searching. The notion of this paper is ‘evaluation function is not a part of the problem; it is a part of the algorithm’:

\[
\text{GT-SS Algorithms} = \text{Evaluation Function} + \text{Exploration Heuristics}
\]

Therefore, the evaluation function values (EF values) are not built-in attributes for configurations of a given combinatorial problem. In the Backtracking algorithm, there is no evaluation function and only two possible judgments for current partial configurations: ‘no conflict’ or ‘conflict’ (or evaluation function in Backtracking has only two EF values). In the Generate-test paradigm, since the system needs to decide whether it accepts a new state or not, it has to compare the new state and the current state: which one might be better? Which one might be more close to the solution state? For this purpose, the evaluation function is introduced to make approximate evaluations for all possible configurations and lead the system to evolve to a solution state. Except for solution states, it is always difficult to compare which configuration is ‘better’ (more close to a solution state) than the other. So the evaluation function is an internal model for the reality problem. As we know, models do not always exactly reflect everything of the reality and it depends on the purpose. Therefore, models will have bias and will lose some information. Different models can be built for a problem. For example, there are several different GEFs for the Coloring Problem.
A rational evaluation function satisfies: (1) all solution states should have smaller EF value than all non-solution states; (2) all solution states should have the same EF value. A good evaluation function will lead the system to evolve to a solution state more directly, like evaluation function 1 is more efficient than evaluation function 2 in Fig.11. Because evaluation function 2 has many local optima which will make the evolution trapped in local optima. However, in many combinatorial problems, it is still impossible to construct an evaluation function that has no local optima. This is why they are still so difficult to solve and people developed so many heuristics to explore the configuration space. Otherwise, for evaluation function 1, we can simply use the greedy heuristic.

Searching (the GT-SS style) for solutions is just like walking in a dark night using a map: the map is the model for the configuration space, so the evaluation function is the generator of the map; heuristics are for exploration in the space based on the map. So an algorithm includes what map it uses and how it uses the map to explore.

It is impossible to compute evaluation values for all configurations at a time. It is only possible to get evaluation values for a small part of the configuration space at each time step. This is like needing a flashlight to look at a map on a dark night. But since the map is very big, only part of the map can be seen unless the flashlight is very powerful.

Using the metaphors of map and flashlight, we try to illustrate the exploration of LEF and GEF:

- GEF — one n-dimensional map and neighborhood flashlight

The traditional GEF methods use only one map which is n-dimensional for n-variable problems. For each time step, the system computes the evaluation values of neighbors in $N_1$, e.g., it uses a flashlight to look at the neighborhood of the current position on the map and make a decision about where to go (see Fig.12). There are several strategies for making decisions based on the highlighted part of the map: Greedy — always selects the best one to go (Fig.12); Greedy-random — most of the time selects the best to go with a small probability it will go to a random place (GEF-GR, see section 3.3); totally random, etc. In Simulated Annealing, the flashlight only highlights one point (configuration) each time in the neighborhood, and will move to that point if it is better than the current one, or with a small probability ($e^{\Delta E/T}$) move to that point.
even though it is worse. There can be many variations in types of flashlight and strategies for selection on the highlighted part of the map. In Section 3.3, we will see that both the evaluation function (map) and the exploration methods will affect the performance.

- **LEF — n d_i-dimensional maps and slice flashlight**

In the LEF, the distributive methods, agent $a_i$ has a $d_i$-dimensional map if $a_i$ has $d_i$ constrained agents (in the Coloring Problem, $d_i$ is the degree of $a_i$). In every time step each agent uses its 1-dimensional slice of its $d_i$-dimensional map for guiding its evolution.

Consistent $E_{LEF}$ (which is consistent with $E_{GEF} = \sum E_{LEF}$) and inconsistent $E_{LEF}$ (which is not consistent with $E_{GEF} = \sum E_{LEF}$) are very different.

For consistent LEF, a map of $a_i$ is actually a $d_i$-dimensional slice of the GEF map of $E_{GEF} = \sum E_{LEF}$, in other words, all LEF maps can make up a GEF map. In every time step, each agent uses a one-dimensional slice of the $n$-dimensional GEF map of $E_{GEF} = \sum E_{LEF}$. Since the system dispatches all agents sequentially, its exploration is from slice to slice. As Fig. 13 shows, the two-agent system evolves as ‘0’→’1’→’2’ using slices (b) and (c) of the GEF map (a). And the flashlight lightens a one-dimensional slice each time and the agent makes a decision according to its strategies: Least-move—always select the best one (Fig.13), or Better-move—select a random one and accept it if it is better than the current assignment, or Random-move which is totally random selection, etc. So here the LEF and GEF use the same map (evaluation function), and the difference only lies in how they use the map: what flashlight they use and what strategies they use for selection on the highlighted part of the map.

Figure 13. Consistent LEF: slice map of GEF and exploration. The system starts from ‘0’, and suppose it is the turn of $a_2$ to move; then $a_2$ gets a slice (b) through ‘0’ and selects the best one ‘1’ to move; and then it is the turn of $a_1$ to move, $a_1$ gets a slice (c) through ‘1’ and selects the best one ‘2’ to move. And then repeats. So the exploration is from one slice (dimension) to the other slice (dimension): 0→1→2.

