

Stochastic Problem Solving by Local Computation based on Self-organization Paradigm*

Yasusi Kanada
Tsukuba Research Center
Real World Computing Partnership
Takezono 1-6-1, Tsukuba, Ibaraki 305, Japan
E-mail: kanada@trc.rwcp.or.jp

Masao Hirokawa
Advanced Research Laboratory, Hitachi Ltd.
Hatoyama, Saitama 350-03, Japan
E-mail: hirokawa@arl.hitachi.co.jp

Abstract

We are developing a new problem-solving methodology based on a self-organization paradigm. To realize our future goal of self-organizing computational systems, we have to study computation based on local information and its emergent behavior, which are considered essential in self-organizing systems. This paper presents a stochastic (or nondeterministic) problem solving method using local operations and local evaluation functions. Several constraint satisfaction problems are solved and approximate solutions of several optimization problem are found by this method in polynomial order time in average.

Major features of this method are as follows. Problems can be solved using one or a few simple production rules and evaluation functions, both of which work locally, i.e., on a small number of objects. Local maxima of the sum of evaluation function values can sometimes be avoided. Limit cycles of execution can also be avoided. There are two methods for changing the locality of rules. The efficiency of searches and the possibility of falling into local maxima can be controlled by changing the locality.

1. Introduction

We are developing a new problem-solving methodology based on a *self-organization paradigm* [Kan 92a, Kan 92b]. The long-term target of this methodology is to develop adaptive *real-world* or *open computational systems* that communicate with human beings, human-developed systems, and natural systems. Real-world systems, such as on-line banking systems, should be ready to process unexpected situations because human beings or natural systems are *autonomous* and *nondeterministic* and thus their behavior is often unpredictable. Complete specifications of these computational systems cannot be written because of such unpredictability.

Current software development methods are mostly top-down and are based on a type of divide-and-concur method. Expert systems are developed in a more flexible way, but are essentially the same. They depend on the illusion that the system is *reductionistic* and can be divided into "independent" functional modules. Current methods assert that the complete specification can be described. However, it is impossible to describe when the systems include or are interfaced to autonomous and nondeterministic systems. The reductionistic programming paradigm has failed to develop real-world systems.

Thus, we must find another paradigm. A most promising solution is the *self-organization paradigm*. The self-organization paradigm holds that computational systems are constructed without a whole and complete plan of computation and that they work basically in a bottom-up manner using local information only but generating global results via *emergent* behavior [For 91]. Thus, they work autonomously and nondeterministically.

However, extensive research is required to establish a methodology based on self-organization paradigm. We are only beginning research on this topic. Our current major research target is to establish a bottom-up computation mechanism and methodology based on local information. This paper presents a computation model called the chemical casting model (CCM) for problem solving using randomized applications of *local* operations and *local* evaluation functions, gives an example, and analyzes them. A major feature of this problem-solving method is that problems can be solved using one or a few simple production rules and evaluation functions, both of which work locally, i.e., on a small number of objects. Constraint satisfaction problems, such as graph coloring or the *N* queens problem, are solved, or approximate solutions of optimization problems, such as traveling salesperson problems, are found by this method.

To clarify the meaning of local operations and local evaluation functions, a general framework of problem solving is briefly introduced. Problem solving, such as optimization or constraint satisfaction, can be regarded as a state-space search. The initial state represents the problem

* Part of this research was done at Central Research Laboratory, Hitachi Ltd.

and the final state of a solution.[1] A problem can be solved by moving the current state from the initial to final state by applying *operators* in an appropriate order. The operators may be local or global. A local operator works on a small number of elements in the current state and moves it to a similar state in the search space. A global operator, such as a crossover in genetic algorithms, works on all the elements in the current state and moves it to a quite different state. A search in CCM is a (not completely) randomized walk in a state-space using local operators.

Evaluation functions are used in several problem solving methods. They are not used in blind search methods, such as depth-first or random searches. Global evaluation functions are used in hill-climbing methods and genetic algorithms. "Global" means that the value of the evaluation depends on all the elements in the current state. Search methods that do not use evaluation functions are usually inefficient. However, it is not easy for humans to define *global* evaluation functions when the problem is complex or multi-purposed. The current state often falls into a local maximum by methods that use local operators and global evaluation functions, such as hill-climbing methods. Local evaluation functions bias the randomized walk in CCM.

The basic paradigm of self-organization, which is the philosophical basis of this work, is explained in Section 2. CCM is explained in Section 3. An example based on CCM, the *N* queens system, is given in Section 4. The characteristics of CCM-based systems are analyzed in Section 5. Related works are mentioned in Section 6. Finally, we summarize our conclusions.

2. The Basic Paradigm of Self-organization

Recently, self-organization has been drawing attention in many research areas ranging from natural sciences to humanities. Several scientific research areas are concerned with self-organizing systems, such as dissipative structure theory [Pri 77], synergetics [Hak 78, Hak 83], the theory of molecule evolution [Eig 79], autopoiesis theory [Mat 80], bio-holonics [Shi 88] and natural and artificial neural networks. Jantsch [Jan 80] discusses a wide area of research on self-organization from a unified philosophical view, and indicates its direction for the future. These theories give us many suggestions for self-organization of computational systems. However, the main objects of these theories are natural systems, and there is large unsought research area between these systems and current computational systems.

Laszlo [Las 72] itemize four properties of natural systems. We believe artificial self-organizing systems should also have these properties. Thus, they are explained below from our point of view.

(1) *Holisticity*: Natural systems are holistic and have irreducible properties, such as mentioned in the previous section. This property may be undesirable for artificial systems, but is inevitable.

