# Combinatorial Problem Solving Using Randomized Dynamic Composition of Production Rules

**Yasusi Kanada**
Tsukuba Research Center, Real World Computing Partnership
Takezono 1-6-1, Tsukuba, Ibaraki 305, Japan
E-mail: *kanada@trc.rwcp.or.jp*, WWW: *http://www.rwcp.or.jp/people/yk/*

***ABSTRACT***

*The present paper proposes a method of solving combinatorial problems using randomized dynamic rule composition. This method is called CCM* and is based on a computational model called Chemical Casting Model (CCM), which is a rule-based computational model for emergent computation. CCM was proposed by the author toward solving dynamic, open and incompletely specified problems using a few simple rules and evaluation functions. By composing a rule from a given production rule dynamically and randomly, CCM* makes it possible to escape from local maxima, which cannot be escaped from by applying the original rule. This method is compared with the original CCM and another extended version of CCM, i.e., CCM with simulated annealing. 0–1 integer programming problems are solved using these methods. Our experiments show that CCM* performs much better than both the original and annealed CCM. In addition, suboptimal solutions can be found in less time than a branch-and-bound method.*

## 1. Introduction

Most combinatorial optimization problems (COPs) in Operations Research and constraint satisfaction problems (CSPs) in Artificial Intelligence are classified as NP-complete or NP-hard problems. They are sometimes impossible to be solved in polynomial time even when finding an approximate solution [1]. However, these problems are still static. Constraints and objective functions, included in these problems, are not varied while solving them. Real-world problems are not necessarily expressed by static constraints and objective functions. New information may be added or preexisting information may be dynamically changed by environmental change while solving problems.

Kanada [4, 6] proposed a computational model called CCM (Chemical Casting Model), which aims to solve problems by using self-organizing or emergent computation [2] in such situation described above. Complete information, which concerns the problem solving, cannot be grasped beforehand in such real-world problems [3]. However, conventional combinatorial problem solving methods, including branch-and-bound methods [13], simulated annealing, evolutionary computation, and so on, usually requires complete information. These methods require an objective function, which is defined using global information, to be specified completely. These methods are considered to be weak when the environment changes. CCM is developed for computation based on local, partial and incomplete information.

CCM has been applied to classical CSPs and COPs. We have applied CCM to such closed and fully informed problems, because real-world problems are complex and we had to add ability to solve complex problems to CCM. CCM has been applied to the *N*-queens problem [6], coloring and fuzzy coloring of graph vertices or maps [7], the traveling salesperson problem [5], and so on. Small-scale problems are proved to be solved using CCM with less chance to fall into local maxima or non-solution final state, because of emergent nature of CCM [6]. However, larger and more complex problem cannot be solved by too local computation, sometimes because it takes too much CPU time, or because the computation can still be trapped by local maxima. Thus, methods of organizing local parts of computation into non-local fragments must be developed. A method of controlling locality to reduce the computation time by adding or removing *catalysts* is proposed [6]. However, this method is not universal and does not solve the problem of local maxima in some cases.

A method of solving combinatorial problems using CCM*, which is an extended version of CCM, solves this problem. A mechanism of organizing computation using rule composition is added in this model. By composing new rules from given rules dynamically and randomly, this method makes it possible to escape from local maxima, which cannot be escaped from by applying the original rule. Another improvement to CCM, a type of simulated annealing, is also proposed and compared with CCM*. CCM and two possible improvements to CCM are explained in Section 2. A method of solving complex combinatorial problems using CCM* and its application to COPs are explained in Section 3. The results of its application to 0–1 integer programming problems are shown in Section 4. Related work is mentioned in Section 5. The conclusion is described in Section 6.

## 2. Computational Model CCM and Two Extensions

This section explains computational model CCM and two extensions of CCM to solve complex problems.