For inconsistent LEF, a map of $a_i$ is no longer a $d_i$-dimensional slice of the GEF map of $E_{GEF} = \sum E_{LEF}$. Each agent uses its own map. So here the difference between the LEF and GEF lies not only on how they use the map, but also on what map (evaluation function) they are using.
As a conclusion to this section, we summarize the main differences between the GEF and LEF in table 5 from two layers: evaluation function (map) and exploration heuristics. This paper focuses on the first layer, the evaluation function. We believe that that using a different evaluation function will result in a different performance. The exploration heuristics will affect the performance too. This will be discussed in the next section using experiments.

<table>
<thead>
<tr>
<th>Evaluation Function (Map)</th>
<th>GEF</th>
<th>LEF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generator</td>
<td>$E_{GEF}$</td>
<td>$\sum E_{LEF}$</td>
</tr>
<tr>
<td>Dimension</td>
<td>$n$ (number of nodes)</td>
<td>$d_i$ (degree of each node $V_i$)</td>
</tr>
<tr>
<td>Equilibria</td>
<td>Local optima $L(E_{GEF})$</td>
<td>Nash Equilibria $N(E_{LEF})$</td>
</tr>
<tr>
<td></td>
<td>$N(E_{LEF})=L(\sum E_{LEF})$</td>
<td>$N(E_{LEF})\neq L(\sum E_{LEF})$</td>
</tr>
<tr>
<td>Exploration Heuristics</td>
<td>Neighborhood</td>
<td>1-dimensional Slice</td>
</tr>
<tr>
<td>Strategies</td>
<td>Greedy, Greedy-random, SA, Random, ...</td>
<td>Least-move, Better-move, Random, ...</td>
</tr>
</tbody>
</table>

Table 5. Differences between the GEF and the consistent/inconsistent LEF.

### 3.3. EXPERIMENTAL RESULTS

In the previous sections, we have proposed the LEF methods while comparing to the traditional GEF methods. We will ask questions about the performance: Can the LEF work for Coloring Problems? Can an LEF (GEF) solve problems that a GEF (LEF) can solve if the LEF is inconsistent with the GEF? If the LEF is consistent with a GEF, are their performances alike? In this section we present some preliminary experimental results on a set of Coloring problems (benchmarks from the Center for Discrete Mathematics and Theoretical Computer Science [http://mat.gsia.cmu.edu/COLOR/instances.html](http://mat.gsia.cmu.edu/COLOR/instances.html)). Section 3.3.1 and section 3.3.2 separately show results on the k-Coloring Problems and the Minimum Coloring Problems. Some observations are given in section 3.3.3.

Note that comparisons between LEF and GEF are difficult. There are two reasons: First, the LEF and GEF are concepts for evaluation functions. One specified LEF/GEF includes several algorithms according to their exploration heuristics and different parameter settings. Second, it is difficult to avoid embodying some preliminary knowledge in the algorithm. So the following experimental comparisons are limited to the specified settings of algorithms and problem instances.

In this paper, performances of three algorithms using different evaluation functions and specified parameters are compared:

1. LEF method: Alife&AER[1][2].
2. GEF methods:
a) **GEF-GR:** *Greedy-random*, which always looks for the best configuration in $N_s^1$ neighborhood and has a small probability for *random walk*;

b) **GEF-SA:** *Simulated Annealing*.

Here are details of the *Evolution*(S(t)) of those three algorithms in the framework shown in Fig. 5:

**LEF**

*Evolution*(S)

{  
1. For each agent $i=1$ to $n$ do 
2. Choose a random number $r$ in [0, 1].  
3. If $r < P_{\text{least-move}}$  // perform *Least-move* strategy  
4. Find one color $c$ that for all other possible colors $b$, $E_{LEF}^i (S, i, c) \leq E_{LEF}^i (S, i, b)$.  
5. Else  // perform a *random walk*  
6. $c =$ random-select (current possible colors);  
7. $S' =$ replace($S$, $i$, $X_i$, $c$);  
8. If $S'$ is a feasible solution, call *ProcessSolution*(S');  
   // feasible solution means non-conflict configuration  
9. Return S';
}

**GEF-GR:** *(Greedy – random + random walk)*

*Evolution*(S)

{  
1. Choose a random number $r$ in [0, 1].  
2. If $r < P_{\text{greedy}}$  // perform *greedy* strategy  
3. Find one neighbor $S' \in N_s^1$, that for all other $S'' \in N_s^1$, $E_{\text{GEF}}(S') \leq E_{\text{GEF}}(S'')$.  
4. Else  // perform a *random walk*  
5. $S' =$ random-select($N_s^1$);  
6. If $S'$ is a feasible solution, call *ProcessSolution*(S');  
7. Return S';
}

**GEF -SA:** *(Simulated Annealing*. More details can be found in paper[9])

*Evolution*(S)

{  
1. $S_0 =$ random-select ($N_s^1$);  
2. $\Delta = E_{\text{GEF}}(S_0) - E_{\text{GEF}}(S)$;  
3. If $\Delta \leq 0$  // perform *down-hill* move  
4. $S' =$S;  
5. Else $\Delta > 0$  
6. Choose a random number $r$ in [0, 1].  
7. If $r \leq e^{-\Delta/T}$  // perform *up-hill* move  
8. $S' =$S;  
9. Else  // don’t accept the new state  
10. return S;
11. If \( S' \) is a feasible solution, call \( \text{ProcessSolution}(S') \);
12. Return \( S' \);
}

\text{ProcessSolution}(S)
{
#define it is a \( k \)-Coloring Problem
Output the result \( S \);

