Constraint Satisfaction by Parallel Optimization of Local Evaluation Functions with Annealing

Yasusi Kanada*

Tsukuba Research Center, Real World Computing Partnership
Takezono 1-6-1, Tsukuba, Ibaraki 305, Japan

Abstract

A method for solving large-scale constraint satisfaction problems is proposed in the present paper. This method is stochastic (or randomized) and uses local information only, i.e., no global plan is expressed in the program and the computation refer to no global information. This method uses CCM (Chemical Casting Model) as a basis, which is a model for emergent computation proposed by the author. The original CCM-based method minimizes the number of constraint violations not directly but throught optimization of local functions, which are called LODs (local order degrees). This method sometimes falls into a "local maximum." This difficulty is solved by a type of annealing, which we call the frustration accumulation method (FAM). FAM also works only with local information. No global functions is used in FAM, No global parameters such as temperature are used, and global control is thus unnecessary. Experiments show that the performance of this method is not very sensitive to parameter values. This means that parameter tuning is easy. In several problems, the performance is comparable to conventional simulated annealing or GSAT, which are based on global evaluation functions. Because of the nonexistence of global information reference, CCM with FAM can be parallelized very easily. Thus, the performance is improved and is almost linear in certain cases.

1. Introduction

Constraint satisfaction and optimization problems in the real world, such as production scheduling or stock management, are not only large-scale and complex but also open to the human society and/or the nature. The problems themselves are continually changing and only incomplete information is available for solving them, because the systems concerning the problem, the environment, and even the requirements are changing. The changes may sometimes be unexpected, because the behavior of humans and other natural systems are sometimes unpredictable.

Conventional methods for problem solving, such as those used in operations research or various search methods in artificial intelligence, such as the branch-and-bound method, alpha-beta search, and so on, do not work well in such situations. We believe that this is because global and complete planning is necessary and global information is used in these methods. Global planning and global information based computation are considered to be easily voided by small changes in the information, requirements or environment. Global information referencing also

---

* The author's current address is Hitachi Central Research Laboratory, Kokubunji, Tokyo 185, Japan. The email address is kanada@crl.hitachi.co.jp.

makes parallel processing difficult, because of the global reference creates a global data dependence.

Toward solving these difficulties, a method of solving constraint satisfaction problems using a type of optimization of local evaluation functions with a type of annealing is proposed in the present paper. This method is based on CCM (Chemical Casting Model) [Kan 92, Kan 94], which is a stochastic computational model for emergent computation [For 91], or for local information based computation, which has deeply been investigated in Artificial Life [Lan 89-94]. Because computation in CCM does not depend on global information, it continues to work even when information is changed dynamically and new information is added or removed. Thus, it seems to have the potential to process unexpected situations properly using implicit knowledge that emerges from interaction between local computation processes. New information may be added or preexisting information may be dynamically changed by environmental changes while solving problems. The nonexistence of global information references also makes parallel processing much easier.

We experimented with solving combinatorial problems, such as the *N* queens problem [Kan 94a], graph/map coloring problem [Kan 93, Kan 95], or fuzzy coloring problem [Kan 95] using CCM without annealing. However, we could not solve difficult and large-scale constraint satisfaction problems (CSPs) using CCM, because the computational efficiency, which is inevitable for solving large-scale problems, and the escaping from "local maxima" of evaluation functions were not compatible. To solve this difficulty, a new type of annealing technique called the frustration accumulation method (FAM) is introduced in the present paper. Because the annealing method must not violate the nature of CCM, FAM does not depend on global information either. No global parameter, such as temperature, is necessary. No global control, such as changing the value of a global temperature, is necessary, and the system thus works autonomously. The absense of global informtion is different from the usual annealing methods. FAM has made solving large-scale problems using CCM possible.

The computational model, CCM, is briefly explained in Section 2. The basic method of constraint satisfaction using CCM is explained in Section 3, and a method using CCM with FAM is explained in Section 4. A method of parallel processing the constraint satisfaction with FAM is explained in Section 5. The results of performance measurement of both sequential and parallel processings are shown in Section 6. Finally, our conclusion is given in Section 7.

2. Computational Model CCM

CCM [Kan 92, Kan 94a] is explained here. Kanada [Kan 96] describes CCM in more detail.

CCM has been developed for computation using only local information. CCM is based on a forward-chaining production system [New 72], such as systems written in OPS5 [For 81]. The forward-chaining production system is a computational model for bottom-up computation. Production systems are often used for developing expert systems or when modeling the human brain in artificial intelligence or cognitive sciences.

However, CCM differs from conventional forward-chaining production systems in two points. Firstly, evaluation functions, which are called local order degrees (LODs). LODs are similar to negated energy in chemical reaction systems, and are evaluated to decide whether to apply a rule or not. Rules and functions are computed using only local information. Secondly, stochastic control or randomized ordering of rule applications, is applied.[1] Because of these features, we believe that CCM is much more similar and analogical to chemical reaction systems than conventional production systems in AI. Production systems in AI are completely symbolic. However, CCM is more pattern-oriented because it is based on LODs that are numerical, and has certain similarity to neural systems [Kan 96]. Because chemical reactions occur in parallel, it is very natural to make reactions occur asynchronously in parallel in CCM, and the parallelization is much easier in CCM than in conventional production systems.

The structural components in CCM are shown in **Figure 1**. The set of data to which the rules apply is called the *working memory*. A unit of data in the working memory is called an *atom*. An atom has a type and an internal state, and may be connected to other atoms by *links*. Links are similar to chemical bonds, but the difference is that links may have directions. Any discrete data structures such as lists, trees, graphs or networks can be represented using atoms and links.

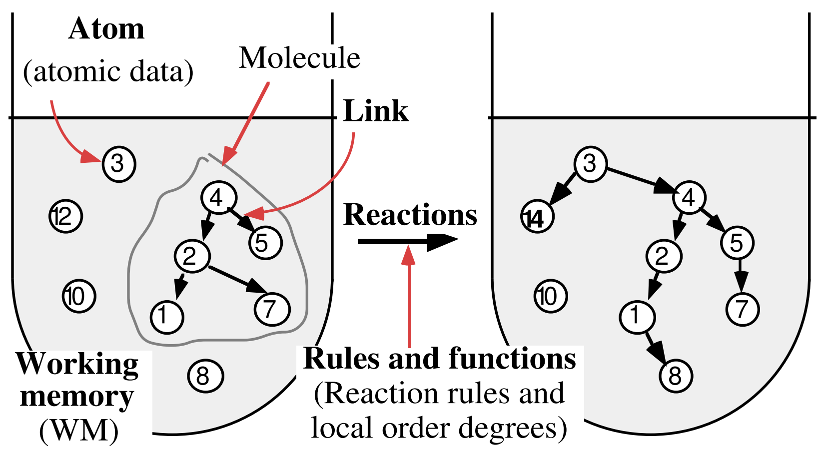


Figure 1. The elements of CCM

A local state of the working memory is changed by a *reaction rule.* "Local" means that the number of atoms referred by a reaction rule is small.[2] The reaction rules are written as forward-chaining production rules, such as rules in expert systems. However, reaction rules are at a lower level, or more primitive, than rules in expert systems. This means that the behavior of the system is not (or should not be) directly programmed using rules but it should emerge from repeated rule applications. Therefore, a reaction rule is more similar to the reaction formula of a single reaction in compex chemical reactions, and thus, this model is called the *chemical* casting model. CCM has both symbolic and pattern-oriented properties, being similar to chemical reaction systems; symbolic as reaction formulae and pattern-oriented as controlled by energy or entropy.

The abstract syntax of reaction rules is as follows:

$$\text{LHS} \rightarrow \text{RHS}.$$

[1] It is possible to regard CCM-based computation as a probabilistic or randomized algorithm. However, this is not discussed here since our aim is to establish a method of solving problems open to the real world but not to solve closed problems.

[2] Because (physical) distance is not (yet) introduced in CCM as opposed to systems such as a chemical reaction system, "local" does not mean that the distance is necessarily small.

The left-hand side (LHS) and the right-hand side (RHS) are sequences of patterns. For example, the following reaction rule, which is a rough sketch, simulates the generation of water from oxygen and hydrogen:

H–H, O=? $\rightarrow$ H–O–H, ? (This approximately means $H_2 + \frac{1}{2}O_2 \rightarrow H_2O$).

