International Journal of Pattern Recognition and Artificial Intelligence
Vol. 16, No. 8 (2002) 1041–1064

DISTRIBUTED PROBLEM SOLVING WITHOUT COMMUNICATION — AN EXAMINATION OF COMPUTATIONALLY HARD SATISFIABILITY PROBLEMS

JIMING LIU* and XIAOLONG JIN†

Department of Computer Science,
Hong Kong Baptist University, Kowloon Tong, Hong Kong
**jiming@comp.hkbu.edu.hk*
†*jxl@comp.hkbu.edu.hk*

JING HAN

Department of Computer Science,
University of Science and Technology of China, Hefei, China

In this paper, we extend and modify the ERA approach proposed in Ref. 13 to solve Propositional Satisfiability Problems (SATs). The new ERA approach involves a multiagent system where each agent only senses its local environment and applies some self-organizing rules for governing its movements. The environment, which is a two-dimensional cellular environment, records and updates the local values that are computed and affected according to the movements of individual agents. In solving a SAT with the ERA approach, we first divide variables into several groups, and represent each variable group with an agent whose possible positions correspond to the elements in a Cartesian product of variable domains, and then randomly place each agent onto one of its possible positions. Thereafter, the ERA system will keep on dispatching agents to choose their movements until an exact or approximate solution emerges. The experimental results on some benchmark SAT test-sets have shown that the ERA approach can obtain comparable results as well as stable performances for SAT problems. In particular, it can find approximate solutions for SAT problems in only a few steps. The real value of this approach is that it is a distributed asynchronous approach without any centralized control or evaluation, where the agents can cooperate to solve problems without explicit communication.

Keywords: Distributed problem solving; propositional satisfiability problem (SAT); self-organization; multiagent system; ERA.

1. Introduction

1.1. *Propositional Satisfiability Problem* (*SAT*)

A *Propositional Satisfiability Problem* (SAT) is an NP-complete problem. Many issues in Artificial Intelligence (AI) as well as in other areas of computer science and

*Author for correspondence.

engineering can be formulated into *Propositional Satisfiability Problems* (SAT).[3,4] Some examples of such problems include: spatial and temporal planning, qualitative and symbolic reasoning, computational linguistics, scheduling, resource allocation and planning, graph problems, and circuit diagnosis.

1.1.1. *Definitions*

Generally speaking, a *Propositional Satisfiability Problem* (SAT) is to test whether or not there exists (at least) one solution for a given propositional formula.

Definition 1. A *Propositional Satisfiability Problem* (SAT), P, consists of:

1. A finite set of propositional variables, $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$.
2. A domain set, $\mathbf{D} = \{D_1, D_2, \ldots, D_n\}$, for all $i \in [1, n]$, $X_i \in D_i$ and $D_i = \{\text{True}, \text{False}\}$.
3. A clause set, $\mathbf{CL} = \{Cl(R_1), Cl(R_2), \ldots, Cl(R_m)\}$, where each R_i is a subset of X, and each clause $Cl(R_i)$ is a disjunction of the literals corresponding to the variables in set R_i.

Definition 2. The solution, S, of a SAT is an assignment to all variables such that, under this assignment, the truth values of all given clauses are true, i.e.

1. S is an ordered set, $S =< v_1, v_2, \ldots, v_n >$, for all $i \in [1, n]$, v_i is equal to True or False, $S \in D_1 \times D_2 \times \cdots \times D_n$.
2. $\forall j \in [1, m], T(Cl(R_j)) = \text{True}$, where $T(\cdot)$ is a function that returns the truth value of a clause.

1.1.2. *Conventional methods*

Generally speaking, the methods to solve a SAT can be divided into two main categories: *systematic* methods and *local search* methods.[10] A *systematic* method is a traditional way to solve SATs that allots a value to a selected variable at each step and then checks if there are some clauses unsatisfied. If this is the case, it will backtrack to a previous variable to assign it with another value, and then repeat this process. Or else, it will select a new variable to branch until a solution is found or the problem is unsatisfiable. This method will walk through the search space, so it is a complete method, i.e. it can obtain a "it is satisfiable" or "it is unsatisfiable" result for a given problem. Some examples of *systematic* method are POSIT, TABLEAU, GRASP, SATZ, and REL_SAT.[10]

Compared to *systematic* methods, *local search* is a relatively new method, which appeared in 1992 when Selman *et al.*[20] and Gu[6] almost simultaneously and independently proposed it. The local search procedure starts with a complete, randomly initialized assignment, then checks if it satisfies all clauses. If not, it will randomly or heuristically select a variable to flip (i.e. change its value). It repeats this process until a solution is found. Local search has three key elements,[1] they are:

1. *Configuration*: one possible assignment of all variables, not required to be a solution.
2. *Evaluation value*: the number of unsatisfied clauses.
3. *Neighbor*: the configuration obtained by flipping the assignment of a variable in the current configuration.

During the past, local search has been shown to outperform *systematic* methods. But, it is incomplete in nature. It cannot prove that a propositional formula has no satisfying assignment. In addition, it cannot guarantee that it will find a solution for a satisfiable formula. Despite this, many improvements have been introduced to local search. As a result, there are two main streams in local search; namely, GSAT[6,20] and WalkSAT.[19] Both of them have many variants, such as, GWSAT,[19] GSAT/Tabu,[16,21] HSAT,[4] HWSAT[5] of GSAT and WalkSAT/Tabu,[17] Novelty,[17] R-Novelty[17] of WalkSAT.

1.2. *Self-organizing system*

The term "self-organization" was first introduced by W. R. Ashby in 1947.[a] The phenomenon of self-organization exists in a variety of natural systems and scientific fields, such as galaxies, planets, biology, chemistry, computer science, geology, sociology and economy, etc.[2,8,18,22,24]

A self-organizing system consists of two main components: elements and an environment where the elements are situated. A system that is self-organized indicates that the system's elements are autonomous individuals and can behave rationally and independently. And further, there is no explicit outside control on the system, i.e. it is not controlled by outer, top–down rules. Self-organization is actually an "evolutionary" process. During the process, the elements change their behaviors according to the changes occurring on their environment. The behaviors are often complex, but not predefined. In other words, they are emergent. More detailed descriptions on self-organization can be found in Refs. 12, 15, 24 and 25.

Based on the point of view of Ünsal in Ref. 22, one can generalize the actions of elements in a self-organizing system with three steps. First, the elements sense the environment or receive the signals from the other elements. And then, based on the information received from the environment or other elements, the elements make rational decisions, i.e. rationally decide what to do next. Finally, the elements act by their decisions. Their actions will in turn affect the environment and the actions of the other elements. The essence of a self-organized system is interactions between its members and the environment. Through the interactions, a self-organized system can exhibit emergent behaviors.

A self-organizing system can yield a global result through local information exchanges without global control and planning. This is a very interesting system

[a]But some people thought that it was Farley and Clark who in 1954 defined this term.

model. Lately, researchers have paid much attention to it. By virtue of self-organization, they have proposed some methods for solving practical problems. Introducing self-organization into neural networks is a successful example. Self-organization has also been used in other fields, such as image feature extraction as described in Ref. 14.