### 2.1 Chemical Casting Model

CCM is a computational model based on a production system. Production systems are often used for developing expert systems or modeling human cognitions. However, CCM differs from conventional production systems in the following two respects. Firstly, evaluation functions, which are called local order degrees (LODs), are evaluated when applying rules. Secondly, stochastic control or randomized ordering of rule applications, is applied. Rules and functions are computed only using local information.

The system components of CCM are shown in **Figure 1**. The set of data to which the rules apply is called the *working memory* (WM). Rules and functions change the state of WM. Thus, they determine the behavior of the system. The collection of rules and functions, which is usually called a program, is called a *caster* in CCM.[1] A unit of data in the WM is called an *atom*. An atom has a type and an internal state, and may be connected to other atoms by *links*. Links are analogous to chemical bonds with the difference that chemical bonds have no direction but links may have.

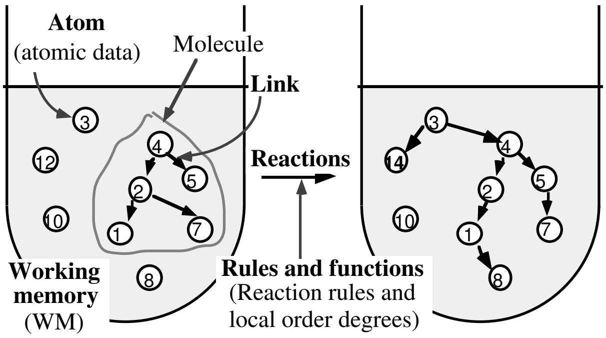


**Figure 1. The elements of CCM**

*Reaction rules* change the state of WM locally. "Locally" means that the number of atoms referred by a reaction rule is small.[2] The reaction rules are written as so-called forward-chaining production rules, such as chemical reaction formulae or as rules used in expert systems. The abstract syntax of reaction rules is as follows: LHS $\rightarrow$ RHS. The left-hand side (LHS) or "reactants," and the right-hand side (RHS) or "products" are sequences of patterns. For example, the following reaction rule swaps the internal states of two (unlinked) atoms, $A_1$ and $A_2$:

$$A_1(S1), A_2(S2) \rightarrow A_1(S2), A_2(S1).$$

The internal states are represented by S1 and S2 here.

A reaction rule can be activated when there is a set of atoms that matches the LHS patterns. If a 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 graph vertex coloring problem or the 0–1 integer programming problem. 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 (local objective functions). They are defined using only local information, and have a larger value when the local state of the WM is *better*. An LOD may be regarded as a negated energy. Thus, it is analogous to bonding energy in chemical reaction systems. Although LODs can be built into normal production rules, the separation of LODs from rules causes flexibility. For example, performance improvement by adding catalysts [6] or rule composition is made possible by this separation.

A reaction takes place when the following two conditions are satisfied. Firstly, there is an atom that matches each pattern in the LHS. Secondly, the sum of the LODs of all the atoms that concern the reaction, i.e., that appear on either side of the reaction rule, is not decreased by the reaction. Reactions repeatedly and stochastically (randomly) occur while the above two conditions are satisfied by any combination of a rule and atoms. The system stops when such a combination is exhausted. However, reactions may occur again if the WM is modified because of changes of the problem or the environment. Thus, the system using CCM can solve the dynamical problems.

In general, there may be two or more combinations that satisfy both conditions at once. There are two possible causes that generate multiple combinations. One cause is that there are two or more combinations of atoms that match the LHS of a rule. The other cause is that there are two or more rules, under which there are patterns in the LHS that match atoms. In each case, the order of the reactions (the selection order of such combinations) and whether they occur in parallel or sequentially can be determined stochastically. The CCM-based systems that we have developed uses random numbers for selecting rules and atoms. We have also experienced both sequential and parallel processing [8] of CCM-based systems. A single caster can be used both in sequential and parallel processing.

### 2.2 Two improvements to CCM

There are at least two possible improvements for CCM to solve complex combinatorial problems.