There are four patterns on each side of the rule: two H's, an O, and a question mark, which is a variable that means an unknown atom. An O matches an atom of oxygen type, an H matches an atom of hydrogen type, and a variable matches arbitrary data in the working memory. This rule can be visualized as shown in **Figure 2**. Reaction rules must be coded SOOC (Self-Organization-Oriented Computing), which is a computational language and a general problem solver for CCM-based computation.[1]

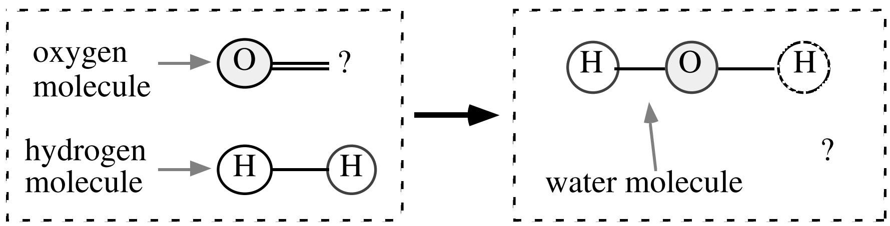


Figure 2. A "water generation" rule

The reaction rule can be activated when there is a set of atoms that matches the LHS patterns. If the reaction rule is activated, the matched atoms vanish and new atoms that match the RHS patterns appear. So an atom may be modified, bonded to another atom, removed, or created by a reaction rule. Just one reaction rule is enough for solving a simpler problem such as the 0–1 integer programming problem or the graph vertex coloring problem as well as "water generation." There will be two or more reaction rules in more complex systems, in which there are two or more ways of changing atoms.

Local order degrees (LODs) are evaluation functions, or local objective functions, whose values are defined for a type of atoms or for a pair of types of atoms, using only local information. LODs are functions with one or two arguments. The arguments are atoms. LODs have a larger value when the local state of the working memory is *better*. An LOD may be regarded as a negated *energy*. For example, it is analogous to the bonding energy in chemical reaction systems. Although LODs can be built into normal production rules, the separation of LODs from rules causes a certain flexibility. The separation causes bidirectionality of rules, i.e., LHS and

---

[1] The same rule as shown in Figure 2 can be expressed in SOOC-94 (the current version of SOOC) as follows:

```
(defrule water-generation        ; rule name
  (var H1 H2 O1)          ; Declaration of variables H1, H2 and O1.
  (exist H H1 :bond H2)   ; A pattern.  H is a type and H1 is a label.
  (exist H H2 :bond H1)   ; A pattern.
  (exist O O1)            ; A pattern.  O is a type.
  -->
  (exist H H1 :bond O1)   ; A pattern.
  (exist H H2 :bond O1)   ; A pattern.
  (exist O O1 :bond1 H1 :bond2 H2))    ; A pattern.
```

RHS may be interchanged. The separation also makes performance improvement by adding catalysts [Kan 94a] or rule composition possible. (See Section 3.2 for the definition of catalyst.)

A reaction takes place when the following two conditions are satisfied. Firstly, there exists an atom that matches each pattern in the LHS. Secondly, the sum of the LODs of all the atoms that are involved in the reaction, i.e., the atoms that appear on either side of the reaction rule, does not decrease as a result of the reaction. For example, a water molecule will be generated by the above rule, if the sum of LODs in the RHS, i.e., the negated energy of a water molecule, is higher than that in the LHS, i.e., the negated energy of a hydrogen molecule and an oxygen atom.

Reactions repeatedly occur while the above two conditions are satisfied by a combination of any rules and atoms. The system enters a stationary state, or stops, when such a combination is exhausted. If the system is to solve a problem, this state must be the solution. However, reactions may occur again if the working memory is modified because of changes in the problem or the environment. A CCM-based system is a dynamical system rather than a problem solving system. Thus, a CCM-based system can solve dynamical problems without additional rules or data.

Typically, there are two or more combinations that satisfy the two conditions at the same time. There are two possible causes that generates multiple combinations. One cause is that there are two or more collections of atoms that satisfy the LHS of a reaction rule, and the other is that there are two or more reaction rules containing atoms that match the patterns in the LHS. In both cases, the order of the reactions (the selection order of such combinations) can be stochastic, and whether they occur in parallel or sequentially is arbitrary. Therefore, although the microscopic behavior of CCM, i.e., a reaction, is deterministic, the macroscopic behavior is nondeterministic or randomized. The same set of rules, LODs and initial state does not necessarily cause the same final state. The CCM-based systems that we have developed use random numbers for selecting rules and atoms. The same set of rules and LODs can be used both in sequential and parallel processing.

For example, a computation process that generates two water molecules is shown in **Figure 3**. Water molecules are generated by two reactions. There are four combinations of atoms for the first reaction; H_1–H_2 and O_1, H_1–H_2 and O_2, H_3–H_4 and O_1, and H_3–H_4 and O_2. Although the combination is selected randomly, the computation process always generates the equivalent result: two water molecules.

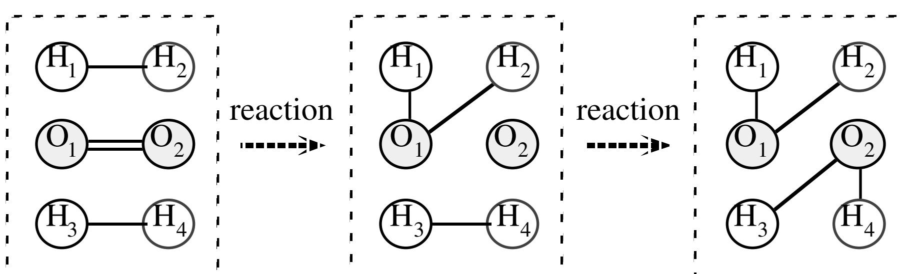


Figure 3. A computation process — generation of water molecules

3. Constraint Satisfaction Using CCM

The principle of problem solving using CCM is explained, and an example system that solves a CSP is given in the present section.

3.1 Principle of problem solving using CCM

Problem solving can be regarded as an activity to find a *better* or *best* state in some sense. The meaning of words, *better* or *best*, may be varied according to the problems to be solved, and "state" may mean the macroscopic (global), microscopic (local), or mesoscopic state. In CCM, LODs define what is a locally better state, and reaction rules determine the paths of transition to neighbor states, or define the neighbor relationship.

A mean value of LODs is called a *mean order degree* (MOD), which can be regarded as a negated mean energy.[1] Although CCM-based systems do not compute MODs, they operate so as to increase MODs or sums of LODs stochastically. The data in the working memory can be divided into small parts, into large parts, or into parts of any other scale. MODs can also be defined in each scale. **Figure 4** illustrates the relationship between MODs of different scales. They may conflict. For example, the macroscopic MOD can decrease when a microscopic MOD increases, and vice versa. A CCM-based system is called a *cooperative system* if no reaction that decreases an MOD can occur, and it is called a *conflicting system* if a reaction that decreases an MOD can occur [Kan 94a]. A conflicting system does not perform hill-climbing.

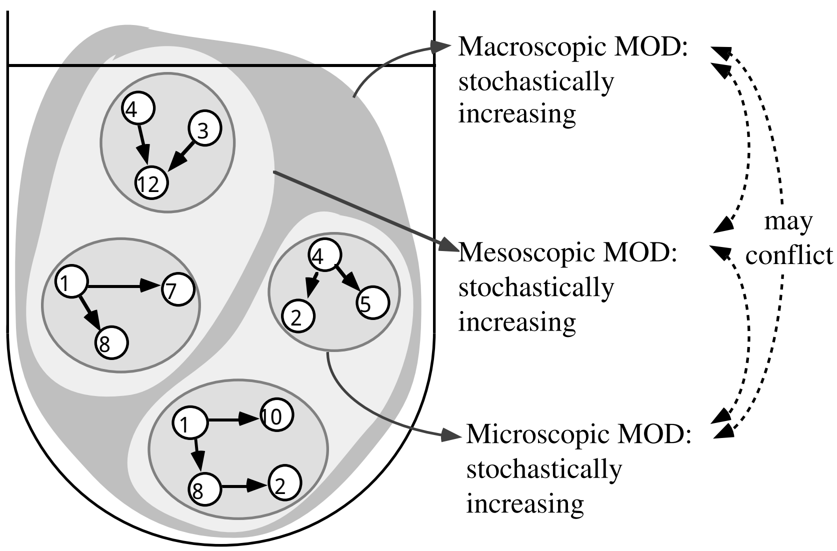


Figure 4. Behavior of CCM-based systems

Although different MODs may behave differently, they increase in average in every scale if the LODs and rules are defined appropriately. Thus, CSPs can be solved if a state in which more constraints are satisfied is defined to have a higher MOD value (or to be *better*), and combinatorial optimization problems can be solved if a more optimized state is defined to have a higher MOD value. A sample time sequence of MOD for the coloring of the USA mainland map (see

[1] Kanada [Kan 94a] used the total of LODs, called global order degree (GOD). GOD has been replaced by MOD, because GOD has the same drawback as global objective functions used in conventional evolutionary computation and other methods. GOD cannot be properly defined in open or partially-informed situations.

Figure 5), which is a typical example, is shown in **Figure 6**. The MOD sometimes decreases, because this is a conflicting system. However, experiments show that this system never fails to find a solution.

A certain solution can probably be found, even if what is a global optimum is not known, because the system can work only if what is locally better is known. However, the system may be unable to satisfy all the constraints, may be unable to find a global optimum solution, or can stop at such a "goal" state by this method. This is because the system may be trapped by a local maximum and there is no assurance that the system stops in finite time in CCM.