In the SAT related area, there have existed two self-organization based approaches. Inspired by *cellular automata*[7,14] and *swarm*,[23] Liu *et al.* in Ref. 13 proposed an approach, namely ERA, for solving *Constraint Satisfaction Problems* (CSPs). In the ERA approach, each distributed agent represents a variable, and a two-dimensional grid-like environment where all agents live corresponds to the domains of all variables. To solve a given CSP problem, all agents will be distributed in the environment. From then on, agents will move in its local area. Agents can interact with each other via their environment until a special solution state emerges. They have employed the ERA approach to solve two kinds of classical CSPs: *n-queen problems* and *graph coloring problems*. Their experimental results showed that the ERA approach is efficient in solving both types of problems, and can find approximate solutions in just a few steps. Another approach was recently proposed by Hirayama and Yokoo in Ref. 9, called *Multi-DB*. In Multi-DB, each agent has multiple variables and relevant clauses to the variables. To solve a problem, all agents will communicate with its related agents by sending and receiving *OK* and *improve* messages. Through communication, the related agents can negotiate with each other to assign appropriate assignments to their variables so that the assignments to the variables satisfy all clauses.

1.3. *The proposed approach*

In this paper, we will explore the use of the ERA approach in SATs. To do so, we extend and modify it in order to make it more applicable to solving SATs. We will see that, from the point of view of solving a SAT, the new ERA approach somewhat behaves like local search. But, in nature, they are different. The main difference between ERA and local search lies in the following three points:

1. *The evaluation value of ERA is not the number of unsatisfied clauses for the whole assignment as in local search, but rather the number of unsatisfied clauses related to the variables of an agent. These evaluation values constitute an environment in the ERA system.*
2. *The ERA system is concurrent whereas local search is sequential. As if in cellular automata, agents can move and update asynchronously.*
3. *In local search, the neighbors of an assignment are restricted to those that are different with this assignment in the value of only one variable. That is, the radius of neighborhood is one. But, in ERA, we enlarge the radius of neighborhood, it can be larger than one.*

As compared to the conventional methods for solving SATs mentioned in Sec. 1.1.2, our EAR approach has three features:

1. *All agents in ERA are autonomous. No centralized control policy exists to govern the actions of agents.*
2. *Agents know and concern only the clauses related to their own variables. Based on the evaluation value to these clauses, agents select their next movements. So, there is no global evaluation in the ERA approach. But in the conventional methods, the evaluation functions are usually based on all clauses.*
3. *In ERA, all agents can move asynchronously. So, at a certain time, there may be a number of variables that change their values. But, in the conventional methods, there is only one variable changing its value at each step.*

As far as the Multi-DB method mentioned in the previous section, we notice that in both Multi-DB and ERA, each agent represents some variables and related clauses of the variables. But, in Multi-DB, all related agents cooperate by message communication. In ERA, agents cooperate via their respective interactions with their environment rather than explicit communication. So, the ERA approach is particularly suited to distributed applications where the explicit communication is not feasible.

1.4. *The organization of the paper*

The remainder of this paper is organized as follows: Sec. 2 describes the basic ideas behind the ERA self-organization approach. Section 3 presents an illustrative example of solving a SAT problem using ERA. Section 4 describes experiments and observations, and discusses several important issues related to the ERA approach for solving SATs. Finally, Sec. 5 concludes the paper by highlighting the contributions of this work and pointing out the future work on the ERA approach.

2. The ERA Model

In this paper, we will introduce a distributed self-organizing approach to solve SATs. In our case, the domain of a SAT is represented into a multiagent environment. Thus, the problem of finding a solution to the SAT is reduced to that of local behavior-governed movements within such an environment. Specifically, the notions of agent and multiagent system can be defined as follows:

Definition 3. An agent, **a**, is a virtual entity that essentially has the following properties:

1. Be able to live and act in its local environment;
2. Be able to sense its local environment;
3. Be driven by certain goals;
4. Have some behavioral strategies.

Definition 4. A multiagent system is one that contains the following elements:

1. An environment, E, a space in which the agents live;
2. A set of reactive rules, R, governing the interaction between the agents and their environment.
3. A set of agents, $A = \{a_1, a_2, \ldots, a_n\}$.

The goal of this work is to examine how exact or approximate solutions to SATs can be self-organized by a multiagent system consisting of E, R and A.

2.1. *General framework*

The ERA system is meant to be a straightforward framework for interacting agents to achieve a goal. In ERA, we divide variables into groups, and each group can include one or more variable(s). Each agent represents a group of variables. The environment records the number of clause violations of the current state for each combination of values in the Cartesian product that is constructed by the domains of variables in corresponding variable group. So the position of an agent indicates the value combination of its related variables. The agent can move freely within a row and has its own moving strategies. Its goal is to move to a position whose clause violation number is *zero*, we call it *zero-position* (for details see Definition 5). The reactive rules correspond to the schedules for dispatching agents and updating the environment.

ERA can be described as ERA $= \{E, R, A\}$. A solution state in ERA is reached when every agent (variable group) finds its zero-position (consistent values combination) that satisfies all clauses. In other words, a solution in ERA is specified by the positions of distributed agents.

In the following paragraph, we will use an example to illustrate how the ERA model works in solving a SAT.

Example 1. A SAT:

$$\begin{aligned}
\mathbf{X} &= \{X_1, X_2, X_3, X_4\}, \quad n = 4;\\
\mathbf{D} &= \{D_1, D_2, D_3, D_4\}, \quad D_1 = \{True, False\}, \quad D_2 = \{True, False\};\\
&\quad D_3 = \{True, False\}, \quad D_4 = \{True, False\};\\
\mathbf{C} &= \{T(X_1 \vee \neg X_2 \vee X_3) = True, \quad T(X_1 \vee X_2 \vee \neg X_3) = True,\\
&\quad T(X_2 \vee X_3 \vee \neg X_4) = True, \quad T(\neg X_2 \vee \neg X_3 \vee X_4) = True,\\
&\quad T(X_1 \vee X_3 \vee \neg X_4) = True, \quad T(X_1 \vee X_3 \vee X_4) = True,\\
&\quad T(\neg X_1 \vee X_2 \vee \neg X_3) = True, \quad T(\neg X_1 \vee \neg X_2 \vee X_3) = True,\\
&\quad T(\neg X_1 \vee \neg X_2 \vee \neg X_3) = True\}.
\end{aligned}$$

This example can be modeled as a multiagent system as follows. First, we divide four variables into two groups: $\{X_1, X_2\}$ and $\{X_3, X_4\}$. In this case, the

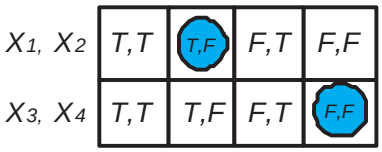

Fig. 1. An illustration of agent model for Example 1.

Cartesian product of each variable group will be $\{\langle True,\ True\rangle, \langle True,\ False\rangle, \langle False,\ True\rangle, \langle False,\ False\rangle\}$. Second, we use two agents to represent the two variable groups. So, the space for agent to move will correspond to the Cartesian product of the variable group (see Fig. 1). In Fig. 1, there are two agents. Each agent occupies a row that is its space to move. The lattices in the row just represent the elements in the Cartesian product of a corresponding variable group.

In Fig. 1, two agents both reside at *zero-positions*. So, Fig. 1 corresponds to a solution state of $S = \langle True, False, False, False\rangle$ to the above SAT.