The original CCM has the following two drawbacks. One drawback is that a CCM-based system may easily be trapped by a local maximum. The other drawback is that computation in a CCM-based system may sometimes be very inefficient, because it refers only local information, i.e., only a few objects. The system may cause reactions that do not help solve the problem and may waste CPU time.

The first possible improvement, which remedies the first drawback, is simulated annealing. In the original CCM, a reaction never occurs when the difference of the LOD sums of RHS and LHS decreases. In the annealed CCM, a reaction is defined to occur in certain probability even when the difference decreases. The relation between the difference and the probability is defined by a Sigmoid function $\left(f(x) = 1 / (1 + e^{-x/T})\right)$ similar to the Boltzmann Machine, as shown in **Figure 2**. The temperature, $T$, is decreased toward zero while computing, following an appropriate scheduling. If the initial temperature is zero, this mechanism is almost the same as the original CCM.[3]

[1] The word "program" is not used in CCM because a "program" means a whole and complete plan.

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

[3] The behavior is different only when the difference is zero.

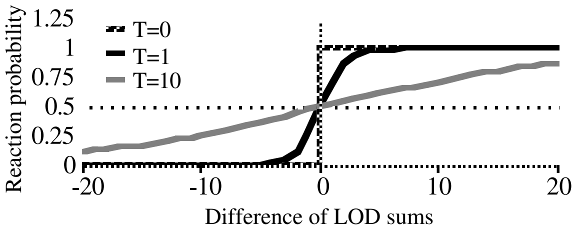


**Figure 2. Reaction probability in the annealed CCM**

Optimum solutions can probably be found by the annealed CCM. However, this extended version is expected to take much more time than the original CCM, which is already slow, because this simulated annealing is very near-sighted, i.e. , no non-local state transition can occur. This method has another problem that, because the temperature is a global parameter, its existence opposes the policy of CCM, i.e., local-information-based computation.

Both drawbacks described above can be remedied by the second possible extension, i.e., dynamic rule composition. If the original rules are applied twice or more times without calculating the sum of LODs in between, and compare the sum after the application to the sum before, both local maxima and inefficiency can be avoided. The system can skip over a valley of MOD, as illustrated in **Figure 3**. This process is nearly equivalent to a reaction caused by a rule that is a composition of the original rules twice or more times. This conceptual rule composition takes place dynamically and randomly. [1] This improved version of CCM is called CCM*.

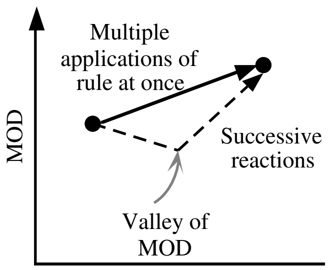


**Figure 3. Skipping over a valley of MOD by a rule composition**

## 3. Solving Complex Combinatorial Problems Using CCM*

This section explains the principle of problem solving using CCM and a method of solving a COP.

### 3.1 Principle of problem solving using CCM

Problem solving can be regarded as activity to find a *better* or *best* state in some sense. The meaning of words, *better* or *best*, may be different among problems to be solved, and "state" may mean 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.[2] Although CCM-based systems do not compute MODs, they operate so as to increase MODs or sums of LODs stochastically. The data in the WM can be divided into small parts or into large parts, or into parts of any other scale. MODs can also be defined in each scale. **Figure 4** illustrates the relation between MODs of different scales. They may conflict, e.g., 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 [6]. Cooperative systems perform hill-climbing.

Although different MODs may behave differently, MODs increase in average in every scale if the LODs and rules are defined properly. 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 COPs can be solved if a more optimized state is defined to have a higher MOD value. **Figure 5** shows an example time sequence of global MOD in the eight queens problem [6]. The MOD sometimes decreases, because this is a conflicting system. However, experiments show that this system never fails to find a solution.