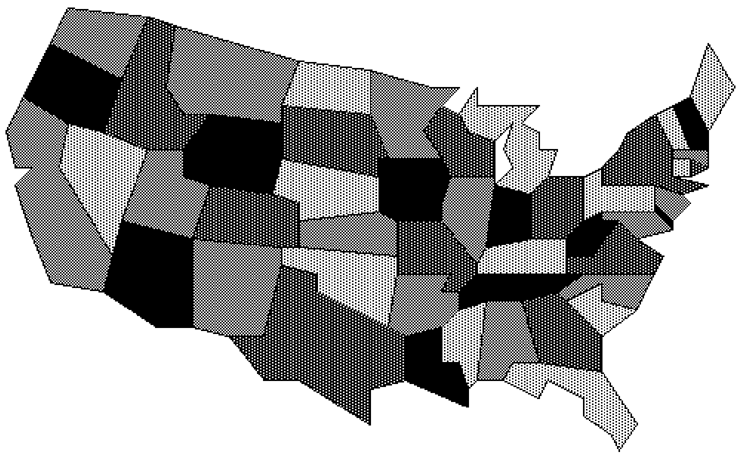


Figure 5. The USA map coloring

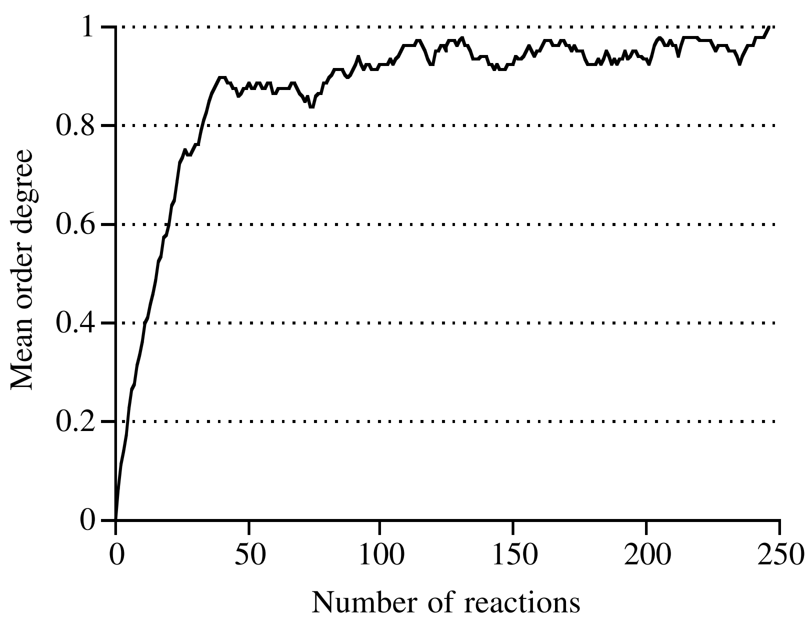


Figure 6. A sample time sequence of MOD in the USA map coloring

3.2 Basic method of Constraint Satisfaction

An example of non-fuzzy CSP and a method of solving it using CCM are demonstrated here.

Kanada [Kan 94a] describes a method of problem solving using CCM, and uses the *N* queens problem, which is a CSP, as an example. In this case, each constraint is expressed as an LOD between two atoms. The value of an LOD is higher when the constraint is satisfied, and is lower when the constraint is violated. The global MOD, or the total of LODs, is optimized in this way. This means that all the constraints can be satisfied by the system operation. The same method can be applied to other CSPs. A CCM-based system to solve a coloring problem is described below.

The problem is how to color the vertices of a graph using a specified number of colors, for example, four. Each pair of neighboring vertices must be given different colors. A map coloring problem can be converted to a graph coloring problem, if areas of the map are converted to vertices and the area borders are converted to edges. Thus, the map coloring problem can be solved by the same rules and LODs as the graph coloring problems. For example, the problem of coloring the graph with five vertices, shown in **Figure 7**, is equivalent to the problem of coloring the map with five areas, which is also drawn in Figure 7. The data structure used for solving the problem is as follows. Vertices are represented by atoms. An atom of type vertex has a color as its internal state. In Figure 7, c1, c2, c3 and c4 are the colors.

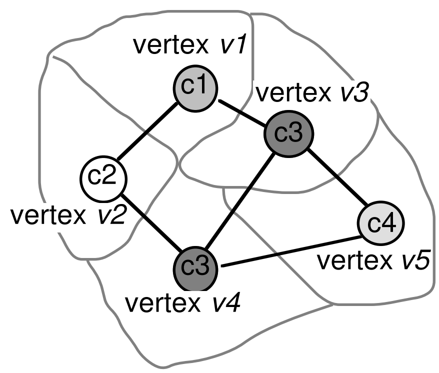


Figure 7. An example of a graph coloring problem and its representation

The reaction rule and LOD to solve the problem are shown below. The only reaction rule is illustrated in a graphical form in **Figure 8**. The same rule written in SOOC-94 is given in **Appendix 1**. This rule refers only to two neighboring vertices and the edge between them, and changes the color of one of the vertices randomly. The LHS of the rule contain two patterns; vertex1 and vertex2. C1 and C2 are variables to be matched to the colors of vertex1 and vertex2. There is a link between the atoms, and this link represents the edge between the vertices. Thus, vertex1 and vertex2 can only match two vertices that have a link between them. When a reaction occurs, the internal state of the atom that matches vertex1 is rewritten. That is, the color that was equal to C1 before the reaction becomes equal to C3, which is generated randomly. The value of C3 is selected from a predefined set of colors. Because there is no constraint between C1, C2 and C3, these colors can either be the same or different.[1]

[1] Because the color for variable C3 is generated randomly, the same color may be selected for C3 as for C1 or C2. However, the sum of LODs does not increase in the case of such a reaction, so the reaction does not occur. (See the definition of LOD given later in the present section.)

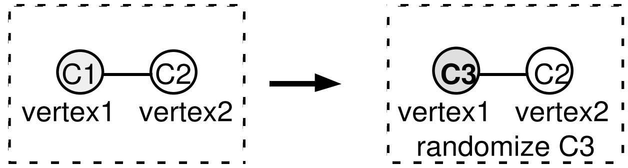


Figure 8. The reaction rule for the graph coloring system

The definition of the LOD is shown below. The LOD is defined between two vertices, *v1* and *v2*. Its value is 1 when the constraint between *v1* and *v2* is satisfied, and otherwise it is 0.

$$o(v1, v2) = \begin{cases} 1 & \textbf{if not } connected(v1, v2) \textbf{ or } v1.color \neq v2.color \\ 0 & \textbf{otherwise} \end{cases}$$

The same LOD written in SOOC-94 is also shown in **Appendix 1**. This definition means that the value of the LOD is 1 (higher) if the vertices are not connected or have different colors, and otherwise that it is equal to 0 (lower). The value of LOD is 1 when the vertices are not connected in this case. The value is also 1 when the vertices are connected and have different colors, because the constraint between the vertices is satisfied. The value is 0 when the vertices are connected and have the same color, because the constraint is violated. In general, a CSP can be expressed (but is not necessarily solvable) using CCM, if methods of state transition in the search space can be expressed as reaction rules, and the problem consists of local constraints that can be expressed by binary LODs.

A reaction occurs when two connected vertices, *V1* and *V2*, are selected by the system, and the following condition holds:

$$o(V1, V2) \leq o(V1', V2),$$

where *V1'* is *V1* after the reaction (with the new color). For example, an application of the rule to the above graph is illustrated in **Figure 9**. Vertices *v3* and *v4* are selected at random, and the system matches *v3* to vertex1 and *v4* to vertex2. *v3* and *v4* are the vertices in the graph and vertex1 and vertex2 are the patterns in the rule. Then, the sum of LODs before the reaction is 0 because the colors of *v3* and *v4* are the same, and the sum after the reaction is 1 because the colors will be different. Thus, this reaction actually occurs.

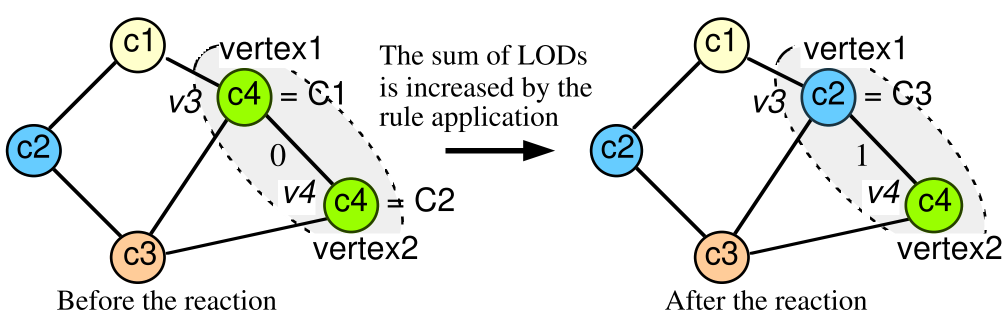


Figure 9. An application of the coloring rule

This is a conflicting system, and works stochastically. Thus, the sum of LODs does not monotonically increase, and the system does not necessarily find a solution in finite time. However, experiments show that the computation time is always finite in practice unless the problem is too difficult. When the system reaches a solution state, it becomes stationary, because no further reactions can occur.