#define it is a Minimum Coloring Problem
If \( S \) uses \( m \) colors & \( m \) is the chromatic number of the graph
Output the result \( S \) and \( m \).
Else if \( S \) uses \( m \) colors & \( m < \) Current Possible Colors Number
Current Possible Colors Number = \( m \); //reduce the possible color numbers, so \( N_s \) is reduced
}

The setting of \( P_{\text{least-move}} \) in the LEF method is equal to \( 1.5n/(1.5n+1) \), while \( n \) is the problem size; \( P_{\text{greedy}} \) in GEF-GR is also simply set to \( 1.5n/(1.5n+1) \). Parameter settings in GEF-SA are the same as in paper[9]: \( \text{initial } T = 10, \text{freezeLim} = 10, \text{sizeFactor} = 2, \text{cutoff} = 0.1, \text{tempFactor} = 0.95, \text{minPercent} = 0.02 \).

In the following experiments, we will examine the performance of finding a solution. First, we give the CPU runtime on PIII-1G PC WinXP of finding a solution and then we measure the runtime as operation counts [25], which is the number of agent moves used in finding a solution.

The cost of an agent move in the LEF includes steps 2-8 in Evolution(s), and a time step includes \( n \) agent moves. The cost of an agent move in GEF-GR includes steps 1-6. The cost of an agent move in GEF-SA (Simulated Annealing) includes steps 1-11. A time step in GEF (both GEF-GR and GEF-SA) only includes one agent move. The complexity mainly relates to calculation of conflicts and comparisons of evaluation function values (the complexity of picking up the smallest value from \( q \) values, is \( O(q) \)). Table 6 lists the complexities of the LEF, GEF-GR and GEF-SA (\( |e| \) is the number of edges in the graph; \( |e_i| \) is the number of edges from node \( V_i \); \( m \) is the current number of colors that are used in the graph, so in a \( k \)-Coloring Problem, \( m \) is constantly equal to \( k \) and in a Minimum Coloring Problem, \( m \) is a dynamic decreasing value but is always less than \( n \)).

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>An agent move of ( a_i )</th>
<th>A time step</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEF (Alife&amp;AER Model)</td>
<td>( O(2</td>
<td>e</td>
</tr>
<tr>
<td>GEF-GR</td>
<td>( O(2</td>
<td>e_i</td>
</tr>
<tr>
<td>GEF-SA</td>
<td>( O(2</td>
<td>e_i</td>
</tr>
</tbody>
</table>

Table 6. Complexity of an agent move and a time step of the LEF, GEF-GR and GEF-SA
So from Table 6, we can get the relationship of complexity of an agent move of LEF, GEF-GR and GEF-SA:

GEF-SA < LEF < GEF-GR.

It is easy to think if for each agent move the larger complexity it is, the better performance it will get. If so, the numbers of agent moves for finding a solution would have the relation:

#Move of GEF-SA > #Move of LEF > #Move of GEF-GR

Is this true? Let’s look at the experimental results in section 3.3.1 and 3.3.2. We will give some comparison there.

3.3.1. k-Coloring Problem

We compare the CPU runtime and the number of moves for finding a solution for a given k colors using different LEFs (formulas 3 and 4) and GEFs (formulas 1, 2 and 10). We will compare different performances of the following cases:

a) Consistent cases: LEF-formula (3) and GEF-formula (1), and LEF-formula (4) and GEF-formula (9);

b) Inconsistent cases: LEF-formula (3) and GEF-formula (9), and LEF-formula (4) and GEF-formula (1);

c) Different evaluation functions for LEF/GEF: LEF-formula (3) and LEF-formula (4), GEF-formula (1), (2) and (9);

d) Different exploration heuristics for the same evaluation function:
   GEF-GR-formula (1) , GEF-SA-formula (1) and LEF-formula (3)

<table>
<thead>
<tr>
<th>Problem</th>
<th>n</th>
<th>k</th>
<th>Measurement</th>
<th>LEF (3)</th>
<th>GEF-GR (1)</th>
<th>GEF-SA (1)</th>
<th>LEF (4)</th>
<th>GEF-SA (9)</th>
<th>GEF-SA (2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>anna</td>
<td>138</td>
<td>11</td>
<td>runtime</td>
<td>0*</td>
<td>0.03</td>
<td>0</td>
<td>0</td>
<td>0.03</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>3174</td>
<td>1049</td>
<td>37070</td>
<td>2070</td>
<td>75218</td>
<td>6304</td>
</tr>
<tr>
<td>david</td>
<td>87</td>
<td>11</td>
<td>runtime</td>
<td>0.01</td>
<td>0</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>957</td>
<td>519</td>
<td>15968</td>
<td>2697</td>
<td>9316</td>
<td>26974</td>
</tr>
<tr>
<td>homer</td>
<td>561</td>
<td>13</td>
<td>runtime</td>
<td>0.04</td>
<td>0.66</td>
<td>0.09</td>
<td>0</td>
<td>0.17</td>
<td>0.15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>17952</td>
<td>7272</td>
<td>194303</td>
<td>52173</td>
<td>345592</td>
<td>272292</td>
</tr>
<tr>
<td>huck</td>
<td>74</td>
<td>11</td>
<td>runtime</td>
<td>0</td>
<td>0</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>148</td>
<td>27</td>
<td>8816</td>
<td>148</td>
<td>950</td>
<td>942</td>
</tr>
<tr>
<td>jean</td>
<td>80</td>
<td>10</td>
<td>runtime</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>240</td>
<td>108</td>
<td>9096</td>
<td>480</td>
<td>2212</td>
<td>1750</td>
</tr>
<tr>
<td>games120</td>
<td>120</td>
<td>9</td>
<td>runtime</td>
<td>0</td>
<td>0</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>240</td>
<td>58</td>
<td>17505</td>
<td>240</td>
<td>1783</td>
<td>1797</td>
</tr>
<tr>
<td>miles250</td>
<td>128</td>
<td>8</td>
<td>runtime</td>
<td>0.11</td>
<td>0.02</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.87</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>69760</td>
<td>591</td>
<td>15656</td>
<td>9728</td>
<td>6343</td>
<td>2000564</td>
</tr>
<tr>
<td>miles500</td>
<td>128</td>
<td>20</td>
<td>runtime</td>
<td>0.01</td>
<td>0.04</td>
<td>0.03</td>
<td>0.02</td>
<td>0.37</td>
<td>3.21</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>2304</td>
<td>1419</td>
<td>60793</td>
<td>38656</td>
<td>625829</td>
<td>2716181</td>
</tr>
</tbody>
</table>