(2) *Self-stability*: Natural systems preserve themselves in changing environments. This property is also important for artificial systems. Self-stable systems work well even when there is a little noise (some errors) in the system or in the input. Although this property is important in natural systems or automatic control theory, it is usually ignored in computational systems or computation theories.[2]

(3) *Self-organizability*: "Natural systems create themselves in response to the challenge of the environment." No further explanation is given here because self-organization has been mentioned earlier.

(4) *Hierarchy*: Both natural and artificial systems are hierarchical. However, the hierarchical structure of natural systems is not as simple as tree structures that are common in artificial systems, nor they are static. Such a complex and dynamic (partial) hierarchical structure is called *heterarchy* [McC 65, Foe 84].

Although holisticity and self-stability are not mentioned explicitly in the rest of this paper, they are important as backgrounds of this work.

A generic macroscopic model of self-organizing systems [Kan 92a], which CCM is based on, is explained here. The model is shown in **Figure 1**. This model can be applied to a wide range of self-organizing systems, but not all of them. It models our target self-organizing computational system, a thermodynamic system that generates a dissipative structure, and other varieties of self-organizing systems.

Self-organizing systems generate order in a macroscopic level. The degree of order should be measurable because if it is not, we cannot claim objectively that the system is self-organizing. Thus, entropy or (global) *order degree*, which is a measure of the order or degree of organization, should be defined. If entropy is defined, it decreases with time. If order degree is defined, it increases with time. If the system is a thermodynamic system, entropy must be "thrown away" from the system, because it must satisfy the second law of thermodynamics which states that entropy increases in a closed system. Thus, it

[1] There may be no final state if the problem solving is an infinite process, such as that in human society. Even the state-space may be impossible to be predefined in such cases.

[2] Artificial systems based on system theories, such as automatic controlling systems with negative feedback, are self-stable. However, computational systems are not self-stable. Computational systems may crash due to a very small bug or unexpected input, which are types of *noise*.

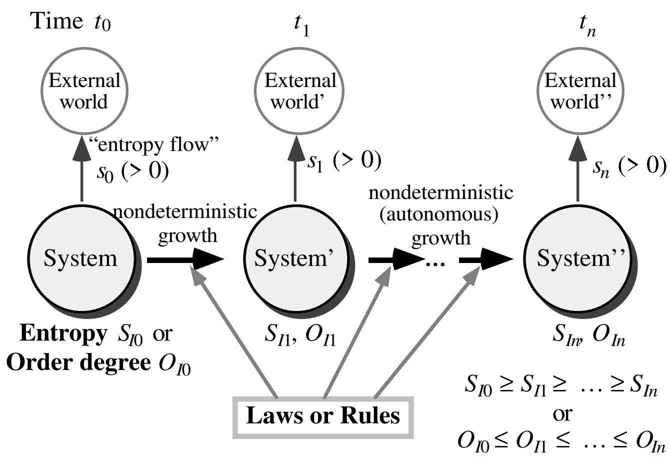


Figure 1. Generic model of self-organizing systems

must be an open system in this case.[1]

There must be a set of laws or rules that (partially) control how the system changes, though they probably do not control the system in a deterministic way for reasons described later. Probably, the rules work at a microscopic level and their emergent behavior generates complex macroscopic behavior of the system. Thus, the system is hierarchical (or heterarchical). For simplicity in this paper, rules, such as natural laws, are asserted as given and unchanging.[2] There will be various ways of describing the laws or rules. The laws can be written as differential equations in some systems, and may be written as sequences of operations in some others, and so on.

The growth of a self-organizing system is autonomous, and, thus, its behavior is unpredictable, or it is observed as nondeterministic or driven by noise that comes from the outside. However, these properties are not sufficient conditions for self-organization, of course. If the behavior is predictable, i.e., observed as deterministic, it is not a *self*-organizing system, but the organization is fully controlled by external laws or rules. In particular, in the case of computational systems, deterministic systems are (indirectly) organized by humans, because the rules, i.e., the programs, are written by humans. This does not constitute self-organizing computation. In thermodynamic or other physical systems, nondeterministic system development is called *bifurcation* or *symmetry breaking* [Pri 77, Pri 84].

3. Computation Model CCM

A computation model called the chemical casting model (CCM) is described in this section. This model defines the microscopic behavior of computation. Its name is derived from an analogy to chemical systems.[3]

3.1 Working Memory and Data

The system components in CCM are shown in **Figure 2**. We assert that the systems we are going to treat are discrete both in space and time. The reason why they are discrete in space is that the model to be defined is a model of self-organizing *symbols*; symbols that humans handle are discrete, as has been pointed out in linguistics and semiotics since Saussure [Sau 49].

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 an internal state and may be connected to other atoms by *links*. Links are similar to chemical bindings, but the difference is that links (may) have directions and may have labels (link names). A set of atoms connected by links can be called a *molecule*.[4] Any discrete data structure such as a list, tree, graph, or network can be represented using atoms and links.

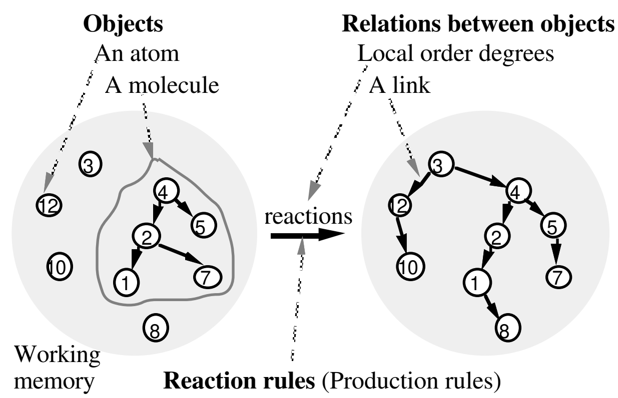


Figure 2. The elements of CCM

3.2 Reaction Rules

Reaction rules change the state of the working memory. Thus, they define the operators. The reaction rules are written as production rules, such as chemical reaction formulae or as rules in production systems. Production rules are used for the following reasons:

(1) The production system is a suitable model for bottom-up computation.