From the aforesaid example, we can give a general framework for ERA:

SAT{X(variables), D(domain), C(clause)} ⇒ Multi-agent system
D & C ⇒ Environment & Updating rule
X ⇒ Agents (each agent represents a group of variable(s))
Solution ⇒ Positions of agents

2.2. *Environment*

If we divide n variables into u groups (each group may has different number of variables), then an environment, E, has u rows corresponding to the number of variable groups. For all $i \in [1, u]$, if we assume row_i represents a variable group $\{X_{i1}, X_{i2}, \ldots, X_{ik}\}$, then it will have $|D_{i1} \times D_{i2} \times \cdots \times D_{ik}|$ columns. It records two kinds of values: *domain value* and *violation value*.

Definition 5. The data structure of E can be defined as follows:

1. **Size**
   - u rows ⇔ u variable groups. $E = \langle row_1, row_2, \ldots, row_u\rangle$.
   - $\forall i \in [1, u]$, row_i ⇔ all possible value combinations of variables in $\{X_{i1}, X_{i2}, \ldots, X_{ik}\}$ ⇔ $D_{i1} \times D_{i2} \times \cdots \times D_{ik}$, so row_i has $|D_{i1} \times D_{i2} \times \cdots \times D_{ik}|$ columns. $row_i = \langle lattice_{1i}, lattice_{2i}, \ldots, lattice_{(|D_{i1} \times D_{i2} \times \cdots \times D_{ik}|)i}\rangle$.
   - E size is $\sum(|D_{i1} \times D_{i2} \times \cdots \times D_{ik}|)$. $e(j, i)$ refers to the position of $lattice_{ji}$.
2. **Value**
   - **Domain value:** $e(j, i).value$ records the jth combination of values in Cartesian product $D_{i1} \times D_{i2} \times \cdots \times D_{ik}$. It is static since the Cartesian product is static.

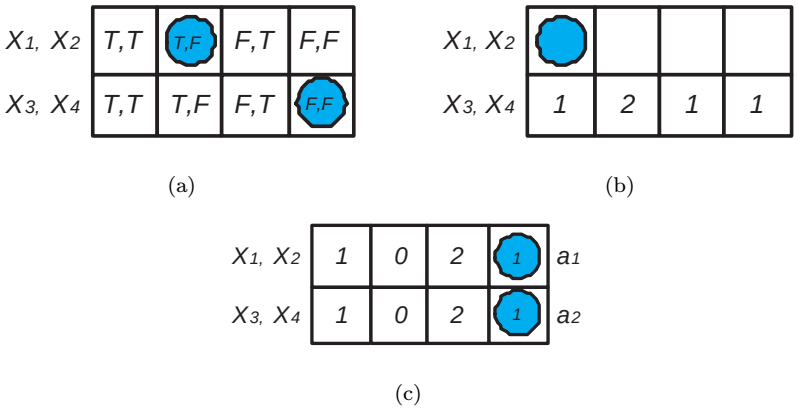

Fig. 2. (a) The representation of domain values, (b) violation numbers if agent a_1 is placed on (1, 1), and (c) violation numbers of the whole environment.

- **Violation number:** $e(j,i).violation$ records in the current state how many clauses were not satisfied, which are related to variables in position $e(j,i)$, i.e. $e(j,i).violation = m$ means there are m clauses, which include some variable(s) in position $e(j,i)$, is unsatisfied. These values are dynamic since the agents keep on moving and their corresponding state is changing. After each movement of one agent, the violation numbers should be updated by applying an updating-rule, which will be described in detail in Sec. 2.4.
- **Zero-position:** position (j,i), in which $e(j,i).violation = 0$. That means all clauses to which the variables in row i are related are satisfied.

For instance, we can further model Example 1 using the above concepts. In Example 1, we have divided four variables into two groups: $\{X_1, X_2\}$ and $\{X_3, X_4\}$. The Cartesian product of each group is $\{\langle True, True\rangle, \langle True, False\rangle, \langle False, True\rangle, \langle False, False\rangle\}$. Figure 2(a) shows the domain value of each lattice in Example 1. Figure 2(b) shows that agent a_1 stays at (1, 1), which means $X_1 = True, X_2 = True$. According to the clause set, if a_2 stays at (1, 2), the clause $T(\neg X_1 \vee \neg X_2 \vee \neg X_3) = True$, which is constructed by variables X_1, X_2 in group $\{X_1, X_2\}$ and variable X_3 in group $\{X_3, X_4\}$, will be violated. If a_2 stays at (2, 2), two clauses $T(\neg X_2 \vee \neg X_3 \vee X_4) = True$, and $T(\neg X_1 \vee \neg X_2 \vee \neg X_3) = True$ will be violated. If a_2 stays at (3, 2), it will violate clause $T(\neg X_1 \vee \neg X_2 \vee X_3) = True$. And if a_2 stays at (4, 2), the clause $T(\neg X_1 \vee \neg X_2 \vee X_3) = True$ will be violated again. So, the violation number of each lattice in *row 2* is 1, 2, 1, and 1, respectively. Figure 2(c) presents a snapshot for the state of the system with the violation numbers. Since there are two agents at positions with violation number 1, it is not a solution state.

2.3. *Agents*

All agents inhabit in an environment, in which their positions indicate value combinations of a variable group. During the operation of the system, the agents will keep

on moving, based on certain moving strategies. At each time step, the positions of the agents provide a consistent or inconsistent assignment for all variables. The agents are trying to find better positions that can lead them to a solution state.

Here is a summary of some main polices for agents in the ERA model:

1. $\forall\, i \in [1, u]$, a_i represents a variable group $\{X_{i1}, X_{i2}, \ldots, X_{ik}\}$.
2. Agent lives and moves in environment E. Agent a_i lives in row_i. It can only move to its right or left, cannot move up or down. $a_i.x$ represents its x-coordinate, which is corresponding to the jth combination of values in $D_{i1} \times D_{i2} \times \cdots \times D_{ik}$. So the position of a_i can be denoted as $(a_i.x, i)$. In this paper, we use function Ψ to define the movement of an agent.

Definition 6. $\Psi : [1, u] \times [1, |D_{i1} \times D_{i2} \times \cdots \times D_{ik}|] \to [1, |D_{i1} \times D_{i2} \times \cdots \times D_{ik}|]$. $\Psi(x, y)$ gives the x-coordination of the new position of agent a_y, after it moves from position of (x, y). So the new position can be represented as $(\Psi(x, y), y)$.

3. In any state of the system, positions of all agents indicate an assignment for all variables. $\forall\, i \in [1, u], e(a_i.x, i).value = \langle v_{i1}, v_{i2}, \ldots, v_{ik}\rangle$, that means $X_{i1} = v_{i1}, X_{i2} = v_{i2}, \ldots, X_{ik} = v_{ik}$. By extracting positions of all agents, we can obtain a complete assignment to all variables. Of course, it may not be a consistent assignment, i.e. not a solution. But, if an assignment satisfies all the clauses, i.e. $\forall\, i \in [1, u],\ e(a_i.x, i).violation = 0$, it is a solution.
4. Agent a_i is able to sense its local environment, which is its row_i. a_i can perceive the violation number for each lattice in row_i. It can find the minimum violation number. Here, we define a function $\Phi(i)$ for finding a position (x-coordination) with the minimum violation number in row_i.

Definition 7. A *minimum-position* is position (x, i) where $i \in [1, u]$, and $(\forall j \in [1, |D_{i1} \times D_{i2} \times \cdots \times D_{ik}|]), e(x, i).violation \leq e(j, i).violation$.

Definition 8. Functions for finding the **first** minimum-position for each agent a_i in row_i:

$$\Phi : [1, u] \to [1, |D_{i1} \times D_{i2} \times \cdots \times D_{ik}|]$$

that is, $\Phi(i) = x$, where (x, i) is a *minimum-position*, and $(\forall\, j \in [1, x)), (j, i)$ is not a *minimum-position*.