A complete termination test can be performed by testing all the combinations of rules and atoms, if the system is closed. For example, it is sufficient to test four combinations for the water generation system shown in Figure 2. However, this method cannot be applied to open systems because the number of combinations is not known and varies. Also, this method maybe expensive because the minimum number of tests is sometimes difficult to be estimated. Thus, we usually use the following method for testing termination. If the system fails to find a set of a rule and matching data that can react by a fixed number of trials (10000 times, for example, depending on the problem to solve), then the system terminates. This method may cause termination before finding a solution. However, premature termination can be mostly avoided by setting an appropriate maximum number of trials, and the correctness of the result can easily be tested.[1] The system can change the maximum number of trials adaptively.[2]

A pattern of atoms, whose value is not changed by the rule, is called a *catalyst* [Kan 94a]. An atom that is not changed by a reaction is called a catalyst of the reaction. A pattern in a rule is also called a catalyst if the same (unchanged) pattern appears in both the LHS and RHS of the rule. In the rule in Figure 8, vertex2 is a catalyst. A rule with no catalyst is illustrated in **Figure 10** (a). If this rule is used instead of the rule in Figure 8, a reaction occurs unconditionally when the system selects a vertex for the reaction. This is because there is only one vertex pattern in the rule, and thus, there is no combination of atoms to compute in LOD. A rule with two or more catalysts can also be used. A rule with two catalysts is illustrated in Figure 10 (b). In this rule, vertex2 and vertex3 are the catalysts. If this rule is used, a reaction occurs when three connected vertices, *V1*, *V2* and *V3*, are selected for vertex1, vertex2 and vertex3 respectively by the system, and the following condition holds:

$$o(V1, V2) + o(V1, V3) + o(V2, V3) \le o(V1', V2) + o(V1', V3) + o(V2, V3),$$

where *V1*' is *V1* after the reaction. Simplifying this, we get

$$o(V1, V2) + o(V1, V3) \le o(V1', V2) + o(V1', V3).$$

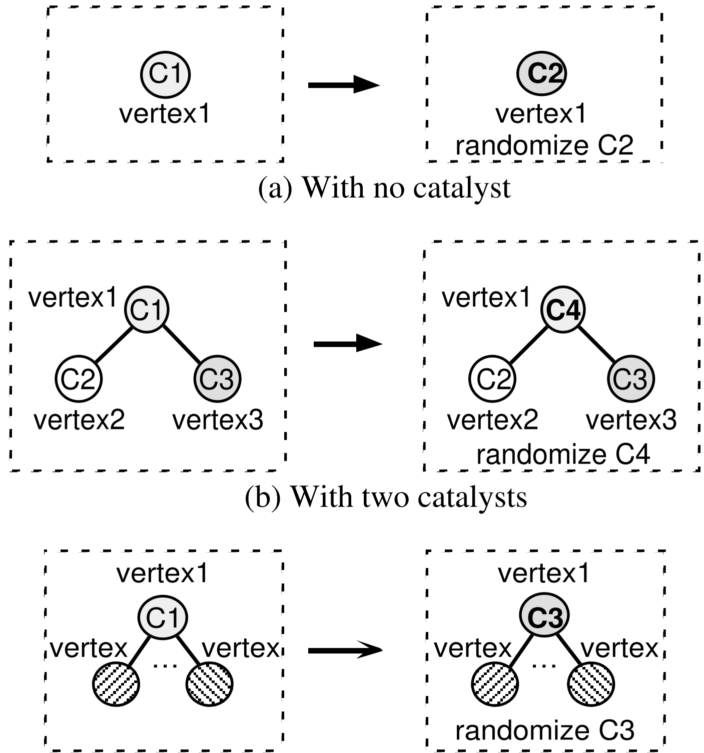
Addition or removal of catalysts changes the locality of the computation and changes the performance [Kan 94a, Kan 95]. In the coloring problem, a system with only the no-catalyst rule performs a complete random walk in the search space. The performance is out of the question because this system does not come to a stationary state even if a solution is found. If more catalysts are added, the number of reactions decreases, and the performance improves under certain conditions. The rules in Figure 8, and in Figure 10 (a) and Figure 10 (b) have a fixed number of catalysts. However, the number of catalysts can vary dynamically. In the rule in Figure 10 (c),

---

[1] The correctness can be checked using CCM by running an appropreate rule using a strict termination test method after the termination of the main computation process. This correctness check terminates rapidly in the coloring problem. Even if the number of tests in the termination test of the correctness check is much larger than the minimum, this process does not spend too long time. Thus, this method performs better than the method with strict termination test in the main computation process.

[2] In our implementation, the maximum number of trials is increased to 1/20 of the total number of LHS tests since the beginning of the computation when it is less. This can avoid most of premature terminations without adding significant overheads.

all the neighboring vertices of vertex1 are the catalysts, and the number of catalysts depends on the vertex matched to vertex1.



(a) With no catalyst

(b) With two catalysts

(c) With a variable number of catalysts

Figure 10. The reaction rules with zero, two, or variable number of catalysts

A slightly modified version of the rule and LOD, which is more complicated but performs better than the above rules and LODs, was coded by SOOC-94, and then executed. The coloring system on SOOC-94 was applied to the map of the USA mainland, consisting of 48 states [Tak 92] (see Figure 5). A correct solution was found in every run. The performance was measured and the effect of catalysts was shown by Kanada [Kan 95]. The result can be summarized as follows. When the number of catalysts increases, the number of reactions decreases but the execution time and the number of LHS matches does not necessarily decrease.

Coloring the USA mainland is a rather easy problem. Thus, it can be solved using the rule with only one catalyst. However, if the problem is more difficult, more catalysts are necessary. Otherwise, the computation time will be too long. If more catalysts are used, the computation can more easily fall into a “local maximum” of MOD [Kan 94a].[1] For example, a solution can usually be found by the rule with a variable number of catalysts and is more efficient than the single-catalyst rule. However, sometimes it fails to solve the problem but falls into a stationary

[1] Actually, such a state is not a local maximum in its strict meaning, because a CCM-based system does not perform hill-climbing, i.e., does not necessarily optimize a global function. However, in this expression, “local maxima,” seems to be intuitive. Exactly the system may come into a set of states, from which it cannot move by any reactions to a state that is not in the set. This set is the set of “local maxima.”

state, which is a "local maximum." Or, sometimes it flips randomly between several states, which are "local maxima."

4. FAM: An Annealing Technique

A method of escaping from "local maxima" in CCM is proposed here. This method is called *the frustration accumulation method* (*FAM*) [Kan 94b]. In this method, the same rule and LOD shown in the previous section are used. However, the mechanism of computation, or the compiler of SOOC, must be modified.

Each atom, or vertex, has a type of energy called *frustration*, whose value is positive, in this method. The frustrations of all the atoms to be modified by the reaction are subtracted from the sum of the LODs. Thus, if a vertex has a non-zero frustration, reactions occur more easily. For example, if the rule with one catalyst (in Figure 8) is used, a reaction occurs when the following condition holds:

$$o(V1, V2) - V1.frustration \le o(V1', V2) - V1.frustration',$$

where *V1.frustration* and *V1.frustration'* are the frustration of *V1* before and after the reaction. The frustrations of catalysts are not counted. If the rule with two catalysts (Figure 10 (b)) is used, the following condition is used instead:

$$o(V1, V2) + o(V1, V3) - V1.frustration \le o(V1', V2) + o(V1', V3) - V1.frustration'.$$

Each vertex initially has a certain level of frustration, f_0, which is, for example, 10^{-5}. The frustration is increased, when the application of a reaction rule fails but there are unsatisfied constraints. That means that the frustration of an atom is increased when the following three conditions hold.

1. The atom is tested for modification.

2. The reaction does not occur.

3. There are constraints, relating to the matched vertex which are not or will not be fully satisfied.

More specifically, if a rule and a set of atoms are tested for a reaction, if the reaction does not occur, and if the sum of the LODs is less than the maximum, then the frustration of each atom to be modified by the rule, f, is replaced by cf, where c (> 1) is a constant. This means that

$$f' = cf,$$

where f' is the frustration after the reaction. The value of c is 2, for example. Thus, the procedure that runs after each test for a reaction can be described as follows.

if a reaction occurred **then**[1]
    $f_i := f_0$;   /* reset to the initial value */
else if not all the constraints are satisfied **then**
    $f_i := cf_i$;   /* increased */
end if;

Assume that the sum of the LODs of the vertices matched to patterns in the rule before the reaction is represented by O, and that after the reaction is represented by O'. The sum of the frustrations before and after the reaction are represented by F and F'. Then the reaction occurs when

$$O - F \leq O' - F'.$$

Thus, the reaction occurs more easily when the value of F is larger.

An example is shown in **Figure 11**. The variable catalyst rule (Figure 10 (c)) is applied to the state shown in the upper figure. A reaction may or may not occur depending the color selected by the RHS of the rule. In case 1, no reaction occurs, and in case 2, a reaction does occur. The rule is applied to vertex *v3* and its three neighboring vertices. The sum of LODs before the reaction, o, is 2, because the colors of *v1* and *v3* are different, that of *v4* and *v3* are the same, and that of *v5* and *v4* are different (See the upper figure in Figure 11). The values labeled on the edges in the figures, 0 or 1, are the LOD values. The value of frustration is assumed to be 0.002, the value increased once from the initial value, $f_0 = 0.001$. The value of c is 2.