What we need is a model that uses bottom-up or emergent computation. Computation in forward-chaining production systems, such as the one in OPS5 [For 81],

[1] However, systems with no such law, such as computational systems, may be closed systems.

[2] "Self-organization" does not mean growth of rules in this simplified model. However, growth of rules is considered to be important in open systems and thus it should be included in our future theory.

[3] The word "program" is not used in this model because a program is a whole and complete plan. We use "cast" and "caster" instead.

[4] Although molecules possess a hierarchical structure, which is important in a complex system, molecules currently play no important role in CCM, so we make no further mention of them in this paper.

works in a bottom-up manner. Thus, a refined production system would be a good model.

(2) The analogy to chemical reaction systems is useful.

We need strong analogies for self-organizing computational systems, because we are just starting research on them. The theory of self-organization [Pri 77] was developed in certain chemical reaction systems, so the analogy would give us many suggestions.

(3) A production system is suitable for writing incomplete programs.

A production system is a good way of describing an incomplete system. In the case of a procedural language, incomplete programs usually violate the syntax or semantics and, thus, are nonsense. On the contrary, certain incomplete programs can be interpreted in a meaningful way in a production system.

The syntax of reaction rules is similar to rules in normal production systems. The reaction rules are also similar to reaction formulae in chemical reactions. The syntax of reaction rules is as follows:

LHS → RHS.

The left-hand side (LHS) and the right-hand side (RHS) are sequences of patterns. Each pattern matches an atom. An atom is used for matching only once in a rule application. This means that no atom matches two or more patterns on the left-hand side of a rule at once. Not only single atoms but atoms in a molecule can be matched here. The rule can be applied when there is a set of atoms that matches the LHS patterns. If the rule is applied, the matched atoms vanish and new atoms that match the RHS patterns are generated. For example, **Figure 3** shows a toy rule that removes oxygen and hydrogen molecules and makes water molecules.

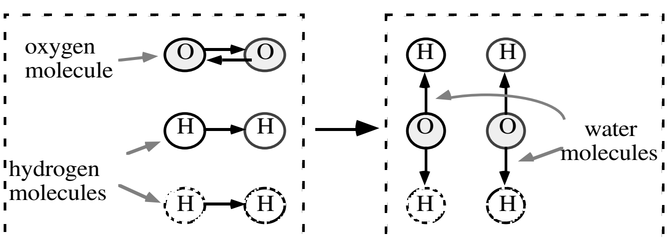


Figure 3. An example of a reaction rule

A pair consisting of a reaction rule and a set of atoms that can match the patterns in the rule is called an *instance*. An instance is said to be *reacted* if its rule is applied with its set of atoms. The instance is an implementation-*independent* concept, and is different from both the concept of an instance in conventional production systems[1] and that in object-oriented systems,

A rule may be reversible. LHS and RHS can be exchanged in a reversible rule. A reversible rule is written using a bi-directional arrow: LHS ↔ RHS.

3.3 Local Order Degrees

Although we did not mention this above, another important condition must hold to react an instance. This is called the *instance order condition*. This condition is computed using *local order degrees* (LODs), an evaluation function.[2] The existence of LODs is the most characteristic feature of CCM among production-system-based models of computation. The value of an LOD is usually an integer or a real number. LODs are defined in one of the following two forms.

(1) *Self order degree* $o(e)$: defined for an atom *e*.

(2) *Mutual order degree* $o(e1, e2)$: defined for a pair of atoms $\langle e1, e2 \rangle$.

The sum of the LODs of the atoms matching at least one of the patterns in the rule is called the *instance order degree* (IOD). The instance is reacted only when the IOD is not decreased by the reaction.[3] If the LOD is defined as a self order degree, the IOD is defined by the sum of the LODs of the matched atoms. The IOD before the reaction is computed and the IOD after the activation is estimated, then they are compared. The instance is activated only if the latter is larger. If the LOD is defined as a mutual order degree, the IOD is defined by the sum of the LODs of all the pairs of matched atoms. The comparison method is the same as for the self order degree.

In CCM, instances are reacted successively when possible. If there is no instance whose IOD is increased by the reaction, the system temporarily terminates. However, the system begins to work again when data in the working memory is modified or data is added and can react an instance.

3.4 Scheduling Strategies

The behavior of the system might not be determined uniquely by the instance order condition. This means that more than one instance can be reacted at the same time. The order of reaction is not predetermined in CCM. They may be reacted in any order, or they may be reacted in parallel, if they do not rewrite the same atom. Different orders of computation may cause different results. However, both results will be as expected, regardless of the order of computation.[4] However, if two instances contain the same atom, they may not be reacted in parallel. Thus, if a set of

[1] Instances are not elements of a conflict set. The system interpreter for CCM does not generate conflict sets.

[2] LODs may be analogical to chemical binding energy of atoms.

[3] This condition, $IOD_{before} \le IOD_{after}$, may be replaced by the following condition: $IOD_{before} < IOD_{after}$.

[4] LODs must be defined to satisfy this condition.

rules is well-defined, an ordered state indirectly induced by the LODs will be organized nondeterministically and in a *self-organizing manner*.

The system interpreter selects the instance whose instance order condition is to be tested and which may be reacted. Although instances are selected autonomously, the user or software can control the selection macroscopically by specifying a strategy. Strategies for the selection are called *scheduling strategies* in CCM, because instances can be regarded as microscopic processes. Scheduling strategies are related to conflict resolution strategies in conventional production systems. The strategies can be classified as sequential strategies and parallel strategies. Four types of sequential strategies are proposed by Kanada [Kan 92a]. The following two are particularly important.