5. The objective of each agent is to stay at a zero-position. For each agent in this system, because it can only sense its local environment and cannot sense positions of other agents, it does not know what a "solution" is and what the whole system wants. It simply acts based on its own objective moving toward a zero-position. That is enough for solving a SAT, since if all agents stay at zero-positions, we have found an exact solution to the problem.
6. In order to achieve the goal, each agent has its own moving strategies. Agents want to move toward zero-positions at each time step. But, in most cases, they

cannot, or only some lucky agents can, find zero-positions, simply because some rows do not contain such positions. In such cases, agents will have to perform other moving strategies. Below we introduce some of them. They are easy to implement. As more than one strategy coexist, there should be a probability associated with each strategy. Therefore, before an agent moves, it will first decide which strategy to perform according to the probabilities.

(a) *Least-move*

An agent moves to a minimum-position with a probability of *least-p*. If there exists more than one minimum-position, we let the agent choose the first one on the left of the row. This strategy is instinctive to all agents. The least-move strategy can be expressed as follows:

$$\Psi(j,i) = \Phi(i)\,.$$

In this function, the result has nothing to do with the current position j, and the number of computational operations to find the position for each i is $|D_{i1} \times D_{i2} \times \cdots \times D_{ik}|$. We use another symbol to represent this movement:

$$\Psi_{-l}(j,i) = \Phi(i)\,.$$

(b) *Better-move*

An agent moves to a position that has a smaller violation number than its current position with a probability of *better-p*. It will randomly select a position and then compare its violation number to decide whether or not it should move to this position. We use function $Random(k)$, which complies uniform distribution, to get a random number between 1 and k. This movement can be defined using function Ψ_{-b}:

$$\Psi_{-b}(j,i) = \begin{cases} j & if\ e(r,i).violation \geq e(j,i).violation \\ r & if\ e(r,i).violation < e(j,i).violation \end{cases}$$

where, $r = Random(|D_{i1} \times D_{i2} \times \cdots \times D_{ik}|)$.

Although it may not be the best choice for the agent, the computational cost required for this strategy is less than that of least-move. Only two operations are involved for deciding this movement, i.e. producing a random number and performing a comparison. This strategy can easily find a position to go to if the agent currently stays at a larger violation position.

(c) *Random-move*

An agent moves randomly with a probability of *random-p*. Random-p will be relatively smaller than the probabilities of selecting least-move and better-move strategies. It is somewhat like a random-walk in local search. For the same reason as in local search, random-move is necessary because without randomized movements the system will get stuck in local-optima, that is, all the agents are at minimum-positions, but not all of them at zero-positions. In the state of local-optima, no agent will move to a new position if using the strategies of least-move and better-move only. Thus, the agents will lose

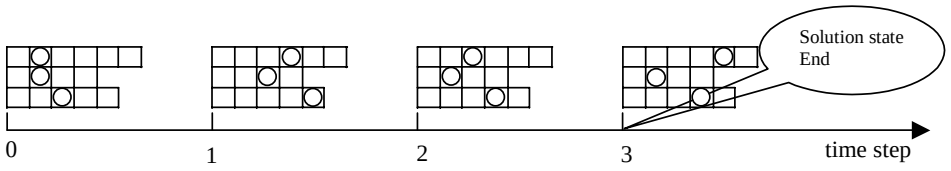

Fig. 3. Asynchronous agent-environment interaction at different time steps.

their chance for finding a solution if without any techniques to avoid getting stuck in local-optima.

Random-move can be defined using function Ψ_{-r}:

$$\Psi_{-r}(j,i) = Random(|D_{i1} \times D_{i2} \times \cdots \times D_{ik}|)\,.$$

The above three moving strategies are elementary. They are simple and easy to implement. We can combine these moving strategies to get new complex strategies. We will discuss this issue in the later part.

2.4. *System schedule*

The multiagent system presented in this paper is asynchronous and discrete in nature, with respect to its space, time and state space. The system will use a discrete timer to synchronize its cycles, as illustrated in Fig. 3.

time step = 0:

The system is initialized. We place u agents onto the environment, a_1 in row_1, a_2 in $row_2, \ldots, a_u$ in row_u. The simplest way to place the agents is to randomly select positions. That is, for a_i, we get a position of $(Random(|D_{i1} \times D_{i2} \times \cdots \times D_{ik}|), i)$. Of course, it can as well be a solution state.

time step ← time step + 1:

After the initialization, the system will start to run. At each time step, which means after one unit increment of the system timer, all agents will have a chance to decide their movements, that is, whether to move or not and where to move, and then move *asynchronously*. It should be pointed out that in this paper, we are concerned only with a simulation of the multiagent system, which dispatches the agents one by one. The order of dispatching does not influence the performance of the algorithm. It may be based on a random or predefined sequence.

After movement of an agent from (j_1, i) to (j_2, i), the violation number of the environment will be updated according to the following two update-rules:

- *Update-rule 1*: Remove from (j_1, i):
  For $(\forall i' \in [1, u])(\forall j' \in [1, |D_{i1} \times D_{i2} \times \cdots \times D_{ik}|])$:
  If: there are v clauses some of whose variables are included in row_i and $row_{i'}$, and whose values are changed from false to true;
  Then: $e(j', i').violation \leftarrow e(j', i').violation - v$;

- *Update-rule 2:* Add to (j_2, i):
  For $(\forall\, i' \in [1, u])(\forall\, j' \in [1, |D_{i1} \times D_{i2} \times \cdots \times D_{ik}|])$:
  If: there are v clauses some of whose variables are included in row_i and $row_{i'}$, and whose values are changed from true to false;
  Then: $e(j', i').violation \leftarrow e(j', i').violation + v$;

End:

After each movement of an agent, the system will check whether all agents are at zero-positions. If yes, a solution state is found. The system will stop and output the answer. Otherwise, the system will continue to dispatch the next agent to move in the dispatching order.

We can also set a threshold *t-max* for the timer such that when the time step reaches *t-max*, the system will stop and output an assignment of the current state, no matter whether it is a solution or not. Another way to terminate the operation is when q agents are staying at zero-positions. Of course, these settings are just for obtaining an approximate solution.

The following shows the complete algorithm for the ERA system.