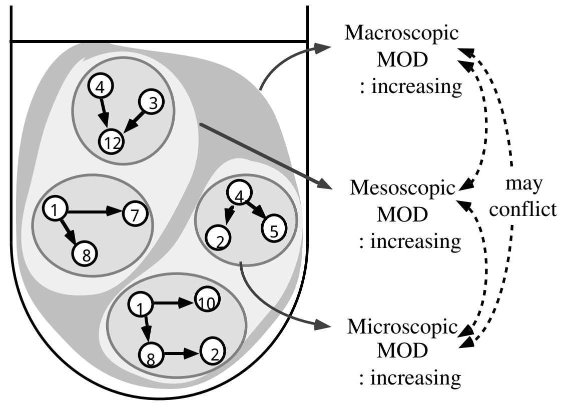


**Figure 4. Behavior of CCM/CCM*-based systems**

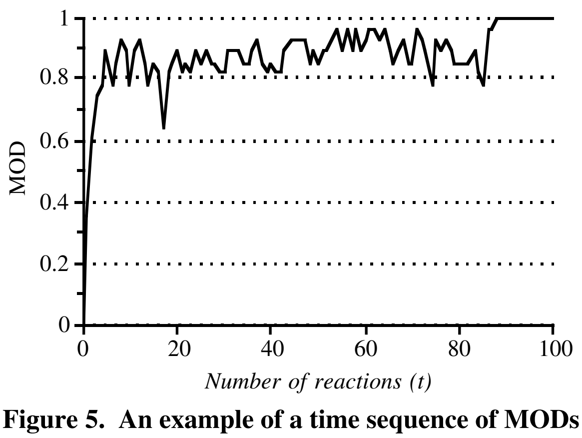


**Figure 5. An example of a time sequence of MODs**

A certain solution can probably be found even when the global optimum condition is not known, because the system works on local information. Thus,

---

[1] The composed rule is not reused. Thus, this is not a learning mechanism.

[2] Kanada [6] 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.

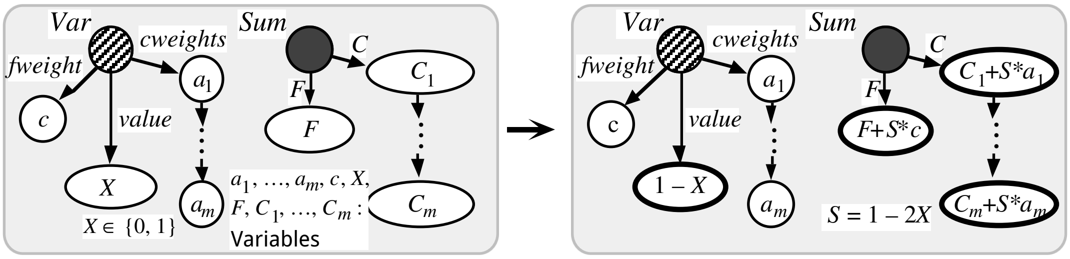


**Figure 6. The reaction rule for 0–1 IP problems**

open or dynamically changing problems may be solved using CCM. However, it is not concluded that all the constraints can always be satisfied, a global optimum solution can always be found, or the system always terminates at such a "goal" state by this method, because the system may be trapped by a local maximum and there is no assurance that the system terminates in finite time.

The principle described in the present section can be applied to wide range of problems, including CSPs and COPs. The 0–1 integer programming problem is used for example, because currently this is most successful as CCM*-based problem solving.

### 3.2 Optimization using CCM

Integer programming (IP) problems [13] are constrained linear programming (LP) problems in which values of variables are limited to integers. IP problems are NP-hard. Thus, the scale of solvable problems is limited, although good heuristics for branch-and-bound methods [13] are known.