(1) *Mathematical random strategies (MR strategies)*: Scheduling strategies that select instances using pseudo-random numbers or some other mathematical method of generating randomness (or noise) are called mathematical random strategies.

(2) *Systematic strategies (S strategies)*: Scheduling strategies that select instances in systematic methods are called systematic strategies. Although they do not select instances randomly, they do not refer to the logic of the system, i.e., selection is made autonomously. Thus the system can still be regarded as nondeterministic.

S strategies are less computation intensive. However, they may cause limit cycles (loops). MR strategies do not cause limit cycles, even if the user pays no attention to them. This is a major merit, and thus MR strategies are regarded as the standard strategies in CCM. Parallel strategies will make CCM-based computation appears more like chemical reactions. However, these are beyond the scope of this paper.

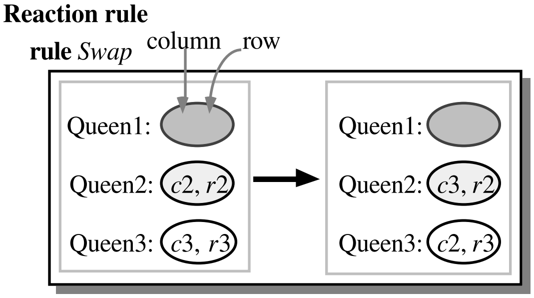
4. Example: The *N* Queens Problem

The *N* queens system, an CCM-based system for finding a solution to the *N* queens problem, is shown in this section. The performance of its trial executions is also shown. The *N* queens problem is an extension of the eight queens problem. The *N* queens system is tried because, although the target of our methodology is to develop complex systems, we have to start with a simple system, and this system has several characteristics that will probably lead us to a better understanding of complex systems.

4.1 Reaction rule and local order degree

Figure 4 shows the visual rule and LOD of the *N* queens system. This system contains only one rule and a definition of the mutual order degree, $o(x, y)$, for queens. This rule swaps the rows of two queens (Queen2 and Queen3 in Figure 4), see **Figure 5**.[1] Queen1, which can be called a *catalyst*, remains unchanged by the swapping. The role of the catalyst is explained later. No link is used in this rule. The value of LOD $o(x, y)$ is defined to be higher (i.e., 1) when queens x and y are not diagonally oriented, and lower (i.e., 0) when they are diagonally oriented.[2]

■ **Reaction rule**



■ **Local order degree (mutual order degree)**

The local order degree is defined between two queens.

$$o(x, y) = \begin{cases} 0 & \textbf{if } x.column - y.column = x.row - y.row \textbf{ or } \\ & \quad x.column - y.column = y.row - x.row, \\ 1 & \textbf{otherwise}. \end{cases}$$

Figure 4. A rule and LOD of the *N* queens problem

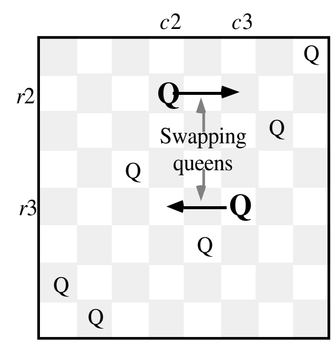


Figure 5. The meaning of the *N* queens rule

If the rule is executed with an appropriate initial state, the system repeats the selection of three queens and reaction of the instance. The initial state must satisfy the following condition: there is only one queen in each row and each column. The easiest layouts of queens that satisfy this condition are for all queens to be put on a diagonal line. If this condition holds, it holds at any time in the system because the reaction preserves the condition. The system stops when a solution to the *N* queens problem is found.[3]

---

[1] The LHS and RHS of this rule can be reversed and this rule can also be a reversible rule, because the LHS and RHS are symmetric, i.e., they are identical if variables $r2$ and $r3$ are exchanged.

[2] Shimizu and Hayashi [Shi 90], and Sosic [Sos 91] independently developed methods of solving the *N* queens problem by swapping columns. However, a *global* evaluation function that is equal to the GOD, explained in §5.1, is used in these methods.

[3] It can be proved that this caster (program) is *correct,* that means,

A more detailed semantics of a reaction is illustrated in **Figure 6**. The instance selections and reaction are repeated while possible. Because there is only one rule, the system interpreter does not have to select a rule. In this rule, the mutual order degree between the queens to be swapped, i.e., Queen2 and Queen3, is not changed. Thus, the system interpreter decides whether to react the instance or not, referring to the other two LODs displayed in the figure. The instance is reacted in this case, because its instance order condition holds. This shows the reason why the catalyst, Queen1, is necessary. The LODs are not changed if the rule contains only the queens to be swapped, and so the system does not work.

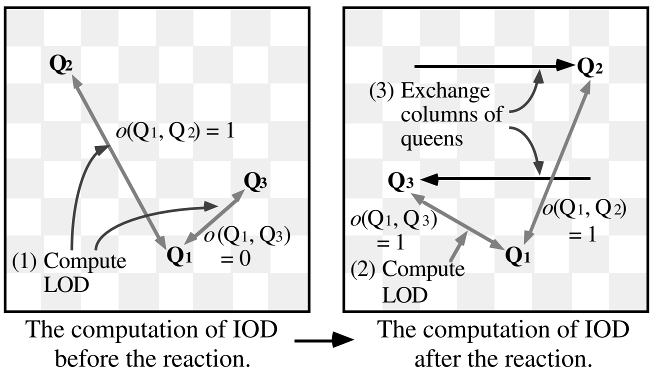


Figure 6. Computation of the IOD in the *N* queens rule

4.2 Performance

We have developed a computation language, SOOC, and its interpreter.[1] The rules and LODs of several problems have been developed and executed using SOOC. The performance of the *N* queens system is summarized below.