In case 1, the randomly selected color for variable C3 (in Figure 8) is c1. The sum of LODs after the reaction, o', is 1, because now the colors of *v1* and *v3* are the same. That of *v4* and *v3* are different, and that of *v5* and *v3* are the same. The sum of LODs would be decreased by the reaction. Thus, no reaction occurs, and the frustration of *v3* is increased to 0.004. The subtraction of frustration does not affect the comparison result in this case, i.e., $o' < o$ and $o' - f' < o - f$. If the value of frustration before the reaction were 1.002 or more, the subtraction of f' from o' would change the comparison result, i.e., $o' < o$ but $o' - f' \geq o - f$, and a reaction would occur.

In case 2, the randomly selected color is c2. The sum of LODs after the reaction, o', is 3, because the colors of all the neighrors of *v3* are different from that of *v3*. The sum of LODs would be increased by the reaction. Thus, the reaction occurs, and the frustration of *v3* is reset to the initial value, 0.001. A subtraction of frustration never changes the comparison result in cases that a reaction occurs.

---

[1] In this vertion of the algorithm, the frustration of an atom is not reset to f_0 when the constraints are satisfied by another reaction even if the atom is tested after the satisfaction. This statement can be modified as follows:

if a reaction occurred **or** all the constraints are satisfied **then**
    $f_i := f_0$;
else $f_i := cf_i$;
end if

Then, the frustration is reset when the atom is tested after the constraints are satisfied. This version also works. However, the parameter tuning seems to be more difficult.

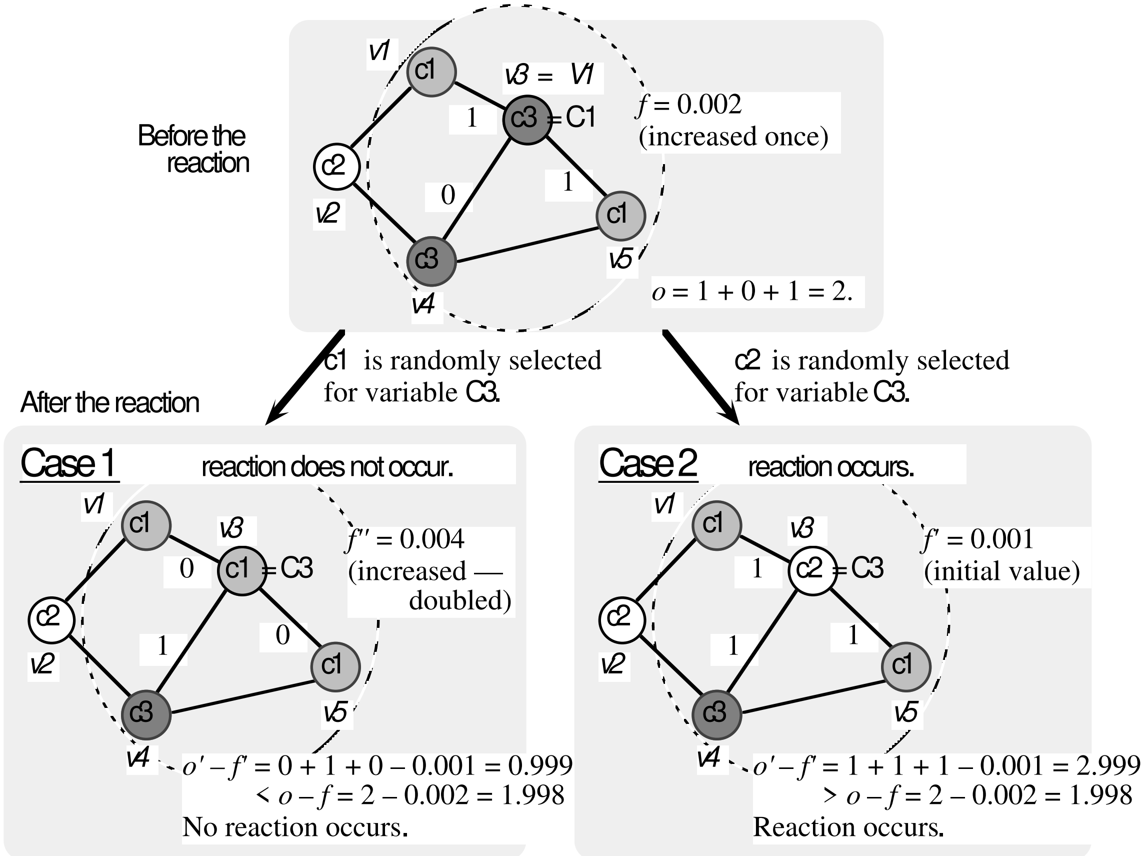


Figure 11. The outline of FAM

Parameters, f_0 and c, are constant during the execution, and they are considered to be properties of each atom or eac type of atoms. Thus, this method works only with local information and there are no global parameters, such as temperature used in simulated annealing. Both f_0 and c are constant for all the vertices in our implementation.

The system works qualitatively as follows. The average frustration increases rapidly when there are constraints that are difficult to be satisfied. This high frustration state corresponds to high temperature state in simulated annealing (SA) and other annealing techniques. If the values of parameters, f_0 and c, has been selected appropriately, the number of violations of constraints decreases, and the average frustration thus decreases. This frustration decrease corresponds to temperature decrease in SA, but the decrease occurs autonomously. No external control is required. If all the constraints are satisfied and all the frustrations become low enough, no state transition occurs anymore. Then the execution stops.[1]

Two sample time sequences of average frustration are shown in **Figure 12**. The problem used is the USA mainland map coloring. The horizontal axis shows the number of tests for reactions (i.e., the number of executions of the LHS of the rule), and the vertical axis shows the average frustration. The stars in the upper part of the graph show the reactions caused by frustration

[1] The termination condition in the algorithm is replaced by “false.” This means that the network operates forever. Then, the network once stops changing its state but begins changing its state again when the state of a neuron is changed or new constraints are added externally. Thus, a dynamic problem or open-ended problem can be solved without an additional mechanism in this method.

accumulation. (Their vertical coordinate has no meaning.) A solution was found by 110067 tests in the first case and by 19506 tests in the second case. Average frustration sometimes increases rapidly and this causes a reaction. Then, the frustration decreases rapidly.[1] Average frustration sometimes decreases without a reaction caused by a frustration accumulation.