The following type of IP problems, in which the variable values are restricted to 0 or 1, is called the 0–1 IP problem:

Maximize objective function $F = \sum_{j=1}^{n} c_j x_j$ ,

subject to $\sum_{j=1}^{n} a_{ij} x_j \le b_i$ ( $i = 1, 2, \ldots, m$), where $x_j$ ( $j = 1, 2, \ldots, n$) are variables, $x_j \in \{0, 1\}$, and $a_{ij}$ , $b_i$ , and $c_j$ are constants.

This problem can be solved as follows. One of the $n$ variables is selected randomly, and its value is switched to the other value if and only if this switching makes the objective function value ($F$) better. This process is repeated until no more switching will occur.

The content of the WM, the reaction rule, and the LOD are explained below. The WM contains an atom, *Sum* (of type sum), which contains the objective function value: $F$, a list of the values of the LHSs of the constraints: $C$, and a list of the values of the RHS of them: $B$. The $i$th element of $C$ (or $B$) is written as $C_i$ (or $B_i$). A simple reaction rule shown in **Figure 6**, which is the only rule, is used. This rule selects an atom *Var* of type var, and changes its value $X$ to $1 - X$, where $X \in \{0, 1\}$. Then the rule updates the values of the objective function and the LHSs of conditions in *Sum*.

An LOD is defined only for *Sum*, or for objects of type sum.

$Os(Sum) =$
$Sum.F$ **if** $Sum.C_i \le Sum.B_i$ **for all** $i \in \{1, 2, \ldots, m\}$
$-\infty$ **otherwise**

An LOD is not defined for variables: the LOD for objects of type var is always zero.

When the above system is activated, a reaction occurs only when the objective function value, which is equal to the total of the LODs, increases by the reaction. Thus, although this system works on local information, i.e., refers only a few objects, in each reaction, this is a cooperative system and realizes a simple hill-climbing method.[1] As a result, the system easily falls into a local maximum. It is even difficult to find a nearly optimum solution. This problem may be solved by supplying a more elaborate rule. However, this reduces the advantage of CCM, that a problem can be solved by a combination of a simple rule and a simple LOD. Thus, CCM* is used.

### 3.3 Detailed method of CCM*

A detailed method of problem solving using CCM* is explained here. This method is not specific to IP problems, but not very general. Both the number of reaction rules and the number of different compositions are asserted to be one. Different compositions mean compositions that generate rules that have different functions. For example, the swapping rule described in Section 2.1 can be composed in two ways:

$A_1(S1), A_2(S2), A_3(S3) \rightarrow A_1(S2), A_2(S3), A_3(S1).$

$A_1(S1), A_2(S2), A_3(S3), A_4(S4) \rightarrow$
$A_1(S2), A_2(S1), A_3(S4), A_4(S3).$

The first rule rotates the internal states of three atoms, and the second rule swaps the states of two pairs of atoms. The functions of these rules are different. So, there are at least two different compositions, and the following procedure does not apply to this case.

The global control of CCM* is the same as CCM. So only the procedure of interpreting a reaction rule is explained.[2] In the following procedure, the initial state of the WM is called $S_0$, the state after applying the reaction rule to the WM in state $S_{j-1}$ is called $S_j$. Thus, the sequence of reactions, $a_1, a_2, \ldots, a_j$ , changes

[1] However, this computation does not necessarily move toward the steepest slope.

[2] This procedure is still a simplified version. In more detail, the state of the WM is updated to $S_i$ in step 2), and it is (locally) backtracked to $S_0$ when going to step 1) from step 3). In addition, this procedure must be modified if there are two or more reaction rules.

state $S_0$ to $S_j$ ( $j \geq 1$). $O_j$ is the sum of the LODs of objects in state $S_j$ .