Input: n variables, domains of variables, and clauses.
Output: an (approximate) solution.
Algorithm:
Section-1. Initialization:

1. $time\ step = 0$;
2. **For** all position $(i, j) \in environment$ **do**
3. $e(i, j).value =$ the jth value of Cartesian product $D_{i1} \times D_{i2} \times \cdots \times D_{ik}$;
4. $e(i, j).violation = 0$;
5. **End for**
6. **For all** $a_i \in A$ **do**
7. $a_i.random\text{-}p = p_1$;
8. $a_i.least\text{-}p = p_2$;
9. $a_i.better\text{-}p = p_3$;
10. $a_i.x = Random(|D_{i1} \times D_{i2} \times \cdots \times D_{ik}|)$;
11. **End for**
12. **For** all position $(j', i') \in environment$ **do**
13. **If** there are v unsatisfied clauses with respect to some variables in row i' **Then**
14. $e(j', i').violation \leftarrow v$;
15. **End for**

Section-2. Running:

16. **While** (*true*) **do**
17. **For** all $a_i \in A$ **do**
18. $p = Random(a_i.random\text{-}p + a_i.least\text{-}p + a_i.better\text{-}p)$;

19. **If** $p \leq a_i.random\text{-}p$ **then**
20. New position $(j'', i) = (\Psi_r(a_i.x, i), i)$;
21. **Else if** $p \leq a_i.random\text{-}p + a_i.least\text{-}p$ **then**
22. New position $(j'', i) = (\Psi_l(a_i.x, i), i)$;
23. **Else**
24. New position $(j'', i) = (\Psi_b(a_i.x, i), i)$;
25. **If** current-position $(a_i.x, i) = (j'', i)$ **then**
26. Stay;
27. **Else**
28. $a_i.x = j''$;
29. **End for**
30. Use two *update-rules* to update the violation values of an environment;
31. **If** current-state is an acceptable solution **then** GoTo 34;
32. time step ++;
33. **End while**

Section-3. Solution:

34. **For** all $a_i \in A$ **do**
35. **For** l **from** 1 **to** k
36. $X_{il} = e(a_i.x, i).value.l$;
37. **End For**
38. **End For**

3. An Example

In this section, we will walk through an example to show how to apply the ERA method to solve a SAT problem.

Example 2. A SAT,

$$\mathbf{X} = \{X_1, X_2, X_3, X_4, X_5\}, \quad n = 5;$$

$$\mathbf{D} = \{D_1, D_2, D_3, D_4, D_5\}, \quad D_1 = \{True, False\}, \quad D_2 = \{True, False\},$$
$$D_3 = \{True, False\}, \quad D_4 = \{True, False\}, \quad D_5 = \{True, False\};$$

$$\mathbf{C} = \{T(X_3 \vee X_4 \vee \neg X_5) = True, \quad T(X_2 \vee \neg X_3 \vee \neg X_5) = True,$$
$$T(\neg X_1 \vee \neg X_2 \vee X_3) = True, \quad T(\neg X_1 \vee \neg X_2 \vee X_4) = True,$$
$$T(\neg X_3 \vee X_4 \vee X_5) = True, \quad T(X_1 \vee \neg X_2 \vee \neg X_3) = True,$$
$$T(\neg X_2 \vee X_4 \vee X_5) = True, \quad T(\neg X_1 \vee \neg X_3 \vee \neg X_5) = True,$$
$$T(X_2 \vee \neg X_3 \vee X_4) = True, \quad T(\neg X_1 \vee X_4 \vee \neg X_5) = True,$$
$$T(X_2 \vee X_3 \vee X_5) = True, \quad T(X_1 \vee X_2 \vee \neg X_4) = True,$$
$$T(\neg X_1 \vee X_2 \vee \neg X_5) = True, \quad T(\neg X_1 \vee X_3 \vee X_4) = True$$
$$T(X_1 \vee \neg X_4 \vee \neg X_5) = True\}.$$

This SAT contains five variables and 15 clauses. First, we divide five variables into three groups: $\{X_1, X_2\}, \{X_3, X_4\}$, and $\{X_5\}$, and use three agents, a_1, a_2 and a_3, to represent them. Second, we model the domains as the environment of the agents. The domain values will be recorded as $e(j, i).value$ [see Fig. 4(a)] and the violation numbers for all positions will be initialized to 0. After that, agents will be randomly placed onto different rows [see Fig. 4(b)]. Accordingly, with respect to the positions of the agents, the violation numbers in the environment are updated [see Fig. 4(c)]. Thereafter, the cycles of distributed agent movements start. In ERA, the agents are dispatched in a random or predefined order. Here, we assume the order is: $a_1 \rightarrow a_2 \rightarrow a_3$.

At the first time step, with respect to the above defined dispatching sequence, the system first dispatches agent a_1 to move. Agent a_1 moves by applying a least-move strategy, $\Psi_{\_l}(1, 1) = 3$. As a result, it moves to position (3, 1). Agent a_2 takes a least-move too, from (2, 2) to (1, 2). And agent a_3 randomly moves from (1, 3)

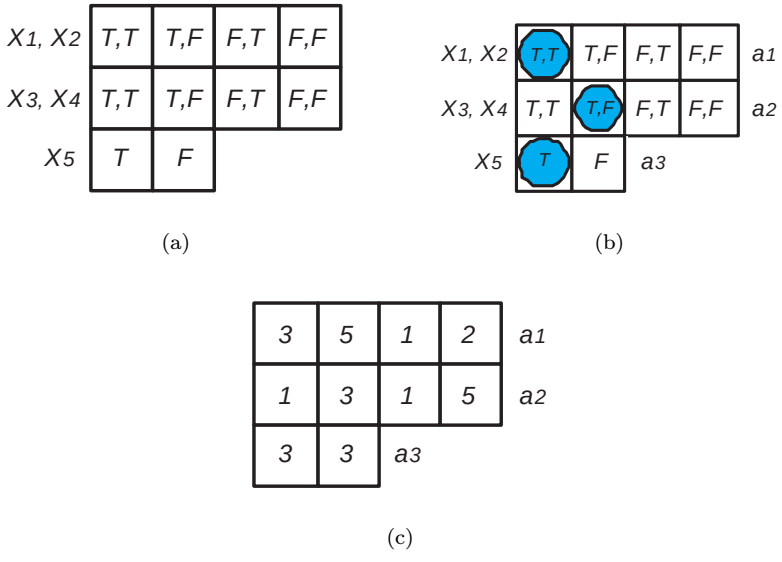

Fig. 4. (a) Domain values, (b) an initialization state, (c) violation numbers.

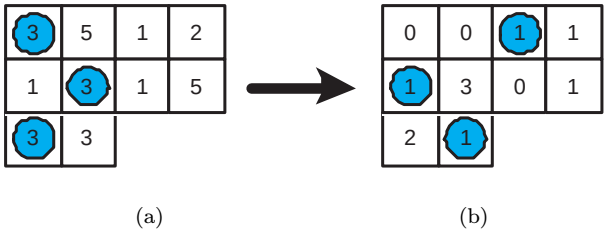

Fig. 5. The first time step. a_1 least-moves. a_2 least-moves. a_3 random-moves.

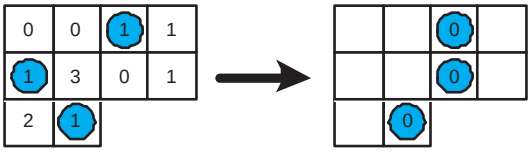

Fig. 6. The second time step. a_1 better-moves, but fails. a_2 least-moves. a_3 least-moves, but fails.

to (2, 3) [see Fig. 5(b)]. Then, the system checks whether this state is a solution state, and finds all three agents are not at zero-positions. So, it is not a solution state. Thus, the system begins the second time step.

At the second time step, agent a_1 select a better-move, $\Psi_{-l}(3,1) = 4$. But, due to $(4,1).violation = (3,1).violation = 1$ [see Fig. 5(b) or 6(a)], a_1 fails to move. Hence, it stays at (3, 1). Agent a_2 lease-moves from (1, 2) to (3, 2). Agent a_3 prepares a least-move. But, it fails to move too. So, it stays [see Fig. 6(b)]. Then, the system finds that all agents are at zero-positions, meaning that it is a solution state.

$$
\begin{aligned}
&a_1 \text{ stays at position } (3,1) \Rightarrow \{X_1 = False, X_2 = True\}\,;\\
&a_2 \text{ stays at position } (3,2) \Rightarrow \{X_3 = False, X_4 = True\};\\
&a_3 \text{ stays at position } (2,3) \Rightarrow \{X_5 = False\}\,.
\end{aligned}
$$

So, the final solution is: $X_1 = False$, $X_2 = True$, $X_3 = False$, $X_4 = True$, and $X_5 = False$.