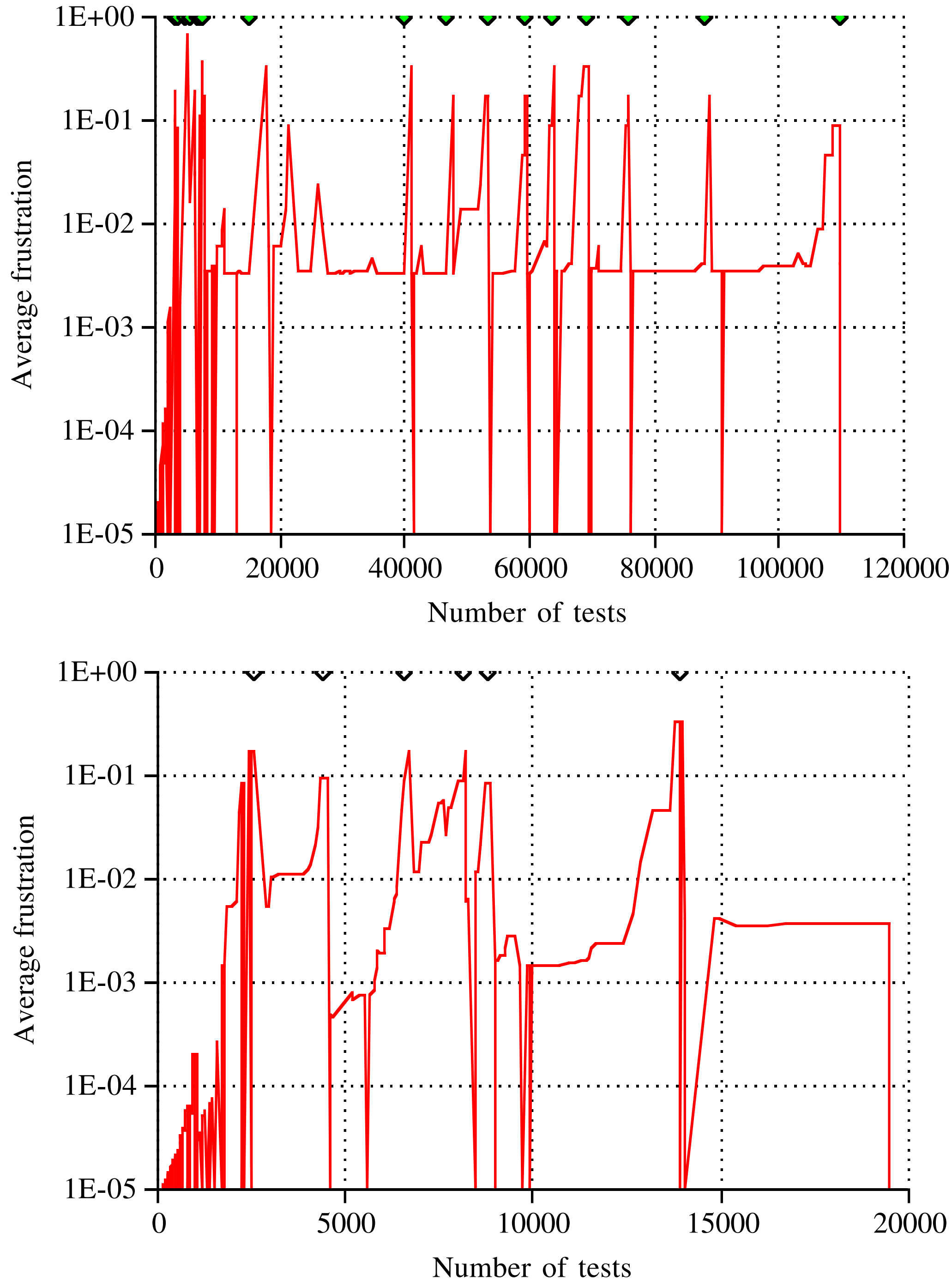


Figure 12. Two sample sequences of average frustration

[1] The frustration of the neurons, in which all the constraints are satisfied, are regarded to be the initial value, although the actual values stored are larger in this measurement.

A hypothesis on the effect caused by FAM can be illustrated in **Figure 13**. The vertical axis of this figure conceptually shows the mean value of “LOD – frustration.” This value is denoted as MOD_f here. It is assumed that there is a peak of an MOD_f, i.e., a local maximum. The computation process cannot pass throuth this peak. So it is impossible to reach the solution using CCM without annealing. This peak is shaved by FAM, i.e., the frustration makes the peak lower. Thus, the computation process can pass throuth this peak and find a solution. Local maxima are shaved because the computation process stays in states[1] with a high MOD value more frequently than states with a low MOD value, and the frustration of a state that the system stays in more frequently is increased more.

This hypothesis is not yet proved, but it is likely to be true. Selman and Kautz [Sel 93a] and Morris [Mor 93] proposed methods of solving satisfiability problems (see Section 7). They use methods of avoiding local maxima by adding weights to clauses. Selman and Kautz stated that their method works as filling local minima. Although the basic method of problem solving is different, this method of “annealing” seems to have similarities to ours.

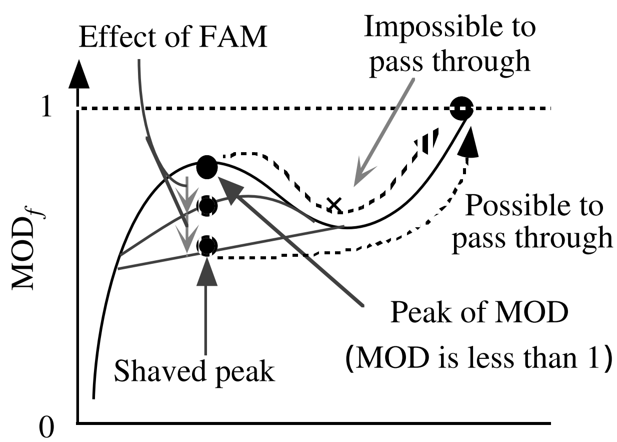


Figure 13. A rough image of the effect of FAM

Frustrations may be increased additively instead of multiplicatively as above. That means, the frustration, f, may be replaced by $f + k$ when f is to be increased, i.e.,

$$f' = f + k,$$

where k (> 0) is a constant.[2] However, if the frustration increase is additive, the appropriate range of parameters will become narrower, because it takes a long time to shave a deep valley of MOD using additive frustration. The additive frustration is mentioned again in the performance evaluation (Section 6.3).

So far, FAM has been described along with CCM. However, FAM can also be applied to other local-function-based architectures such as neural networks. For example, the frustration can be the parameter of threshold function of neurons.

---

[1] A state does not mean an element of the global state space here. If the MOD or MOD_f is defined on a collection of atoms. The state is an element of the Cartesian product of the state spaces of these atoms.

[2] Several variations of the method of increasing/decreasing frustrations are available. Additive frustration is one of them. However, the method described above is the best tested so far.

5. Method of Parallel Processing Using Annealed CCM

The parallel version of the method of constraint satisfaction using annealed CCM (i.e., CCM with FAM) is shown in the present section.

Reactions may occur in parallel in CCM. Thus, if the working memory is shared between parallel processors, the computation is parallelized.[1] This is shown in **Figure 14**. However, if two processes try to modify an atom concurrently, a wrong result may be obtained in general. The analogy to chemical reaction systems is useful here. In chemical systems, two reactions that affect the same bond do not occur at the same time. The same principle must be applied to CCM. Atoms to be modified by the reaction must be accessed exclusively.

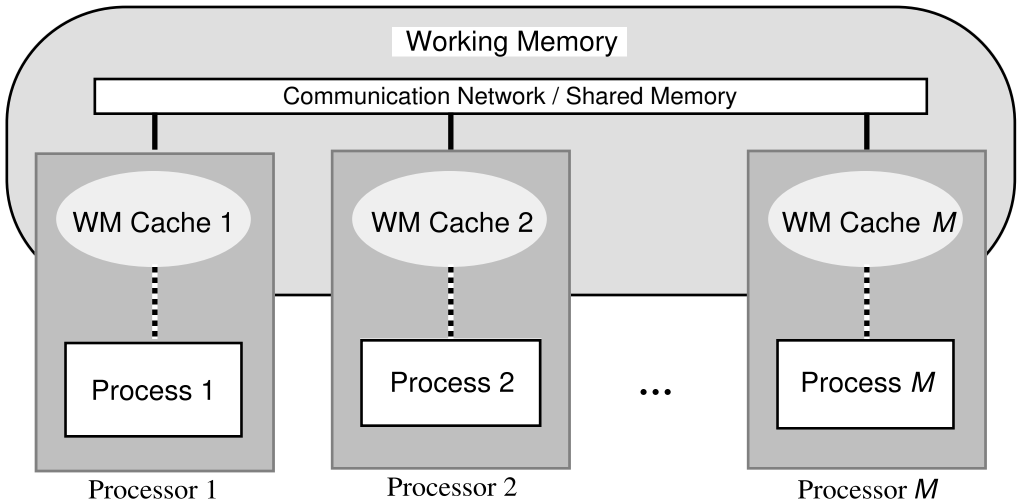


Figure 14. Parallel processing of CCM-based computation

However, fortunately, because the rules of coloring modify only one atom at a time, no wrong result can be obtained by parallel reactions without explicit mutual exclusion in this case. The only possible new effect caused by parallel reaction is that the sum of LODs may be decreased accidentally by parallel reactions even when the frustration is small. This is because, after a process tested the reaction condition but before rewriting an atom, another process may rewrite the atom. This effect seems to be small because the probability of two or more reactions occurring on the same atom concurrently is small, if there are enough atoms and the degree of parallelism is not massive. Thus, no mutual exclusion seems to be necessary here. This non-strictness in parallel processing is made possible by the stochastic nature of CCM.

The following problem has to be solved to improve performance. Each parallel process works independently in principle in this method. A process tests termination condition without testing the states of other processes if it follows the principle strictly. However, this sometimes causes premature termination. Some processor (process) may run for long time after other processors terminate, because the termination test is stochastic. This load imbalance lowers the degree of parallelism, and causes performance degradation. This actually occurred in our experiments.

---

[1] The sharing of working memory is conceptual. Thus, it might not be shared physically but is implemented using process communication.

A typical example of this load imbalane is shown in **Table 1**. The number of reactions and the number of tests (of the LHS of the rule) until the process termination is listed in this table. The processors except processor #3 terminated earlier, and thus, the processing is almost sequential.

Table 1. Example of load imbalance
(DSJC125.1, 5 colors, 12 processors, $f_0 = 10^{-5}$, $c = 2$)

Processor #	0	1	2	**3**	4	5	6	7	8	9	10	11
# of reactions	171	242	215	**23637**	254	222	262	204	190	231	254	177
# of tests	12786	11060	13805	**2646845**	11390	13423	13986	10685	11502	10834	11739	12212

To solve this load imbalance problem, weak global communication is introduced. The termination test is modified as follows. A counter variable is allocated and is set to the number of processes when the system is initialized. Each process decrements the counter (where mutual exclusion is necessary) when its local termination condition becomes satisfied. However, the process continues its task until the counter value becomes zero. This technique avoids premature termination and the consequent performance degradation.

6. Experiments on Large-scale Problems

The method of problem solving using annealed CCM has been applied to several large-scale problems. Both sequential and parallel performance, and the relationship between parameter values and performance are shown here.

6.1 Sequential performance

The problems used to evaluate performance are chosen from DIMACS benchmarks [Tri][DIM]. Some have been used by Johnson, et al. [Joh 91] and also used by Selman and Kautz [Sel 93a]. These problems are static and not the real target of CCM, but are used here mainly for comparisons.