The procedure is as follows:

1) **Initialization**: Compute the upper bound of the number of reactions, $M$, using a random number. Compute $O_0$. Set $i$ to 1.
2) **Test and reaction**: Compute $O_i$ . If $O_0 < O_i$ (or $O_0 \leq O_i$ ) holds, cause the sequence of reactions, $a_1$, $a_2$, …, $a_i$ , and return from this procedure.
3) **Preparation for next test**: Increment $i$ by 1. If $i \leq M$, go to step 2) and test the next state. If $i > M$, go to step 1) (restart this procedure from $S_0$).

The distribution of the random numbers used in Step 1) can be selected from a variety of distributions. The distribution used in the experiments in the next section is approximately the following exponential function:

$$P(M) = (1 - e^{-\lambda}) e^{-\lambda M} \qquad (\lambda > 0)$$

For example, if $\lambda = 0.7$, the probability P(0) is 0.5. This means that the probability that no composition occurs (the original rule is used) is 0.5. It is not yet known whether this distribution is really better. The performance is not very sensitive to the shape of distribution function in our experiments.

Known facts on the random numbers to define the upper bound of the number of reactions (the upper bound of the distribution) are as follows.

- **Search efficiency**: It is possible to use a fixed value instead of random numbers for the upper bounds, $M$, or to use an unbound number of reactions. However, experiments have shown that these modified methods took more time at least in 0–1 IP problems. The reason is probably that to search neighbor state in the solution search is probabilistically more efficient, i.e., can find better state, than to search distant states when the current value of the objective function is closer to the optimum value, because the search space is not very steep.
- **Possibility of getting an optimal solution**: There are cases in which the system can reach the optimum solution by a reaction rule composed $\mu$ times, but it cannot reach the optimum solution by a rule composed less than $\mu$ times. In such a case, it cannot reach the optimum solution if the number of reactions is limited to less than $\mu$. It is therefore better not to set an upper bound on the random numbers if the value of $\mu$ is not known. The upper bound can be set to $n$ in 0–1 IP problems because $\mu$ is known to be $n$ (the number of variables). However, a reasonable upper bound cannot be known in general.

# 4. Experimental Results of 0–1 Integer Programming Problems

The experimental results of 0–1 IP problems are shown in the present section, comparing with those of the original CCM and the annealed CCM. Some other results are shown by Kanada [9]. The value of $m$ is fixed to 10, $a_{ij}$ satisfies $0 \leq a_{ij} < 10^5$, and fifty problems are generated randomly for each value of $n$, 10, 20, 30 and 40. The value of $c_j$ is $2500n$ for all $j$. These restrictions are set so as to make experiment easier, but they are not essential. The probability that the optimum solution is found by one trial is called the *optimum probability* (OP).

Part of the results is shown in **Table 1**.[1] The OP, the ratio of the approximate and exact values of the solutions in average (CCM*-Average ratio) and in the worse case (CCM*-Worst ratio) when $\lambda = 0.7$ are shown. The CCM* has been compared with three other methods on the same 0-1 IP problem. The first method is the original CCM, and the second is the annealed CCM. Their performances are also shown in Table 1. This table shows that CCM* is much better (the OP is 2.3 to 81 times higher) than the other two.

**Figure 7** shows a more detailed result of solving each problem by CCM*. The horizontal coordinate of each point indicates the average CPU time of 100 runs (two runs per problem), the vertical coordinate indicates the OP, and the number below is the value of $\lambda$ (from 0.125 to 2). If the value of $\lambda$ is changed from 2 to 1, the performance is much improved. However, the OP is saturated when $\lambda = 0.125$. This performance might not be enough if $n$ is large (e.g., $n = 40$).