4. Experimental Results and Discussions

The preceding sections have provided a formal description of the ERA method for solving SATs. In this section, we will present several ERA experimental results on a set of benchmark SAT problems. We will also discuss some important issues related to ERA for solving SATs. We can see that the results on SAT are comparable to those of well-known algorithms for SAT.

In the experiments, we initialize all the agents with the same parameters *random-p, least-p, better-p*, $\forall\, i \in [1,u], a_i.$*random-p* = *random-p*, $a_i.$*least-p* = *least-p*, $a_i.$*better-p* = *better-p*.

4.1. *Benchmark SAT problems*

In order to compare with other algorithms for SAT, we will show the ERA experimental results on some benchmark problems from Ref. 26: *uf100-430* and *flat50-115*. In Ref. 11, Hoos and Stützle made an empirical evaluation of different local search algorithms for SAT, and two of their test-sets are *uf100-430* and *flat50-115*. *uf100-430* is a subset of Uniform Random-3-SAT problems[26] where each

clause of instances has exactly three literals which are randomly selected from $2n$ literals according to uniform distribution. (Here, we assume there are n variables to construct instances.) *flat50-115* is a subset of *Flat Graph Coloring* problems[26] where clauses may have different number of literals.

In the following experiments, we tune the optimal parameter settings (see each experiment) at first, and then collect statistical results on *uf100-430* and *flat50-115*.

Experiment 1. In this experiment, the test-set is a subset of *Uniform Random-3-SAT* problems: *uf100-430*. This test-set includes 1000 instances, and each instance contains 100 variables and 430 clauses. We run each instance 100 times. The size of variable group is 4, *least-p* : *random-p* = 40, *type* = $F2BLR$.

Experiment 2. Test-set is a subset of *Flat Graph Coloring* problems: *flat50-115*. This test-set includes 1000 instances, and each instance contains 150 variables and 545 clauses. We give each instance 100 runs. The size of variable group is 3, *least-p*: *random-p* = 80, *type* = $F2BLR$.

Observation 1. In the last row of Table 1, we have listed our experimental results, i.e. the mean number of movements of agents to get solutions in 100 runs and in 1000 different instances. Also in Table 1, we have extracted and listed some experimental data from Ref 11, and compared them with our results:

1. When contrasting with other popular algorithms in the SAT community, our ERA method gives comparable results with both test-sets.
2. We note that, for some algorithms in Table 1, their performances are not stable for different test-sets. Such as R-Novelty, its performance is the best in test-set *uf100-430*, but in test-set *flat50-115*, the performance is poor. In this respect, our ERA method yields consistent results. That means our method is stable between two different problem types.

4.1.1. *Measurement considerations*

ERA is a distributed approach where agents move asynchronously. On the contrary, other algorithms listed in Table 1 are sequential ones. Therefore, it is, in essence,

Table 1. Mean-movement(flip)-number of different algorithms on two benchmark SAT problem test-sets.

Algorithms	**Uf100-430**	**Flat50-115**
GWSAT	6532	7023
GSAT/TABU	4783	1040
HWSAT	3039	2641
WalkSAT	3672	3913
WalkSAT/TABU	2485	61393
Novelty	28257	20065
R-Novelty	1245	7109
ERA	**3105**	**3866**

hard to compare these two kinds of entirely different approaches. In order to give a qualitative view about the performance of the ERA approach in solving SATs, in this paper we gave a comparison between a sequential simulation of the actual ERA algorithm and those in Table 1. In particular, we compared the movements in the sequential version of the ERA algorithm with the flips in other algorithms.

In other algorithms, the number of *flips* is an important and commonly used index to evaluate the performance. "Flip" means changing the value of a variable in a complete assignment from $True$ to *False*, or from *False* to *True*. In the sequential version of the ERA model, a movement of an agent will cause, in an extreme case, $\lfloor n/u \rfloor$ variables to change their values. If we just consider the value changes that occur on the variables, the comparison in Table 1 is unfair to the other algorithms. But, in essence, what those algorithms should count is how many time steps they take to get a solution rather than values changed. Therefore, the correct way to compare the sequential ERA and other algorithms is to compare their time steps. In other algorithms, the number of time steps corresponds to the number of flips. In our ERA case, we recorded the movements of all agents where one movement may possibly cause multiple flips simultaneously.

4.1.2. *Performance*

In what follows, we will examine the performance of the ERA method in finding an "approximate" solution to SAT. We know that, for a SAT problem, there is only three possible answers: "satisfiable", "unsatisfiable" and "unknown", no matter we use which algorithm to solve it, i.e. there is no "approximate" solution. But, here, we employ the term "approximate" to mean how many clauses will be satisfied under a complete assignment to all variables. To some extent, it is like the MAX-SAT problem. Through the following experiments, we will see that ERA is efficient in finding an "approximate" solution in first three steps.

Experiment 3. Test-sets are five subsets of *Uniform Random-3-SAT* problems: $\{uf50, uf100, uf150, uf200, uf250\}$. The five test-sets include 1000, 1000, 100, 100, and 100 instances, respectively. The number of variables is from 50 to 250, and the

Table 2. Satisfied clauses number and its ratio to the number of clauses in the first three steps on benchmark test-sets of SAT: *Uniform Random-3-SAT*. Note: S_C_N is Satisfied Clauses Number. Ratio is the ratio between S_C_N and Clauses Number. The same is true for Tables 3 and 4.

Step	**1**		**2**		**3**	
Test-set	**S_C_N**	**Ration**	**S_C_N**	**Ration**	**S_C_N**	**Ration**
Uf50	208	*0.954*	212	*0.972*	213	*0.977*
Uf100	410	*0.953*	418	*0.972*	420	*0.977*
Uf150	615	*0.953*	628	*0.974*	630	*0.977*
Uf200	820	*0.953*	836	*0.972*	840	*0.976*
Uf250	1016	*0.953*	1036	*0.972*	1040	*0.977*

Table 3. Satisfied clauses number and its ratio to the number of clauses in the first three steps on benchmark test-sets of SAT: *Flat Graph Coloring*.

Step	**1**		**2**		**3**	
Test-set	**S_C_N**	**Ration**	**S_C_N**	**Ration**	**S-C-N**	**Ration**
Flat50	208	*0.941*	534	*0.980*	536	*0.983*
Flat100	410	*0.941*	1093	*0.980*	1097	*0.982*
Flat150	615	*0.941*	1644	*0.979*	1650	*0.982*
Flat200	820	*0.940*	2189	*0.979*	2197	*0.982*

Table 4. Satisfied clauses number and its ratio to the number of clauses in the first three steps on benchmark test-sets of SAT: *Uniform Random-3-SAT*.

Step	**1**		**2**		**3**	
Test-set	**S_C_N**	**Ration**	**S_C_N**	**Ration**	**S_C_N**	**Ration**
Uuf50	208	*0.954*	211	*0.968*	212	*0.972*
Uuf100	410	*0.951*	417	*0.970*	419	*0.974*
Uuf150	615	*0.952*	626	*0.970*	629	*0.975*
Uuf200	820	*0.952*	835	*0.971*	838	*0.974*
Uuf250	1016	*0.952*	1034	*0.971*	1038	*0.975*

corresponding number of clauses from 218 to 1065. We give each instance 10 runs, and at the same time, we calculate the mean value for the number of satisfied clauses of each time step at the first three time steps. (see Table 2.)