The results are shown in **Table 2**. The graphs used here, which were randomly generated and used by Johnson et al., have 125 or 250 vertices, and the probability of existence of edge, p, is 0.1 to 0.9. The performance of annealed CCM is comparable to GSAT or the methods tested by Johnson et al. in several cases. The parameters used here, which are kept constant during the computation, are shown in the table. Each case has been measured at least 20 times, and the results shown in the table are the averages except the cases in which the system failed to find a solution. The probability that the system fails to find a solution is less than 0.05 experimentally, with appropriate parameter values. Although a compiler and interpreter for SOOC-94 is available, a reaction rule and LOD coded using C are used in these measurements, for better performance of the program written by C. The performance has been measured on a Cray Superserver 6400 (CS6400), which is a parallel computer with 60MHz SuperSPARC processors, but only one CPU is used here.

Table 2. The performance of coloring by annealed CCM compared with several annealing methods and GSAT (c = 2)

Graph Name (DSJC-#vertices.p)	Number of colors	CPU time (sec)			
		Annealed CCM (CS6400, Single CPU)	Annealed CCM (CS6400, 12 CPUs)	Johnson et al. (Best methods, Sequent Balance 21000)	GSAT (Best method, MIPS R6000)
DSJC125.1	6	0.2 ($f_0 = 10^{-5}$)	0.2 ($f_0 = 10^{-5}$)	< 360	
DSJC125.1	5	13.2 ($f_0 = 10^{-5}$)	1.5 ($f_0 = 10^{-5}$)	720	
DSJC125.5	18	34.0 ($f_0 = 10^{-15}$)	2.4 ($f_0 = 10^{-15}$)	360	44
DSJC125.5	17	3113 ($f_0 = 10^{-30}$)	149 ($f_0 = 10^{-30}$)	6700	1800
DSJC125.9	44	208 ($f_0 = 10^{-30}$)	13.6 ($f_0 = 10^{-30}$)	1080	
DSJC125.9	43	Not solved	Not solved	Not solved	
DSJC250.1	9	2.9 ($f_0 = 10^{-4}$)	0.3 ($f_0 = 10^{-4}$)	< 360	
DSJC250.1	8	298 ($f_0 = 10^{-10}$)	17.6 ($f_0 = 10^{-10}$)	9360	
DSJC250.5	31	239 ($f_0 = 10^{-35}$)	18.9 ($f_0 = 10^{-35}$)	360	
DSJC250.5	30	513 ($f_0 = 10^{-35}$)	47.9 ($f_0 = 10^{-35}$)	2900	
DSJC250.5	29	5111 ($f_0 = 10^{-45}$)	360 ($f_0 = 10^{-45}$)	22000	4000

6.2 Parallel performance

The parallel performance of the same problems, as shown in the previous section, has been measured on CS6400 with 12 processors. It has been measured using a relatively small number of processors because the performance will not improve much by using hundreds or thousands of processors; only 125 atoms are used in these problems.

The performance of four problems, DSJC125.1 with 5 and 6 colors, and DSJC125.5 with 17 and 18 colors, is displayed in **Figure 15**. The parameter values of FAM used in the evaluations are fixed; $q = 8$, $c = 1.05$, $f_0 = 0.8$. Each case is measured at least 20 times, and the results are the averages. The performance of the 5 and 17 color problems is close to ideal. They are accelerated nearly linearly. However, the performance of the 6 and 18 color problems is less than the half the ideal. The reason for this is not yet known, but the performance is known to be better for some other parameter values. For example, in the case of 18 colors with $M = 12$, $c = 1.2$ and $f_0 = 0.2$, the acceleration ratio was 10.3.

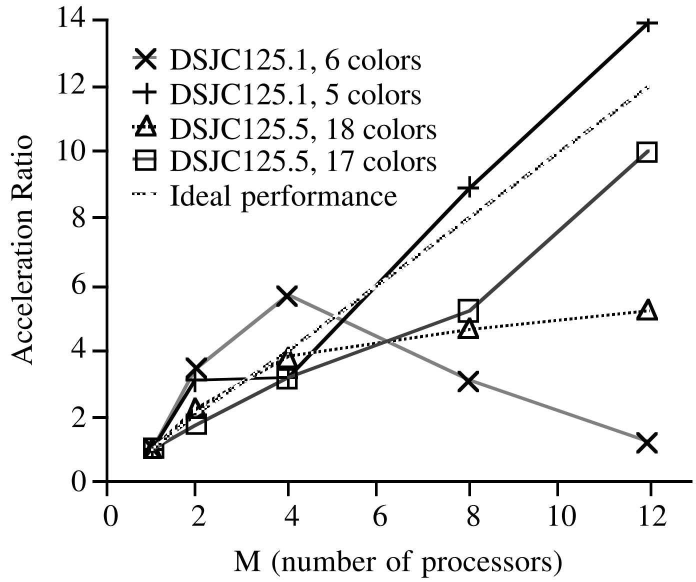


Figure 15. Performance of coloring problems by parallel annealed CCM

The effect of load balancing is measured for DSJC125.1 with 5 colors. The result for 100 runs is shown in **Table 3**. The standard deviation of CPU time is reduced to 40 %, and the average CPU time is reduced to 75 %.

Table 3. The effect of load balancing
(DSJC125.1, 5 colors, 12 processors, $f_0 = 10^{-5}$, $c = 2$)

Load balance	Average CPU time (sec)	Standard deviation of CPU time (sec)
Balanced	15.1	11.5
Not balanced	20.1	28.4

6.3 Relationship between parameter values and performance

The relationship between parameters, f_0 and c, and the performance are measured. The results in the cases of sequential processing and parallel processing with 12 processors are shown in **Figure 16** and **Figure 17**. Figure 16 shows that the range of f_0, in which the performance is better, is wider, if c is large. Figure 17 shows the complementary result that the range of c in which the performance is better, is wider for small f_0. This means that, if the value of c is large enough and the value of f_0 is small enough, the performance is not very sensitive to either c or f_0, and thus, parameter tuning is easy.

When c is smaller, the increase of frustration is almost linear. Thus, this result also shows that multiplicative frustration is better than additive frustration, as has been already mentioned in Section 4.

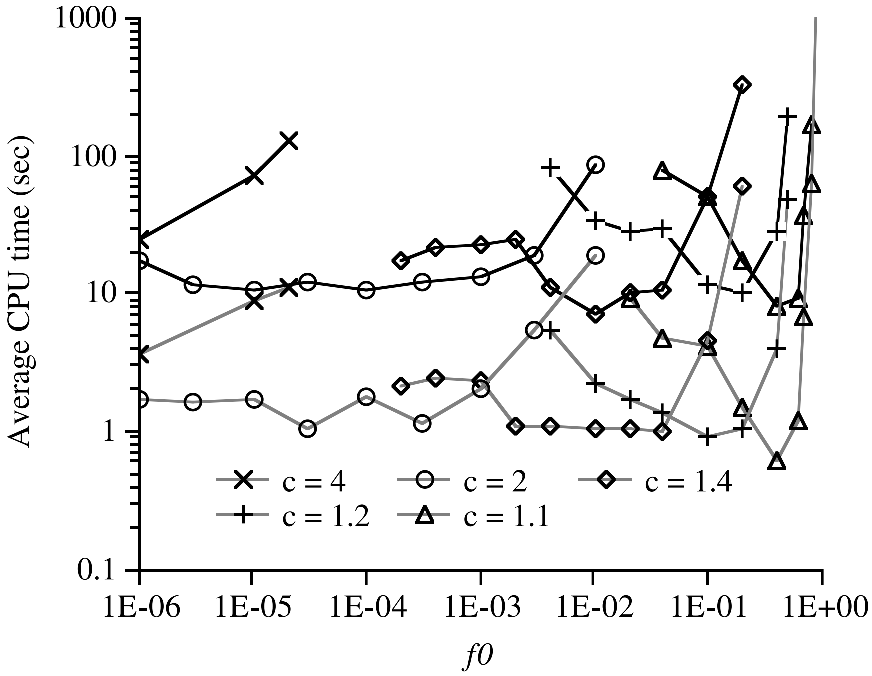


Figure 16. Relationship of f_0 and performance (DSJC125.1, 5 colors, 1 or 12 processors)