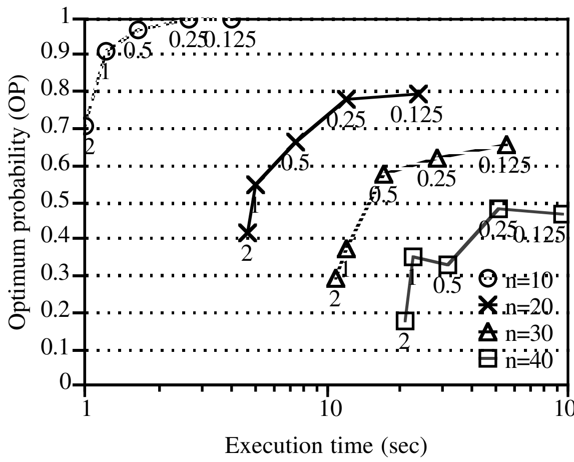


**Figure 7. The optimality and CPU time of solutions of 0–1 IP problems using CCM* ($\lambda$ is varying)**

A better performance is obtained by multiple trials. The result of solving each problem eight to sixteen times by CCM* is shown in **Figure 8**. The vertical coordinate of the left end of each polygonal line, which is labeled by "1," indicates the OP. The probability that the best solution in $k$ trials is optimum is called the *k-trial OP*. Two- to sixteen-trial OP and the CPU time needed for calculating the solutions are also shown in Figure 8. The eight-trial OP is more than 0.89.

CCM* has also been compared to a branch-and-bound (B&B) method by Kanada [9]. The result shows that CCM* found suboptimal solutions in less time if the multiple-trial OP is allowed to be 0.9 to 0.99. However, the B&B method finds optimum solutions, although solutions found by CCM* are not assured to be optimum even when the trials are repeated millions of times.

[1] The performance was measured using a SOOC (Self-Organization-Oriented Computing) language processor on a Macintosh Quadra 700. The optimum solutions for the comparison were found using a branch-and-bound method.

**Table 1. The optimality of solutions of the 0–1 IP problems using CCM***

| | CCM* ($\lambda$ = 0.7) | | | Original CCM | | | Annealed CCM | | |
|---|---|---|---|---|---|---|---|---|---|
| ***n*** | Optimum probability | Average ratio | Worst ratio | Optimum probability | Average ratio | Worst ratio | Optimum probability | Average ratio | Worst ratio |
| 10 | 0.97 | 0.998 | 0.83 | 0.012 | 0.62 | 0.14 | 0.42 | 0.95 | 0.77 |
| 20 | 0.71 | 0.995 | 0.91 | 0.000 | 0.63 | 0.19 | 0.006 | 0.80 | 0.37 |

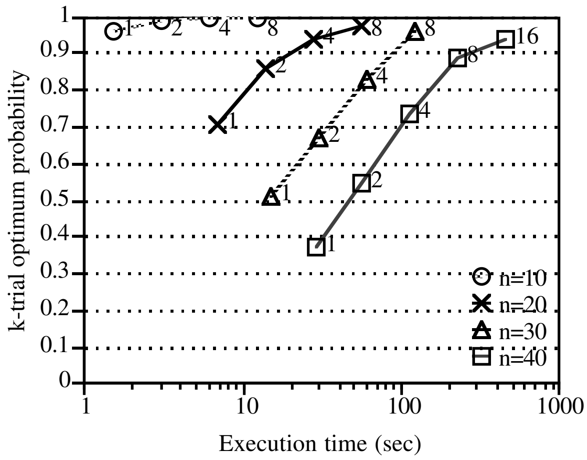


**Figure 8. The optimality and CPU time of solutions of 0–1 IP problems using CCM* ($\lambda$ = 0.7)**

## 5. Related Work

Kauffman et al. [11] investigated a method of problem solving or optimization of systems with many conflicting constraints by partitioning systems into small "patches," each of which behaves in a selfish way, i.e., without referring to other patches. The optimization is performed by the emergent collective behavior of the coevolving patches. This method performed better than conventional global optimization methods when there are many conflicting constraints.

The randomized dynamics of the Kauffman system, which are called Glauber or modified Glauber dynamics, are very close to CCM*. Patches correspond to the atoms concerning to a reaction caused by a composed rule in CCM*. However, they applied the method only to the *NK* model [10], in which data are arrayed in a lattice. The patches are fixed during the execution, and the efficiency of computation is out of scope in their paper. However, CCM is applied to more general data structures, i.e., graphs. The "patches" are dynamically determined during the execution in CCM.