* The measurement unit in the experiments is 1 second. If the CPU runtime is less than 1 second, it reports 0.
<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>miles750</td>
<td>128</td>
<td>31</td>
<td>runtime</td>
<td>0</td>
<td>0.4</td>
<td>0.18</td>
<td>0.19</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>896</td>
<td>8482</td>
<td>29982</td>
<td>602240</td>
</tr>
<tr>
<td>miles1000</td>
<td>128</td>
<td>42</td>
<td>runtime</td>
<td>0.04</td>
<td>0.61</td>
<td>0.33</td>
<td>0.63</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>30720</td>
<td>11976</td>
<td>646690</td>
<td>1503744</td>
</tr>
<tr>
<td>miles1500</td>
<td>128</td>
<td>73</td>
<td>runtime</td>
<td>0</td>
<td>0.1</td>
<td>0.18</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>2816</td>
<td>1343</td>
<td>271035</td>
<td>25216</td>
</tr>
<tr>
<td>queen5_5</td>
<td>25</td>
<td>5</td>
<td>runtime</td>
<td>0</td>
<td>0.07</td>
<td>0.01</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>250</td>
<td>32172</td>
<td>16970</td>
<td>350</td>
</tr>
<tr>
<td>queen6_6</td>
<td>36</td>
<td>7</td>
<td>runtime</td>
<td>0.01</td>
<td>0.56</td>
<td>0.26</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>16488</td>
<td>138624</td>
<td>609253</td>
<td>53352</td>
</tr>
<tr>
<td>queen7_7</td>
<td>49</td>
<td>7</td>
<td>runtime</td>
<td>0</td>
<td>256.71</td>
<td>4.86</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>22736</td>
<td>44497823</td>
<td>9776836</td>
<td>50323</td>
</tr>
<tr>
<td>queen8_12</td>
<td>96</td>
<td>12</td>
<td>runtime</td>
<td>0.01</td>
<td>0.27</td>
<td>0.05</td>
<td>0.38</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>28320</td>
<td>19492</td>
<td>104292</td>
<td>1811904</td>
</tr>
<tr>
<td>queen8_8</td>
<td>64</td>
<td>9</td>
<td>runtime</td>
<td>0.03</td>
<td>32.28</td>
<td>1.62</td>
<td>0.13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>92352</td>
<td>4077409</td>
<td>3251627</td>
<td>655872</td>
</tr>
<tr>
<td>queen9_9</td>
<td>81</td>
<td>10</td>
<td>runtime</td>
<td>0.05</td>
<td>68.71</td>
<td>1.5</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>142722</td>
<td>6618526</td>
<td>3416765</td>
<td>1594404</td>
</tr>
<tr>
<td>fpsol2.i.1</td>
<td>496</td>
<td>65</td>
<td>runtime</td>
<td>-</td>
<td>5.21</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>-</td>
<td>22607</td>
<td>-</td>
<td>1488</td>
</tr>
<tr>
<td>fpsol2.i.2</td>
<td>451</td>
<td>30</td>
<td>runtime</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.02</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>58179</td>
</tr>
<tr>
<td>fpsol2.i.3</td>
<td>425</td>
<td>30</td>
<td>runtime</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>57375</td>
</tr>
<tr>
<td>inithx.i.1</td>
<td>864</td>
<td>54</td>
<td>runtime</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>25056</td>
</tr>
<tr>
<td>inithx.i.2</td>
<td>645</td>
<td>31</td>
<td>runtime</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>6450</td>
</tr>
<tr>
<td>inithx.i.3</td>
<td>621</td>
<td>31</td>
<td>runtime</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>4347</td>
</tr>
<tr>
<td>mulsol.i.1</td>
<td>197</td>
<td>49</td>
<td>runtime</td>
<td>0.01</td>
<td>0.06</td>
<td>0.33</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>22261</td>
<td>732</td>
<td>734416</td>
<td>394</td>
</tr>
<tr>
<td>mulsol.i.2</td>
<td>188</td>
<td>31</td>
<td>runtime</td>
<td>-</td>
<td>0.2</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>-</td>
<td>2976</td>
<td>-</td>
<td>376</td>
</tr>
<tr>
<td>mulsol.i.3</td>
<td>184</td>
<td>31</td>
<td>runtime</td>
<td>-</td>
<td>0.15</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>-</td>
<td>2662</td>
<td>-</td>
<td>552</td>
</tr>
<tr>
<td>mulsol.i.4</td>
<td>185</td>
<td>31</td>
<td>runtime</td>
<td>-</td>
<td>0.16</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>-</td>
<td>2567</td>
<td>-</td>
<td>370</td>
</tr>
<tr>
<td>mulsol.i.5</td>
<td>186</td>
<td>31</td>
<td>runtime</td>
<td>-</td>
<td>0.15</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#move</td>
<td>-</td>
<td>2674</td>
<td>-</td>
<td>372</td>
</tr>
<tr>
<td>zeroin.i.1</td>
<td>211</td>
<td>49</td>
<td>runtime</td>
<td>40.27</td>
<td>0.27</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 7. Avg. CPU runtime (second) and Avg. number of agent move of 100 runs for \( k \)-Coloring Problems. Row \( n \) shows the number of nodes in the problem; Row \( k \) shows the number of colors; \('-'\) means it can not find one solution within 6000 seconds. For each instance, the upper row shows CPU runtime and the lower row shows the number of agent move. Bold runtime numbers indicate the best among all the algorithms and evaluation functions.