Although a solution is not always found in general by this method, because the search is stochastic and not exhaustive, our experiment shows that the problems never fail to be solved, and are solved in polynomial order time in average. The performance is fairly good with no extra device, probably because the problem is easy to solve.

The average number of matches, i.e., the number of executions of LHS of rules, and the average number of reactions until a solution is found have been measured for $N \leq 50$ using an S strategy. The initial layouts of queens are random (generated using pseudo-random numbers), and runs that fall into limit cycles are ignored. The results are shown in **Figure 7**. All values shown in this figure are averages of ten executions. The execution time is approximately proportional to the number of matches. Thus, the order of execution time is estimated to be $O(N^{4.6})$. The performance using an MR strategy is not shown here, but the trend is almost the same as the S strategy.

If backtracking is used to solve this problem, the order of execution time is exponential, so the CCM-based method is much faster. However, this simple CCM-based method is slower than other more intelligent methods, such as those shown in Yagrom and others [Yag 64, Cha 74, Shi 90, Sos 91], which find a solution in $O(N)$.

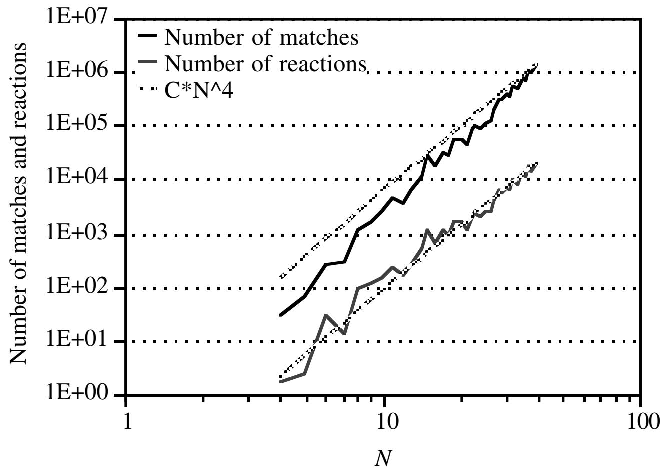


Figure 7. Average computation time of the *N* queens system (using an S strategy)

4.3 Other Examples

Several optimization and constraint satisfaction problems have been solved by CCM-based computation. They include traveling salesperson problems (TSP), integer programming problems, graph coloring problems [Kan 92b], and sorting. The results are summarized in **Table 1**. This table shows that simple problems can be solved using only one or a few simple rules and only one LOD by CCM.

5. Characteristics of CCM-based Computation

This section explains the characteristics of CCM-based computation after defining the global order degree.

5.1 Global Order Degree and its Stochastic Process

The *global order degree* (GOD) of an CCM-based system is the sum of the LODs.[2, 3] If the LOD is defined as the self order degree, the GOD is the sum of the LODs of all atoms. If the LOD is defined as a mutual order degree, the

a solution is obtained every time the system terminates. However, it is much easier to write a caster that tests the correctness of the result.

[1] We do not call SOOC (Self-Organization-Oriented Computing) a programming language because it is not a language to describe a complete procedure or a complete specification for computation. SOOC is a textual language. The current SOOC (SOOC-92) compiler and interpreter are written in Common Lisp.

[2] GOD is not computed nor used in CCM, but it may be used in macroscopic models.

[3] GOD may be analogical to total energy or entropy of chemical systems, and may be analogical to energy of Hopfield networks.

Table 1. Current Applications of CCM

Classification		Problem	Rules and LODs: Number of rules*	Rules and LODs: Number of LODs	Performance: Time	Performance: Solution quality
NP-hard	Optimization	TSP	1	1	$O(N^3)$	97 times optimum out of 100 trials (N = 10)
		0–1 Knapsack	1 (or 2)	1	$O(N^2)$	45 times optimum out of 100 trials (N = 20)
	Constraint satisfaction	N Queens	1	1	$O(N^{4.6})$	–
		Graph (or map) coloring	1	1	–	–
P-hard		Sorting	1	1	$O(N^2)$	–

* Rules for working memory initialization are not counted.

GOD is the sum of the LODs of all pairs of atoms in working memory. In the case of the N queens system, there are ${}_N C_2$ (= $N(N-1)/2$) pairs of queens. Because each LOD is 0 or 1, the minimum and maximum values of the GOD are 0 and ${}_N C_2$. The GOD is at a maximum at the solutions and *only* in these states, because the GOD is equal to the number of satisfied unit constraints in this system.[1] The GOD is at a minimum when all the queens are on a diagonal.

A GOD is a function of time. It may change when an instance is reacted. The time can be measured by the number of reactions since the system began to work. An example of the GOD transitions in the eight queens system, measured using SOOC with an MR strategy, is shown by the unshaded line in **Figure 8**. The solution was found in 88 reactions in this trial. The rule used in this measurement computes the GOD explicitly. The shaded line in this figure is explained in Section 5.3.