Tunneling algorithms were proposed by Levy and Montalvo [12], Shima [14] and others. The tunneling algorithms are motivated by the concept of a tunnel effect, which is similar to skipping over a valley of MOD (Figure 2), and are used for optimizing continuous systems. The direction of search can be determined dynamically and randomly. CCM* can be regarded as a symbolic version of a randomized tunneling algorithm [14].[1]

[1] Thus, the relation between the randomized tunneling method and CCM* are similar to the relation between EP (evolutionary programming) and GA (genetic algorithms), because EP is numerical and GA is mostly symbolic.

## 6. Conclusion

Methods of solving combinatorial problems using extended CCM are explained in the present paper. CCM* performs much better than both the original and annealed CCM, a solution can be found using CCM* in less time than the latter, and good suboptimal solutions can be found in less time than B&B methods in randomly generated 0–1 IP problems. CCM* can easily be built-in to the reaction rule compiler, then suboptimal solutions can be found only writing a simple reaction rule and LOD.

In future work, the method of rule composition should be extended to be able to be applied to systems with two or more rules or different compositions. Other future work is mentioned by Kanada [9].

## References


1. Feige, U., Goldwasser, S., Lovász, L, Safra, S., and Szegedy, M.: Approximating Clique is Almost NP-complete, *32th Symposium on Foundations of Computer Science*, 2–12, 1991.
2. Forrest, S., ed.: *Emergent Computation*, MIT Press, 1991.
3. Hasida, K.: Dynamics of Symbol Systems, *New Generation Computing*, 12, 285–310, 1994.
4. Kanada, Y.: Toward Self-organization by Computers, *33rd Programming Symp.*, Information Processing Society of Japan, 1992 (in Japanese).
5. Kanada, Y.: Optimization using Production Rules and Local Evaluation Functions, *11th Mtg. of the Tech. Group on S. E.*, SICE, 1993 (in Japanese).
6. Kanada, Y., and Hirokawa, M.: Stochastic Problem Solving by Local Computation based on Self-organization Paradigm, *27th Hawaii Int'l Conf. on System Sciences*, 1994.
7. Kanada, Y.: Fuzzy Constraint Satisfaction Using CCM — A Local Information Based Computation Model, *FUZZ-IEEE/IFES '95*, 2319–2326, 1995.
8. Kanada, Y.: Large-scale Constraint Satisfaction Using Local-information-based Annealing and Its Parallel Processing, *SWoPP '95*, *Tech. Rep. of IEICE*, 1995 (in Japanese).
9. Kanada, Y.: Combinatorial Problem Solving Using Randomized Dynamic Tunneling on A Production System, *1995 IEEE Int'l Conferences on Systems, Man and Cybernetics*, 1995.
10. Kauffman, S. A.: *The Origin of Order*, Oxford University Press, 1993.
11. Kauffman, S. A., Macready, W. G., and Dickinson, E.: Divide to Coordinate: Coevolutionary Problem Solving, *ftp://ftp.santafe.edu/pub/Users/-wgm/patches.ps.Z.*
12. Levy, A. V., and Montalvo, A.: The Tunneling Algorithm for the Global Minimization of Functions, *SIAM J. Sci. Stat. Comp.*, 6: 1, 15–29, 1985.
13. Salkin, H. M., and Mathur, K.: *Foundations of Integer Programming*, North-Holland, 1989.
14. Shima, T.: Global Optimization by Annealing Type of Random Tunneling Algorithm, *TR of SICE*, 29: 11, 1342–1351, 1993 (in Japanese).
15. Takefuji, Y.: *Neural Network Parallel Processing*, Kluwer Academic Publishers, 1992.