3.3.2. Minimum Coloring Problem
Finding the chromatic number of the graph in a Minimum Coloring Problem is a global goal. The LEF of each agent can only use the local information to guide itself and sometimes there is not enough information to make a good decision. We will test the CPU runtime (second) and number of agent moves for finding a \( \chi(G) \)-color solution when \( \chi(G) \) is unknown. The system will try using a large number \( m \) colors, and decreases it if a feasible \( m \)-color solution is found, until \( m \) is equal to \( \chi(G) \). We will compare performances of the following cases:

a) Consistent case: LEF-formula (4) and GEF-formula (9)
b) Inconsistent case: LEF-formula (4) and GEF-formula (2) and (12)
c) Different GEFs: GEF-formula (2), (9) and (12);
d) Different exploration heuristics for the same evaluation function:
   GEF-GR-formula (2) , GEF-SA-formula (2)
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>runtime</th>
<th>0</th>
<th>0</th>
<th>0.01</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>huck</strong></td>
<td>74</td>
<td>11</td>
<td></td>
<td>0 144</td>
<td>7525</td>
<td>69 4744</td>
<td>70745</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>jean</strong></td>
<td>80</td>
<td>10</td>
<td></td>
<td>0 182</td>
<td>8495</td>
<td>140 5389</td>
<td>272143</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>games120</strong></td>
<td>120</td>
<td>9</td>
<td></td>
<td>0 144</td>
<td>7525</td>
<td>69 4744</td>
<td>70745</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>miles250</strong></td>
<td>128</td>
<td>8</td>
<td></td>
<td>0.01 17969</td>
<td>2408174</td>
<td>38059 2680230</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>miles500</strong></td>
<td>128</td>
<td>20</td>
<td></td>
<td>0.02 49997</td>
<td>4330434</td>
<td>5240 930225</td>
<td>819209</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>miles750</strong></td>
<td>128</td>
<td>31</td>
<td></td>
<td>0.33 837815</td>
<td>92804870</td>
<td>42023 3022937</td>
<td>395228</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>miles1000</strong></td>
<td>128</td>
<td>42</td>
<td></td>
<td>0.05 315503</td>
<td>180942683</td>
<td>128155 4305429</td>
<td>415805</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>miles1500</strong></td>
<td>128</td>
<td>73</td>
<td></td>
<td>0.03 28462</td>
<td>167743</td>
<td>1979 26522</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>queen5_5</strong></td>
<td>25</td>
<td>5</td>
<td></td>
<td>0.1 722</td>
<td>4383</td>
<td>18628 2754</td>
<td>6402</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>queen6_6</strong></td>
<td>36</td>
<td>7</td>
<td></td>
<td>0.05 378589</td>
<td>90174885</td>
<td>354197 5760684</td>
<td>4096577</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>queen7_7</strong></td>
<td>49</td>
<td>7</td>
<td></td>
<td>0.1 144792</td>
<td>54526453</td>
<td>20349212 5861025</td>
<td>2845785</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>queen8_12</strong></td>
<td>96</td>
<td>12</td>
<td></td>
<td>8.55 30930691</td>
<td>61457752</td>
<td>16572598 24723491</td>
<td>170069666</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>queen8_8</strong></td>
<td>64</td>
<td>9</td>
<td></td>
<td>0.02 199061459</td>
<td>111757232</td>
<td>38294560 37049734</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>queen9_9</strong></td>
<td>81</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>fpsol2.i.1</strong></td>
<td>496</td>
<td>65</td>
<td></td>
<td>0.05 918</td>
<td>321354</td>
<td>476 319550</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>fpsol2.i.1</strong></td>
<td>451</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>fpsol2.i.1</strong></td>
<td>425</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>inithx.i.1</strong></td>
<td>864</td>
<td>54</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>inithx.i.2</strong></td>
<td>645</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>inithx.i.3</strong></td>
<td>621</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>mulso1.i.1</strong></td>
<td>197</td>
<td>49</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>#move</td>
<td>runtime</td>
<td>0</td>
<td>0.13</td>
<td>0.04</td>
<td>0.05</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>-------</td>
<td>---------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>mulsol.i.2</td>
<td>188</td>
<td>31</td>
<td>0</td>
<td>0.13</td>
<td>0.04</td>
<td>0.05</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>mulsol.i.3</td>
<td>184</td>
<td>31</td>
<td>0</td>
<td>0.14</td>
<td>0.03</td>
<td>0.04</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>mulsol.i.4</td>
<td>185</td>
<td>31</td>
<td>0</td>
<td>0.15</td>
<td>0.03</td>
<td>0.04</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>mulsol.i.5</td>
<td>186</td>
<td>31</td>
<td>0</td>
<td>0.14</td>
<td>0.04</td>
<td>0.03</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>zeroin.i.1</td>
<td>211</td>
<td>49</td>
<td>0</td>
<td>0.11</td>
<td>0.07</td>
<td>0.04</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>zeroin.i.2</td>
<td>211</td>
<td>30</td>
<td>0</td>
<td>0.31</td>
<td>0.07</td>
<td>0.06</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>zeroin.i.3</td>
<td>206</td>
<td>30</td>
<td>0</td>
<td>0.22</td>
<td>0.11</td>
<td>0.11</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>myciel3</td>
<td>11</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>myciel4</td>
<td>23</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>myciel5</td>
<td>47</td>
<td>6</td>
<td>0</td>
<td>0.02</td>
<td>0</td>
<td>0</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>myciel6</td>
<td>95</td>
<td>7</td>
<td>0</td>
<td>0.03</td>
<td>0.01</td>
<td>0</td>
<td>843.8</td>
<td></td>
</tr>
<tr>
<td>myciel7</td>
<td>191</td>
<td>8</td>
<td>0</td>
<td>0.04</td>
<td>0.06</td>
<td>0.01</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Table 8. Avg. CPU runtime (second) and avg. number of agent moves of 100 runs for solving the Minimum Coloring Problems. Row n shows the number of nodes in Coloring Problems; Row χ shows the chromatic number; '-' means it can not find one solution within 6000 seconds. For each instance, the upper row shows CPU runtime and the lower row shows the number of agent moves. Bold runtime numbers indicate the best among all the algorithms and evaluation functions.