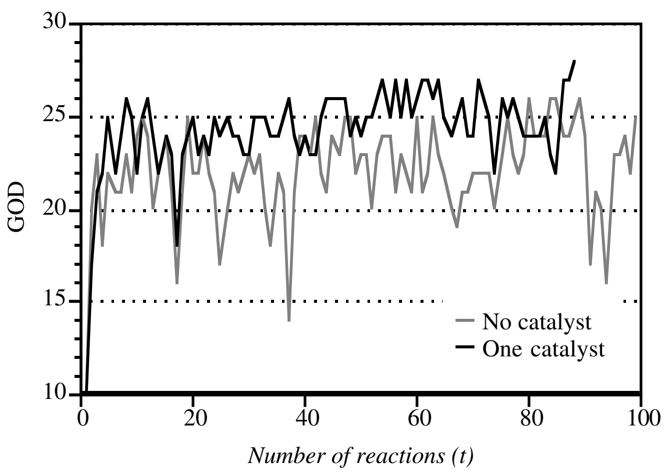


Figure 8. An example of GOD transition in the eight queens computation

[1] CCM-based systems maximize the GOD when their target is to solve constraint satisfaction or optimization problems. However, they may work in other way if their target is different.

Computation can be regarded as a stochastic process in CCM even when an S strategy is used. Figure 8 illustrates the meaning of this statement. The following three states occur in this order in the computation process of CCM, if appropriate rules and LODs are given.

(1) *Strongly non-stationary state*: The state in which the probability distribution rapidly changes when a reaction occurs.

(2) *Quasi-stationary state*: The state that the probability of the solution state, $p(g_{max})$, increases when a reaction occurs, where g_{max} is the maximum value of the GOD (= ${}_N C_2$), but that the ratio of other states, $p(g)/(1 - p(g_{max}))$ ($g = g_{min}, \ldots, g_{max}-1$), are almost constant when a reaction occurs, where g_{min} is the minimum value of the GOD (= 0).

(3) *Termination state* (*Stationary state*): The state that $p(g_{max})$ is 1. This is the limit state when $t \to \infty$.

In Figure 8, the quasi-stationary state is estimated to begin when t (number of reactions) is around 10. The above states can be modeled by a Markov chain [Kan 92b].

5.2 Conflict and Cooperation in CCM

CCM-based systems can be put into two categories.

(1) *Cooperative system*; where no reaction will decrease the GOD.

(2) *Conflicting system*; where a reaction may decrease the GOD.

Cooperative systems are called such because reactions cooperate toward the local or global maximum of the GOD.

The N queens system and the graph coloring system mentioned in Section 4.3 are conflicting systems, unless the IOD is the same as the GOD. Thus the GOD does not increase monotonically as shown in Figure 8. Some systems have little conflict while others have considerably more. However, the definition of the degree of conflict is a task for the future. On the contrary, the GOD monotonically increases in a cooperative system. Examples of cooperative systems are the TSP system, the 0–1 Knapsack system and the sorting systems mentioned in Section 4.3.

5.3 Variability of Locality

The locality of computation can be controlled by adding catalysts to rules or composing two or more rules. Rules with less or more locality work differently because the value of IOD is used at reaction. This characteristic of CCM-based systems is called the variability of locality. The effects of controlling the locality in cooperative and conflicting systems are explained below.

In a cooperative system, the local maxima of the GOD can partially be avoided by reducing the locality by composing rules. This property is explained in the next section.

In a conflicting system, to add catalysts to a rule makes the system less conflictive and decreases the average number of reactions required to find a solution. If more catalysts are added, the rule refers to a more global state. For example, the rule in the *N* queens system contains a catalyst. If this catalyst is removed, the system does not stop even when a solution is found. However, this system performs a random search. An example of GOD transition in a random search is shown by the shaded line in Figure 8. On the contrary, more catalysts may be added to the rule. However, if more catalysts are added, the execution time for the rule matching increases, thus the efficiency of the system is reduced.

Figures 9 and **10** shows the effect of adding or removing catalysts in the *N* queens system. The average and standard deviation of GOD in the quasi-stationary state as functions of the number of catalysts, *Nc*, are shown in Figure 9. The average becomes higher and the standard deviation becomes lower when *Nc* increases. This means that catalysts bias the search: a system with more catalysts searches among the states where the GOD is higher, thus the computation is more efficient. The performance as a function of *Nc*, measured on a Macintosh IIX computer, is shown in Figure 10. The data are not given for $Nc = 0$ because the execution does not stop in this case. Although the number of reactions decreases when *Nc* increases, the execution time, which is approximately proportional to the number of matches, increases when $Nc > 2$.

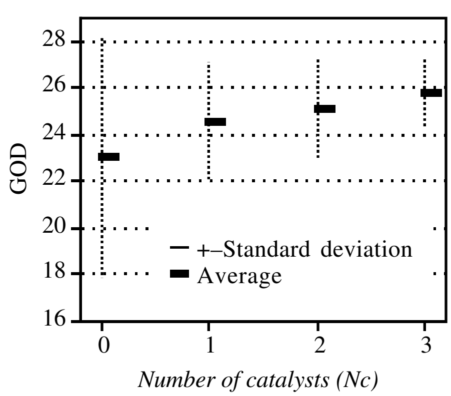


Figure 9. Relation between number of catalysts and global order

Adding catalysts decreases the possibility of escaping from local maxima. This property is explained in the next subsection. Adding catalysts also makes it impossible to solve the problem for small *N* in the *N* queens problem, because the rule cannot produce a solution if the number of matching patterns in the LHS is greater than *N*. Thus, there is an optimum number of catalysts.

The rule with catalysts can sometimes be generated not only by adding catalysts to the rule, but also by composing a rule with no catalyst, and thus rule composition can be used for controlling rule locality. **Figure 11** shows the composition of a rule with a catalyst using a rule without a catalyst in the *N* queens system. An application of the rule with a catalyst, shown in Figure 4, moves the current state from State 0 to 3. The three contiguous applications of a rule without a catalyst also move the current state from State 0 to 3. In the first step, the columns of Q1 and Q2 are swapped. The columns of Q2 and Q3, and the columns of Q1 and Q2 are swapped in the following steps. Rules with two or more catalysts can be composed in the same way.