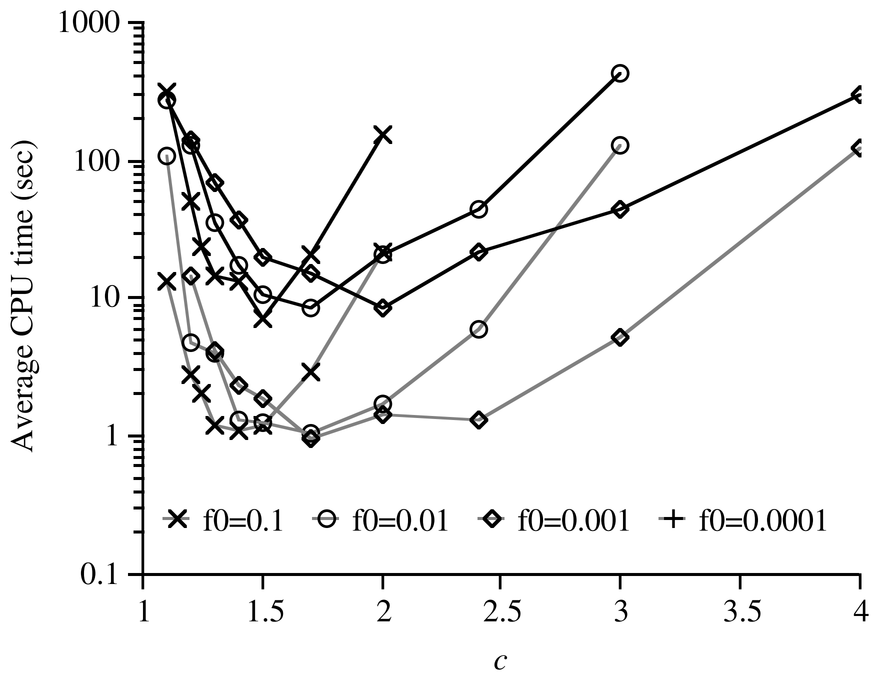


Figure 17. Relationship between *c* and performance (DSJC125.1, 5 colors, 1 or 12 processors)

7. Related Work

The basic mechanism of neural networks works only with local information if the neurons are connected locally, i.e., connected to a limited number of neurons. However, for the purpose of simulated annealing, The Boltzmann Machine [Ack 85] introduced a global mechanism. The behavior of the Boltzmann Machine is controlled by temperature, which is a global and top-down parameter. Thus, this mechanism is quite different from that of FAM in terms of locality or emergence. Most methods using annealing are similar to the Boltzmann Machine in this respect.

CCM with FAM has some similarity to several problem solving methods for solving CSPs or satisfiability problems (SATs) using a global evaluation function, as developed by Minton [Min 92], Selman [Sel 92], Gu [Gu 93], and others. For example, GSAT [Sel 92] is a greedy search method for solving SATs using an evaluation function that represents the number of satisfied propositional clauses. Very difficult SATs and difficult CSPs, which can be converted to SATs, have been solved using improved versions of GSAT [Sel 93a, Sel 93b]. This method uses a type of annealing, but does not require detailed knowledge on the problem domain.

However, these methods also depend on global information; the dynamics depend on a global evaluation function. Thus, these methods can only be applied to closed problems, and parallel processing is difficult because of global data dependence.

8. Conclusion

A method for solving constraint satisfaction problems using the chemical casting model (CCM) with the frustration accumulation method (FAM), which is a type of annealing method, is proposed in the present paper. FAM makes large-scale constraint satisfaction using CCM possible, without spoiling the feature of CCM, i.e., local-information-based computation. No global functions or global parameters such as temperature are used in FAM.

Experiments show that the performance of this method is not very sensitive to parameter values, which means that parameter tuning is easy. The performance is comparable to conventional simulated annealing or GSAT, which are based on global evaluation functions, for several problems. Because of the nonexistence of global information references, CCM with FAM can be parallelized very ealily with little mutual exclusion. The performance is improved nearly linearly by parallelization in cases such as graph coloring problems in DIMACS benchmarks.

There are several possible future research directions. A method of tuning the parameter values of FAM must be developed. Implementation and performance evaluation of the method on a parallel computer without shared memory is also necessary. Application of FAM to other architectures, such as neural networks, should be examined. Finally, the statement that local-information-based computation makes problem solving in open and changing environment much easier must be proved.

Acknowledgment

The author thanks to Susumu Seki and Hironobu Takahashi of Tsukuba Research Center of Real World Computing Partnership, for their assistance in use and programming of the Cray Super-server 6400.

References

[Ack 85] Ackley, D. H., Hinton, G. E., and Sejnowski, T. J.: A Learning Algorithm for Boltzmann Machines, *Cognitive Science*, 9, 147–169, 1985.

[DIM] Center for Discrete Mathematics and Theoretical Computer Science, *http://dimacs.rutgers.edu/*.

[For 81] Forgy, C. L.: *OPS5 User's Manual*, Technical Report CMU-CS-81-135, Carnegie Mellon University, Dept. of Computer Science, 1981.

[For 91] Forrest, S., ed.: *Emergent Computation*, MIT Press, 1991.

[Gu 93] Gu, J.: Local Search for Satisfiability (SAT) Problem, *IEEE Trans. on Systems, Man, and Cybernetics*, 23:4, 1108–1129, 1993.

[Joh 91] Johnson, D. S., Aragon, C. R., McGeoch, L. A., and Schevon, C.: Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning, *Operations Research*, 39:3, 378–406, 1991.

[Kan 92] Kanada, Y.: Toward Self-organization by Computers, *33rd Programming Symposium*, Information Processing Society of Japan, 1992 (in Japanese).

[Kan 93] Kanada, Y.: Features of Problem-Solving Method using Computation Model CCM, based on Production Rules and Local Evaluation Functions, *Summer United Workshops on Parallel, Distributed, and Cooperative Processing '93 (SWoPP '93)*, Information Processing Society of Japan, 1993 (in Japanese).

[Kan 94a] Kanada, Y., and Hirokawa, M.: Stochastic Problem Solving by Local Computation based on Self-organization Paradigm, *27th Hawaii Int'l Conf. on System Sciences (HICSS-27)*, 82–91, 1994.

[Kan 94b] Kanada, Y.: Methods of Controling Locality in Problem Solving using CCM: A Model for Emergent Computation, *Summer United Workshops on Parallel, Distributed, and Cooperative Processing '94* (*SWoPP '94*), 94-AI-95-4, 29-38, 1994 (in Japanese).

[Kan 95] Kanada, Y.: Fuzzy Constraint Satisfaction Using CCM — A Local Information Based Computation Model, *FUZZ-IEEE/IFES '95*, 2319–2326, Yokohama, Japan, 1995.

[Kan 96] Kanada, Y.: CCM: A Model for Emergent Computation and Its Application to Combinatorial Problems, *IEEE Trans. on Systems, Man, and Cybernetics*, Submitted.

[Lan 89-94] Langton, C. G. et al. eds.: *Artificial Life I, II and III*, Addison Wesley, 1989, 1991 and 1994.

[Min 92] Minton, S., Johnston, M. D., Philips, A. B., and Laird, P.: Minimizing Conflicts: a Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems, *Artificial Intelligence*, 58, 1992, 161–205, 1992.

[Mor 93] Morris, P.: The Breakout Method For Escaping From Local Minima, *AAAI 93*, 40–45, 1993.

[New 72] Newell, A., and Simon, H. A.: *Human Problem Solving*, Englewood Cliffs, Prentice-Hall, N. J., 1972.

[Sel 92] Selman, B., Levesque, H. J., and Mitchell, D. G.: A New Method of Solving Hard Satisfiability Problems, *AAAI '92*, 440-446, 1992.

[Sel 93a] Selman, B., and Kautz, H.: Domain-Indepen-dent Extensions to GSAT: Solving Large Structured Satisfiability Problems, *Int'l Joint Conf. on Artificial Intelligence '93 (IJCAI '93)*, 290–295, 1993.

[Sel 93b] Selman, B., and Kautz, H. A.: An Empirical Study of Greedy Local Search for Satisfiability Testing, *AAAI '93*, 46–51, 1993.

[Tak 92] Takefuji, Y.: *Neural Network Parallel Processing*, Kluwer Academic Publishers, 1992.

[Tri] Tricks, M.: The Second DIMACS Challenge, *http://mat.gsia.cmu.edu/challenge.html*.

Appendix 1. The Coloring System in SOOC-94

The declarations of a type, a rule, and an LOD for the coloring problem in SOOC-94 is described below. The rule and LOD are equivalent to those shown in Section 3.2.

```
(defelement vertex
                 ; Declaration of type vertex. A type is called an "element" in SOOC-94.
  color          ; vertex has an element named color.
  (* next)       ; vertex has arbitrary number of (zero or more) elements named next.

(defrule coloring      ; Declaration of rule named coloring.
  (var vertex1 vertex2 C1 C2 C3)        ; variable declaration
  (exist vertex vertex1 :next vertex2 :color C1)
  (exist vertex vertex2 :next vertex1 :color C2)
      ; The link between vertex1 and vertex2 are bidirectional, rather than non-directional.
  -->
  (select C3 `(red yellow blue green))      ; random selection from four colors
  (exist vertex vertex1 :color C3)
  (exist vertex vertex2)) ; This pattern, a catalyst, can be ommitted from the RHS.

(deforder ((vertex v1) (vertex v2))[1]
                 ; Declaration of LOD between two vertices.
  (if (or (connected v1 v2)
          (not (eql (vertex-color v1) (vertex-color v2))))
    1
    0))
```

[1] Declarations of LODs between two or more atoms have not yet been implemented to the SOOC-94 compiler. Thus, this declaration must be expressed in Lisp to be run. The LOD can be declared as an LOD for a single vertex. Then it can be expressed by SOOC-94.