### 3.3.3. Observations from Experiments

1. **LEF works for a k-Coloring Problem and a Minimum Coloring Problem.** The GEF does not always lead the evolution more efficiently than the LEF although the GEF uses more information to make decision and has centralized control for the whole system. In the k-Coloring Problem, the LEF, especially LEF-formula (4), beats the GEFs in CPU runtime for almost all instances; In the Minimum Coloring Problems, although the LEF only knows local information and the purpose is global, the LEF still beats the GEFs for most instances except queen8_12 and queen9_9. For instances of homer, miles250, queen7_7, queen8_8, fpsol2, initix, mulsol and zeroin, the LEF is much faster than GEFs. Although GEF-formula (9) is consistent with LEF-formula (4) and GEF-formula (2) works better than GEF-formula (9), LEF-formula (4) still beats GEF-formula...
Local and Global Evaluation Function for Evolution 31

(2). In addition to Coloring Problems, the LEF also works for N-queen Problems [1][2][3]. LEF for N-queen problem can beat a lot of algorithms, but is not as faster as one specified local search heuristic [10]. One important reason is the specified local search heuristic embodies preliminary knowledge of the N-queen Problem. For more experimental results, such as distributions and deviations of runtime, and performance based on different parameter settings, please read our papers [1][2][3].

(2) Evaluation function affects performance. As we know, the definition of GEF will affect the performance for solving function optimization [37]. This is also true for combinatorial problems. For the k-Coloring Problem (decision problem): the performances of LEF- formula (4) and LEF-formula (3) behave very differently. In fpsol2, initlx, mulsol and zeroin, LEF- formula (4) beats LEF- formula (3) strongly, while LEF- formula (3) slightly beats LEF- formula (4) in miles and queern; GEF-formula (2) can solve fpsol2, initlx, mulsol and zeroin less than 2 seconds, while GEF- formula (1) can not solve using 6000 seconds. We guess that is because part (a) of formula (2) gives a pressure to each node to select a small index number color, which helps to break the symmetry [33] in the coloring problems [34]. In the Minimum Coloring Problem (optimization problem), it is very obvious: GEF-formula (12) can not solve more than half of those instances, while the others can solve; GEF-formula (2) works better than GEF-formula (9) on average.

(3) Performance of LEF and GEF are more alike if LEF is consistent with GEF. In the k-Coloring Problem, LEF-formula (4) is consistent with GEF-formula (9) while LEF-formula (3) is consistent with GEF-formula (1), so performance between LEF- formula (3) and GEF- formula (1) are more alike than the performances of LEF-formula (4) and GEF-formula (9). LEF-formula (4) and GEF-formula (9) can solve fpsol2, initlx, mulsol and zeroin quickly while LEF formula (3) and GEF- formula (1) can not. LEF formula (3) and GEF- formula (1) beat LEF-formula (4) and GEF-formula (9) in miles1000 and queern8_12. In the Minimum Coloring Problem, for example, the queen8_12 is difficult for LEF-formula (4) and GEF-formula (9), but easy for GEF-formula (2). These results tell us that if the Nash Equilibria of LEF are identical to the local optima of the GEF, they will behave more alike.