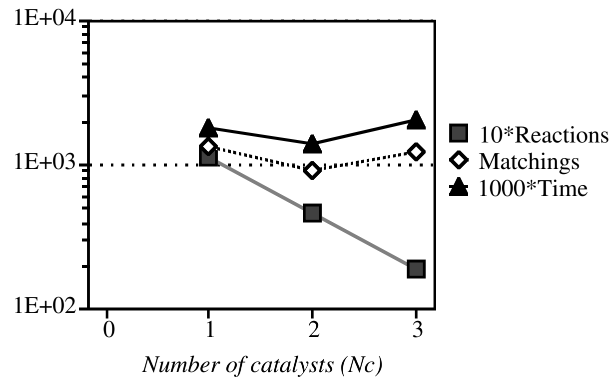


Figure 10. Relation between number of catalysts and performance

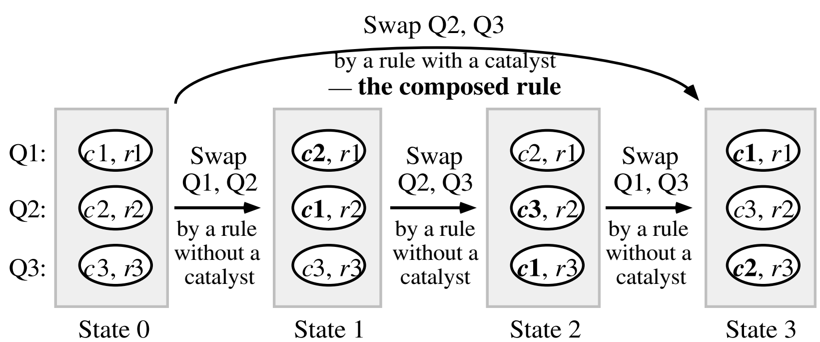


Figure 11. Composition of a rule with a catalyst using a rule without a catalyst

5.4 Escaping from Local Maxima

Conflicting and cooperative systems behave differently in regards to local maxima. Thus, the behavior of conflicting systems is explained first, and then that of cooperative systems is explained.

A conflicting system, such as the *N* queens system but not all conflicting systems, never falls into the local maxima of the GOD and can reach the global maxima, i.e., reach solutions, even when using an S strategy. This behavior can be regarded as emergent.

Here, a local maxima of GOD means that a state S such that GOD(S) ≥ GOD(S') holds for any state S' derived from S by a reaction. An example of a local maximum in the six queens system is shown in **Figure 12**.[1] The maximum value of the GOD is 15, and that of the state shown is 14.

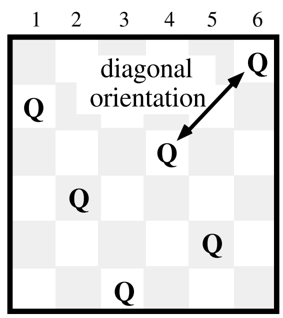


Figure 12. An example of a local maximum in the six queens system

If a hill-climbing method is used for finding the maximum value of the GOD, the search may fall into a local maximum. However, the CCM-based system can escape from local maxima because there is always an instance that can be reacted even when the system is in a local maximum.[2] For example, in Figure 12, if the queens in the sixth, fifth and fourth columns are matched to Queen1, 2 and 3 of the rule in Figure 4 respectively, the reaction occurs and the system escapes from this local maximum.

Reducing the locality by adding catalysts decreases the possibility of escaping from local maxima. For example, the state shown in Figure 12 is a local maximum if a rule with four catalysts, i.e., a rule with six patterns, is used, because the IOD computed in this rule is equal to the GOD. On the contrary, in cooperative systems, reducing the locality by adding (but not replacing) rules that are compositions of the original rule usually increases the possibility of escaping from local maxima. This happens, for example, in optimization problems such as the TSP system or the 0–1 Knapsack system.

6. Related Work

Katai et al. [Kat 93] proposes a method of constraint satisfaction using autonomous decentralized systems. This research is motivated by synergetics. A complicated hierarchical mechanism is used in this method. Our research targets a simpler mechanism.

CCM, genetic algorithms (GA), neural networks such as Hopfield's [Hop 85] and constraint programming all target algorithm-less computation. CCM and GA are similar in that both perform stochastic computation, and that data are converted and tested by evaluation functions. However, they differ in that the evaluation functions in GA use global information, but those in CCM use local information only. GA is applicable only when a specification can be written. Hopfield networks also work with global evaluation functions called energy functions, though they are not computed explicitly. In constraint programming, symbolic constraints are usually satisfied by constraint propagation. The N queens system in CCM suggests an alternative method of constraint satisfaction, which is much easier to program and probably more robust.

CCM-based systems that use a parallel scheduling strategy have similarity to the chemical abstract machine [Ber 90] which is also based on a production system. However, their target is not self-organization but the building of a semantic model for parallel computation, and it does not use evaluation functions. A set of rules and LODs in CCM can also be regarded as a description of a probabilistic algorithm [Bra 88]. However, our target is quite different from that of conventional probabilistic algorithms. A probabilistic algorithm must have correctness, which is meaningful only when there is a specification of the algorithm. On the contrary, our target is computation without complete specification.

Other approaches related to the self-organization of computational systems include self-organizing neural networks, artificial life, and connectics [Tak 91].

7. Concluding Remarks

The CCM-based systems shown in this paper can be interpreted as self-organizing systems in a sense. However, they are still far from our goal of *real* self-organizing system, which is even hard to be defined. This paper contributes to this grand challenge in the following points.

(1) The self-organization paradigm is explained, and the relation between this paradigm and computation from local information to global result via emergent behavior are explained.

(2) A computation model, CCM, which will lead us to a better understanding of self-organizing systems, is proposed, and a "programming" style using local operations and local evaluation functions is shown.