Experiment 4. Test-sets are four subsets of *Flat Graph Coloring* problems: $\{flat50, flat100, flat200, flat250\}$. The first test-set includes 1000 instances. The last three test-sets include 100 instances. The number of variables ranges from 150 to 600, and the number of clauses from 545 to 2237. We also give each instance 10 runs, and calculate the mean value for the number of satisfied clauses at each time step of the first three time steps. (see Table 3.)

In Experiments 3 and 4, all instances are satisfiable. Let us see a special situation, that is, all instances are unsatisfiable.

Experiment 5. In this experiment, like Experiment 3, test-sets are also from *Uniform Random-3-SAT* problems: $\{uuf50, uuf100, uuf150, uuf200, uuf250\}$. The other parameters are the same as Experiment 3 except that all instances in this experiment are unsatisfiable. The results are shown in Table 4.

Observation 2. From Tables 2–4, we note that, using the ERA method,

1. We can get an approximate solution with about 94–95% satisfied clauses after the first time step, no matter the instances are satisfiable or unsatisfiable.
2. After the second time step, the numbers of satisfied clauses will have quick improvements. But, the lengths of improvements are different between the two different test-set types. The improvements in the type of *Flat Graph Coloring* are larger.

3. After the third time step, the ratio of satisfied clause will go up to about 97–98% no matter the instances are satisfiable or unsatisfiable.
4. Although there are few differences among test-sets, we can say that ERA is stable and robust.

4.2. *Discussions*

In this subsection, we will discuss several important issues related to the ERA approach.

4.2.1. *The necessity of better-move strategy*

Generally speaking, in other search algorithms, there are only two kinds of flips. One is the greediest flip, i.e. flipping will cause the steepest hill-climbing, and another is random flip. But in our ERA method, there are three moving strategies: least-move, random-move and better-move. That is to say, ERA has a new moving strategy better-move. Is it necessary?

From the aforesaid sections, we know that better-move and least-move are similar: move to a position based on the violation number. But, they are different. At each time step, it would be much easier for an agent using least-move to find a better position to move to than for the one using better-move. This is because least-move checks all the positions in its row, whereas better-move just randomly selects and checks one position. If all agents use only random-move and better-move strategies, the efficiency of the system will be low, since many agents cannot find a better position to move to at each time step. But, on the other hand, the time complexity of better-move is much less than that of least-move. So, we think better-move must be necessary. And we guess if we can find an equilibrium point between better-move and least-move, it will greatly improve the ERA method.

In order to balance the shortcomings and advantages of these two strategies, we have found a way to combine them. At first, an agent uses a better-move to compute its new position. If it succeeds, the agent will move to the new position. If it fails, it will continue to perform several other better-moves until it finds a successful better-move. If it fails all better-moves, it will perform a least-move without any other choice. But, in this case, there is a very straightforward question: How many better-moves before a least-move will be desirable?

It is obvious that, during the initialization step, many agents are not in a "good" position, that is, they stay at the positions with large violation numbers. In this case, the probability of using better-move to successfully find a position to move to is high. But, as the process goes on, more and more agents will be at good positions. At this time, there will be little chance for an agent to move by a better-move strategy. Under this situation, intuitively, better-move seems to waste the time of agents without further improvement. Is this guess right?

In order to answer the above questions, we have further conducted two experiments. The experimental results show that: to the first question, the ERA algorithm

will yield the best performance if there are two better-moves before a least-move. More better-moves will increase the runtime complexity. And fewer better-moves cannot create enough chance for agents to find a better position. To the second question, we find that $F2BLR$ moving strategy could obtain best performance. Here, $F2BLR$ means that at the first step, the ERA algorithm will probabilistically select a least-move or random-move strategy to move. If it chooses least-move strategy to move, it will has two chances to select a better-move before performing a least-move. But, it is the case only for the first step.

4.2.2. *How to set the probabilities*?

Among three elementary moving strategies in the ERA method, we know that least-move and better-move play important roles in performance of ERA to find a solution. However, random-move is still necessary, because if there is no random-move, i.e. $random-p = 0$, the system may get stuck in a local optimum and cannot find a solution. Therefore, all three moving strategies are necessary in ERA.

Now, there is a further question: How to set the probabilities for the three strategies in order to have the best performance of ERA? From Sec. 4.2.1, we can see that better-move occurs as a prologue of best-move in the best moving strategy of F2BLR. In this case, the combination of better-move and least-move will have the same probability with a single least-move. Therefore, the probabilities we mostly care about are *least-p* and *random-p*. It is the ratio of *least-p* to *random-p* that plays an important role in the system. Our experimental results show that when the ratio of *least-p* to *random-p* is about $1.5u$ for SAT problems, the performance of ERA algorithm will be the best.

4.2.3. *About variable grouping*

In the ERA model, we divide variables into groups. In the experiments given in Sec. 4.1, we equally divided variable into groups, i.e. four or five variables are grouped together in a SAT problem. Through experiments, we note that how to partition variables is a very important factor influencing the performance of the ERA algorithm. In fact, in the case of equally dividing variables into groups, the above configurations for the size of a variable group are the best ones we have found. But what is more interesting is that equally dividing variables into groups may not be the best way to partition variables. We have observed from some other experiments that which variables should be placed into a group is a more important aspect than the size of a variable group. But, to date, how to partition the variables in an optimal way still remains unsolved. This is one of the questions we will study in the future.

4.2.4. *What are the real values of ERA*?

In the above subsections, we have discussed several important issues about the ERA method. But what are the real values of ERA? In Sec. 4.1, we have shown that the

ERA method can generate comparable result in solving SATs. Also in this section, we have seen that the performance of ERA in finding an approximate solution is very efficient. After only the first three time steps, there will be about 97–98% clauses that are satisfied in SAT problems. This property is quite important if a solution response is required with a hard deadline. Here, we also want to emphasize some other features and advantages of ERA:

1. *The ERA method is quite open and flexible*: *we may easily add new moving strategies to it or combine present moving strategies into a complex one. In addition, we may give each agent different parameter settings. Further, we may modify the way of agent-environment interaction for different problems.*
2. *The ERA system is a self-organizing system. All agents in it are autonomous in governing their actions. The process of solving SATs by ERA is entirely determined by the locality and parallelism of individual agents. The ERA system has no centralized control and global evaluation policy.*
3. *All agents in the ERA system move asynchronously. The movement of an agent may affect the whole environment. The change of environment will in turn affect moves of other agents. In other words, the interaction among agents is indirectly carried out through the medium of their environment. In this sense, we may regard that the agents can cooperate among them in finding a solution without the cost of the explicit communication. On the contrary, in Multi-DB, all agents must communicate with explicit message sending and receiving policies.*

Because of the above features and advantages, we believe that both ERA and Multi-DB are well suited to the situations that are of a distributed nature and can be translated into a SAT form. But, if the situations do not allow explicit communication or the communication is too costly, our ERA approach will become the only choice.

5. Conclusion and Future Work

In this paper, we have extended and modified the ERA approach in Ref. 13 to solving Satisfiability Problems (SATs). The key ideas behind this approach rest on three notions: Environment, Reactive rules and Agents (ERA). In ERA, each agent can only sense its local environment and apply some behavioral rules for governing its movements. The environment records and updates the local values that are computed and affected according to the movements of individual agents.