(4) Exploration heuristics affect performance. There are so many efforts on exploration heuristics that we know the exploration heuristics will affect the performance. This is also true for the consistent LEF and GEF: even though they have the same evaluation function (map), they still work differently in some instances. For a k-Coloring Problem, LEF- formula (3) beats GEF-formula (1) in miles and queens while GEF-GR-formula (1) beats LEF- formula (3) and GEF-SA-formula (1) in mulsol and zeroin; LEF- formula (4) and GEF-SA- formula (9) are consistent but LEF works faster than GEF-SA: For the Minimum Coloring Problems, LEF- formula (4) and GEF-formula (9) are consistent but LEF- formula (4) beats GEF-formula (9) strongly; Both using GEF-formula (2), performances of SA and GR are different especially in queen7_7, queen8_8 and queen9_9.

(5) For an agent movement, larger complexity does not always means better performance. We have the ranking for complexities of an agent move in different algorithms: GEF-GR>LEF>GEF-GR. However, GEF-GR is not always the one that uses the fewest agent moves to reach the (optimal) solution state of Coloring Problems, even in the consistent case. In both k-coloring problems and Minimum Coloring Problems, the LEF uses fewer moves than GEF-GR in some instances, such as queen5_5 and queen7_7. So this suggests that in some instances using local information does not always mean decreasing the quality of the decision. Even using the same evaluation function, GEF-GR looks at n neighbors and selects one while GEF-SA only looks at one neighbor, we still see in some problem instances, such as queen5_5, queen7_7 and queen9_9, GEF-SA uses fewer moves than GEF-GR.
Performs of the LEF and GEF are problem instance dependent. The LEF fits for most instances except *queen8_12* in the Minimum Coloring Problem. GEF-formula (1) and LEF-formula (3) are not good for *fpsol2*, *initlx*, *mulso1* and *zeroin*. As in our other paper [3], the algorithmic details will not change the performance for a problem instance very much. We guess the graph structure (constraint network) will have a big impact on the performances of the LEF and GEF. In other words, the LEF might be good for some networks while the GEF might be good for some other networks. In the language of multi-agent systems, some interaction structures among agents are more suitable to use self-organization to reach to a global purpose while some interaction structures favor centralized control. To find out what the ‘good’ networks for the LEF and the GEF are will be a very important future topic.

4. CONCLUSIONS and DISCUSSIONS

This paper proposes a new look at the evolution of solution for combinatorial problems, using NP-complete cases of the *k*-Coloring Problem (*Decision problem*) and the Minimum Coloring Problem (*Optimization problem*). The LEF&GEF outlook suggests several insights and suggests several lines of future work and applications.

We first briefly reviewed the current algorithms and classify them by ‘Single or multi-solutions’ and ‘Systematic search or Generate-test’ (Fig.2). Then we limited our discussion to algorithms in the intersection of Single-solution and Generate-test, called ‘GT-SS’ algorithms, because finding solutions using those algorithms is like the evolution of the multi-agent system when the Coloring Problem is represented in the Multi-agent system framework. The GT-SS algorithms (Fig.2) for solving combinatorial problems can be divided into two parts:

\[
\text{GT-SS Algorithms} = \text{Evaluation Function} + \text{Exploration Heuristics}
\]

This paper focuses on the *evaluation function*: the internal ranking of all configurations. That is, the paper focuses on the evaluation of all configurations. The *Evaluation function* is the more basic part because the exploration is based on the comparison of evaluation value of configurations.

We point out that there are two ways to construct evaluation functions: the Global Evaluation Function (GEF) and the Local Evaluation Function (LEF). The GEF uses global information to evaluate a whole system state, such as is done in Simulated Annealing. The LEF uses local information to evaluate a single agent state, as is done in Alife&AER models. The computer experiments are done on benchmarks for the *k*-Coloring Problem and the Minimum Coloring Problem. We compare Simulated Annealing (GEF-SA), Greedy-random (GEF-GR) and AER model (LEF). This shows how LEF works for the *k*-Coloring Problem and the Minimum Coloring Problem. The results demonstrate that using local information to guide each agent can efficiently lead the whole system to a global goal in many instances. Moreover, even though the computational complexity of an agent move in LEF is smaller than in GEF-GR, LEF uses fewer moves than GEF-GR in some instances. So doing more computation and using more information selecting a configuration does not always mean better performance for the whole run. Using local information does not always mean decreasing the quality of decision. This encourages us to apply LEF in more problems and systems in the future. From the experiments, we also found that the performance of the LEF and GEF are problem instance dependent. This indicates the applicability of LEF and GEF is important, and it might relate to the constraint network, such as the graphs of Coloring Problem. Using network dynamic theory [47][48][49] to study the good instances for LEF will be one of our future investigations.
The way to construct the evaluation function for a problem is not unique. We give examples in section 3.1: formula (1) – a GEF for k-Coloring Problem; formula (2) – a GEF for Minimum Coloring Problem; formula (3) – an LEF for k-Coloring Problem; formula (4) – an LEF for Minimum Coloring Problem. We also construct other evaluation functions: formula (9) and formula (12) – GEFs for Minimum Coloring Problem; actually the GEF formula (2) and the LEF formula (4) can be used for k-Coloring Problem too. The experimental results show that the evaluation function affects performance, so defining a good evaluation function is very important. There have been many investigations of exploration heuristics, but not so much on the evaluation functions for combinatorial problems. This indicates an important direction for improving algorithm for combinatorial problems: Discover good evaluation functions for a problem and exploit phase transitions and correlations [51] of different evaluation functions. It also gives us the possibility of constructing the LEF since the evaluation function is not unique for a problem. We will discuss how to construct LEF in a future paper.