(3) An emergent property that a solution can be found without falling into local maxima is reported using the N queens system, for example.

(4) Methods of controlling the locality of computation by adding catalysts to rules or composing rules in CCM are proposed, and their effects are explained.

[1] Sosic [Sos 91] states that there is no local maxima when $N \geq 1000$.

[2] A conflicting system may have cooperative local maxima, where no reaction decreases the GOD. Thus, the condition that the system is a conflicting system is not a sufficient condition for this characteristic.

The main focuses of future work are as follows. First, the *N* queens problem and other problems mentioned in this paper are very simple and basically closed problems, that means we can write specifications for these problems. We have to develop CCM-based open systems for problems that are not just constraint satisfaction nor optimization problems, and to observe and to analyze more complex emergent properties in those systems. Second, rules and order degrees are invariant in the model shown in this paper. However, not only the rules but also the goal or target of computation, which is related to the value of evaluation functions, may change during computation in real self-organizing systems. Thus, CCM should be extended to express self-organization or *reflection* of rules or LODs.

Acknowledgments

The authors wish to thank the following persons. Mr. C. Bando of Hitachi Research Laboratory gave the first author a chance to study the theories of self-organizing systems and to begin this research. The people who joined in the discussion at the 33rd programming symposium provided us with many useful ideas. Dr. R. Oka of Real World Computing Partnership gives the first author a chance to progress this research. One of the referees of this paper gave the authors very detailed and stimulus comments.

References

[Bra 88] Brassard, G., and Bratley, P.: *Algorithmics — Theory and Practice*, Prentice-Hall, 1988.

[Ber 90] Berry, G., and Boudol, G.: The Chemical Abstract Machine, *17th Annual ACM Symposium on Principles of Programming Languages*, pp. 81–94, 1990.

[Cha 74] Chandra, A. K.: Independent Permutations, as Related to a Problem of Moser and a Theorem of Polya, *J. of Combinatorial Theory (A)*, Vol. 16, pp. 111–120, 1974.

[Eig 79] Eigen, M., and Schuster, P.: *The Hypercycle: A Principle of Natural Self-organization*, Springer-Verlag, 1979.

[Foe 84] von Foerster, H., in *Self-Organization and Management of Social Systems*, Springer Series in Synergetics, Springer-Verlag GmbH & Co. KG, 1984.

[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, Stephanie, ed.: *Emergent Computation*, MIT Press, 1991.

[Hak 78] Haken, H.: *Synergetics — An Introduction*, Springer-Verlag GmbH & Co. KG, 1978.

[Hak 83] Haken, H.: *Advanced Synergetics: Instability Hierarchies of Self-organizing Systems and Devices*, Springer-Verlag GmbH & Co. KG, 1983.

[Hop 85] Hopfield, J. J., and Tan, D. W.: Neural Computation of Decisions in Optimization Problems, *Biological Cybernetics*, Vol. 52, pp. 141–152, 1985.

[Jan 80] Jantsch, E.: *The Self-organizing Universe: Scientific and Human Implications of the Emerging Paradigm of Evolution*, 1980.

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

[Kan 92b] Kanada, Y.: Software as Self-organizing Systems, *Summer Programming Symposium*, Information Processing Society of Japan, 1992 (in Japanese).

[Kat 93] Katai, O., Matsubara, S., Masuichi, H., Katayama, T., Sawaragi, T., Ida, M., and Iwai, S.: An Autonomous Decentralized System for Constraint-Oriented Problem Solving Involving Continuous and Fuzzy Varibles, *1st Int. Symp. on Autonomous and Decentralized Systems*, 1993.

[Las 72] Laszlo, E.: *The Systems View of the World — The Natural Philosophy of the New Developments in the Sciences*, George Braziller, 1972.

[Mat 80] Maturana, H. R., and Varela, F. J.: *Autopoiesis and Cognition: The Realization of the Living*, D. Reidel Publishing Co., Dordrecht, Holland, 1980.

[McC 65] McCulloch, W. S.: A Heterarchy of Values Determined by the Topology of Nervous Nets, in *Embodiments of Mind*, MIT Press, 1965.

[Pri 77] Nicolis, G., and Prigogine, I.: *Self-organization in Nonequilibrium Systems — From Dissipative Structures to Order through Fluctuations*, John Wiley & Sons, Inc., 1977.

[Pri 84] Prigogine, I., and Stengers, I.: *Order Out of Chaos*, Bantam Books, New York, 1984.

[Sau 49] de Saussure, F.: *Cors de Linguistique Generale*, Charles Bally and Albert Sechehaye, 1949.

[Shi 88] Shimizu H., et al.: *Survey Report on Bio-information Systems*, 63-A-298, Japan Electronic Industry Development Association, 1988 (in Japanese).

[Shi 90] Shimizu, H., and Hayashi, A.: A Fast Algorithm for the n-Queen Problem, *Technical Report of the IEICE* COMP90-39, 1990 (in Japanese).

[Sim 81] Simon, H. A.: *The Sciences of the Artificial*, Second Ed., MIT Press, 1981.

[Sos 91] Sosic, R., and Gu, J.: 3,000,000 Queens in Less Than One Minute, *ACM SIGART Bulletin*, Vol. 2, No. 2, pp. 22–24, 1991.

[Tak 91] Takeuchi, I., Goto, S., Onai, R., Saito, Y., and Okuno, H.: The Plan of Connectics, *JSSST 8th Annual Conference,* B3-1, 1991 (in Japanese).

[Yag 64] Yaglom, A. M., and Yaglom, I. M.: *Challenging Mathematical Problems with Elementary Solutions*, Holden-Day, San Francisco, 1964.