In solving a SAT with the ERA method, we first divide variables into several groups, and then represent each variable group with an agent whose positions correspond to elements of Cartesian product of variable domains. The environment for the whole multiagent system contains all the possible domain values for the problem, and at the same time, it also records the violation numbers for all the positions.

An agent can move within its row, which represents its domain. So far, we have introduced three elementary moving strategies: *random-move*, *least-move* and *better-move*. Using these moving strategies, we can constitute other complex moving strategies, such as *F2BLR*. The movement of an agent will affect the violation numbers of lattices in the environment. It may add or reduce the violation number of a position. After being randomly initialized, the ERA system will keep on dispatching agents, according to a random or predefined order, to choose their movements until an exact solution or approximate solution is found.

While describing the ERA method, we also presented some experimental results on benchmark SAT test-sets: *Uniform Random-3-SAT* and *Flat Graph Coloring*, which can be found from the SATLIB[26] website. Our experimental results have shown that the ERA method can be applied to SAT problems, and we can obtain comparable and more stable results than other popular algorithms. Besides this, we have noticed that our new ERA approach is well suited to the SAT related situations where explicit communication is not allowed or the communication is too costly.

From our experiments, we have identified that with respect to ERA used in solving SAT, there are some aspects we should study further: first, we have mentioned in the previous section that how to partition variables significantly influences the performance of the ERA algorithm. Second, there may be other moving strategies for agents. We have seen that, although ERA can find a very good "approximate" solution to a SAT problem in the first three steps, it spent many steps in finding an exact solution. That is because present moving strategies cannot most efficiently lead agents to escape from local optima.

Acknowledgment

This work has been supported in part by HKBU FRG research grants.

References

1. R. Barták, On-Line Guide To Constraint Programming, http://kti.mff.cuni.cz/ bartak/constraints/stochastic.html.
2. S. Camazine, J. L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraulaz and E. Bonabeau, *Self-Organization in Biological Systems*, Princeton University Press, 2001.
3. G. Folino, C. Pizzuti and G. Spezzano, "Parallel hybrid method for SAT that couples genetic algorithms and local search," *IEEE Trans. Evolut. Comput.* **5**, 4 (2001) 323–334.
4. I. P. Gent and T. Walsh, "Towards an understanding of hill-climbing procedures for SAT," *Proc. Eleventh National Conf. Artificial Intelligence* (*AAAI-93*), AAAI Press/MIT Press, 1993, pp. 28–33.
5. I. P. Gent and T. Walsh, "Unsatisfied variables in local search," *Hybrid Problems, Hybrid Solutions*, ed. J. Hallam, IOS Press, 1995, pp. 73–85.
6. J. Gu, "Efficient local search for very large-scale satisfiability problem," *SIGART Bull.* **3** (1992) 8–12.

7. H. Gutowitz, *Cellular Automata*: *Theory and Experiment*, MIT Press, Cambridge, MA, 1991.
8. F. Heylighen, "Self-organization, emergence and the architecture of complexity," *First European Conf. System Science*, 1989, pp. 23–32.
9. K. Hirayama and M. Yokoo, "Local search for distributed SAT with complex local problems," *Proc. First Int. Joint Conf. Autonomous Agent and Multiagent System*, Bologna, Italy, July 2002, pp. 1199–1206.
10. H. H. Hoos and T. Stützle, "Systematic vs. local search for SAT," *Proc. KI-99*, LNAI Vol. 1701, Springer, 1999, pp. 289–293.
11. H. H. Hoos and T. Stützle, "Local search algorithms for SAT: an empirical evaluation," *J. Autom. Reason.* **24** (2000) 421–481.
12. J. Liu, *Autonomous Agent and Multi-Agent Systems*: *Explorations in Learning*, *Self-Organization and Adaptive Computation*, World Scientific Publishing, 2001.
13. J. Liu, J. Han and Y. Y. Tang, "Multi-agent oriented constraint satisfaction," *Artif. Intell.* **136** (2002) 101–144.
14. J. Liu, Y. Y. Tang and Y. Cao, "An evolutionary autonomous agents approach to image feature extraction," *IEEE Trans. Evolut. Comput.* **1** (1997) 141–158.
15. J. Liu and K. C. Tsui, "Autonomy oriented computation," *IEEE Trans. Evolut. Comput.*, to appear.
16. B. Mazure, L. Sais and É. Grégoire, "Tabu search for SAT," *Proc. Fourteenth Natl. Conf. Artificial Intelligence* (*AAAI'97*), 1997, pp. 281–285.
17. D. McAllester, B. Selman and H. Kautz, "Evidence for invariants in local search," *Proc. Fourteenth Natl. Conf. Artificial Intelligence* (*AAAI'97*), 1997, pp. 321–326.
18. T. Mori and K. Nishikimi, *Self-Organization in the Spatial Economy*: *Size, Location and Specialization of Cities*, http://keiken8.kier.kyoto-u.ac.jp/ mori/research.html
19. B. Selman, H. Kautz and B. Cohen, "Noise strategies for improving local search," *Proc. Twelfth Natl. Conf. Artificial Intelligence* (*AAAI'94*), 1994, pp. 337–343.
20. B. Selman, H. Levesque and D. Mitchell, "A new method of solving local search," *Proc. Tenth Natl. Conf. Artificial Intelligence* (*AAAI'92*), 1992, pp. 440–446.
21. O. Steinmann, A. Strohmaier and T. Stützle, "Tabu search vs. random walk," *Adv. Artif. Intell.* (*KI'97*) **1303** (1997) 337–348.
22. C. Ünsal, "Self-organization in large populations of mobile robots," M.S. Dissertation, Virginia Polytechnic Institute and State University, 1993, http://armyant.ee.vt.edu/unsalWWW/cemsthesis.html.
23. Swarm Team, "An overview of the Swarm simulation system," http://www.santafe.edu/projects/swarm/swarm-blurb/swarm-blurb.html.
24. http://www.calresco.org/sos/sosfaq.htm.
25. http://algodones.unm.edu/ ehdecker/SOS/SOS.html.
26. http://www.intellektik.informatik.tu-darmstadt.de/SATLIB/.

Jiming Liu is an Associate Professor in the Department of Computer Science at Hong Kong Baptist University. He is the Editor-in-Chief of *Web Intelligence and Agent Systems: An International Journal* (IOS Press) and *Annual Review of Intelligent Informatics* (World Scientific). Among his works, he published several research monograph books, including Autonomous Agents and Multi-Agent Systems: Explorations in Learning, Self-Organization, and Adaptive Computation (World Scientific, 2001), Multi-Agent Robotic Systems (CRC Press, 2001), both of which are being translated into Chinese language and published by Tsinghua University Press, and Web Intelligence (Springer, 2002). He received his Ph.D. in electrical engineering from McGill University, Canada. He can be contacted at http://robotics.comp.hkbu.edu.hk/∼jiming

Xiaolong Jin received the B.Sc. degree from the Department of Applied Mathematics, Beijing University of Aeronautics and Astronautics in July, 1998 and the M.Eg. from Academy of Mathematics and System Sciences, Chinese Academy of Sciences in July, 2001. He is currently pursuing the Ph.D. in computer science at Hong Kong Baptist University.

His current research interests include autonomous agent and multi-agent systems, autonomy oriented computation, distributed problem solving, satisfiability problems, constraint satisfaction problems, etc.

Biography and photograph of Jing Han are not available.