We also analyze the relationship between LEF and GEF in terms of consistency and inconsistency. Table 5 summarizes the differences between GEF and consistent/inconsistent LEF:

(I) Consistency (definition 1) means LEF and GEF will rank two neighboring configurations (N1 neighborhood structure) in the same order. A good agent decision that decreases the LEF value is also a good decision for the whole system decreasing the GEF value. So LEF and GEF will make the same decision for choosing a neighboring configuration during the evolution. We propose a Consistency Theorem which shows that Nash Equilibria (definition 4) of an LEF are identical to the local optima (definition 3) of a GEF if the LEF is consistent with the GEF. In the spin glass theory, this theorem means the LEF and the GEF have the same ground state structure and the same number of solutions, as well as the same phase transitions between satisfied and unsatisfied [38]. So the LEF can be partly explained by the current theoretical results on traditional GEF [6][19][29][32]. This helps us to understand why the LEF can guide agents to evolve to a global goal state if the global goal can be expressed by a GEF that is consistent with the LEF. When an LEF is inconsistent with a GEF, the LEF’s Nash Equilibria are not identical to the GEF’s local optima. Obviously, the LEF evolution proceeds differently than the GEF evolution. Experimental results support this; performance of LEF and GEF are more alike if the LEF is consistent with the GEF.

(II) Since the way to construct a GEF is not unique for a problem, we can always construct a GEF by summing up all agents’ LEF as \( E_{GEF} = \sum E_{LEF} \). \( \sum E_{LEF} \) can be consistent with \( E_{LEF} \). For example, LEF- formula (4) and LEF-formula (3) are consistent LEF (consistent with \( \sum E_{LEF} \). However, \( \sum E_{LEF} \) also can be inconsistent with \( E_{LEF} \) as the paper shows in Fig.10 (Prisoner Dilemma problem). This paper gives a preliminary theorem for constructing consistent LEF which have the form of the spin glass Hamiltonian (formula 11): If the penalty function \( g_{ij}(X_i, X_j) \) in LEF is a Symmetrical penalty, the LEF is consistent with \( \sum E_{LEF} \). More study on consistent and inconsistent LEF should give more insights into computation and game theory.

(III) Even though an LEF is consistent with a GEF, they still explore differently. Searching (the GT-SS style) for solution is just like walking in dark night using a flashlight looking at a map; the evaluation function is the generator of the map; heuristics are for exploring the mapped space. The GEF uses one n-dimensional map while the LEF uses n partial maps: \( d_i \)-dimensional map for agent \( a_i \). GEF methods explore in an n-dimensional space and use a neighborhood-flashlight (Fig.12); Agents in LEF methods explore in n \( d_i \)-dimensional spaces and use a slice-flashlight (Fig. 13). If an LEF is a consistent LEF, those \( n d_i \)-dimensional maps are slice cuts of the map of \( E_{GEF} = \sum E_{LEF} \). But explorations of LEF \( E_{LEF} \) and GEF \( E_{GEF} = \sum E_{LEF} \) are different: GEF searches a neighborhood
(neighborhood-flashlight) and LEF searches from slice to slice (slice-flashlight). Based on the partial map which is highlighted by the flashlight, an LEF or GEF will make a decision according to its strategy: Greedy or Simulated Annealing, or random... The experimental results show that exploration heuristics affect performance, so even though the LEF is consistent with GEF, LEF still behaves differently than GEF and works better in some instances.

The LEF and GEF concepts provide both a new view of algorithms for Combinatorial Problems, and for the collective behavior of complex systems. Using Coloring Problem clarifies the concepts of ‘Local’ and ‘Global’. These concepts are important distinguishing characteristics of Distributed systems and Centralized systems. Obviously, Combinatorial Problems provide a good case for studying ‘Local’ and ‘Global’ because the problem can be modeled as a multi-agent system with network structural interactions. Combinatorial problems have been translated to spin glass models and studied by physicists for collective behaviors [6]. Actually, I think the LEF & GEF concept might also be a natural concept for studying the evolution of other systems: Physics (Spin glass), Biology (NK model [50]), Economics (Game theory), etc. Boid and Sandpile are models that use LEF to evolve. If we can find a consistent GEF for them, we will understand more about the collective behavior of complex systems.

This paper is just the beginning of LEF&GEF studies. In addition to the research report above, there is still a lot of future work: more experiments on LEF methods; analytical study on LEF; sufficiency of local information for LEF; and the existence of a consistent GEF for any LEF; Is the consistency concept sufficient? Since Genetic Algorithms also have an evaluation function (fitness function), can we apply LEF&GEF to Genetic Algorithms? ... It is our intention to study and attempt to answer all of these questions.

ACKNOWLEDGMENTS

This paper is supported by the International Program of the Santa Fe Institute. The author wants to thank Prof. Lei Guo and Prof. John Holland for their strong support and discussions. Thanks to Mark Newman, Eric Smith, Haitao Fang, Cris Moore, Supriya Krishnamurthy, Dengyong Zhou, Li Ming, Melanie Mitchell, Jose Lobo, John Miller, Sam Bowles, Jim Crutchfield, Chien-Feng Huang and other people for helpful discussions. Thanks to Laura Ware for proofreading.

REFERENCES

42. http://www.red3d.com/cwr/boids/applet