Self-organizing Formation Algorithm for Active Elements

Kenichi FUJIBAYASHI[1]*, Satoshi MURATA[1], Ken SUGAWARA[2,3] and Masayuki YAMAMURA[1]

[1]*Department of Computational Intelligence and Systems Science, Tokyo Institute of Technology, Yokohama, Kanagawa 226-8502, Japan*
[2]*PREST, JST*
[3]*The Graduate School of Information Systems, The University of Electro-Communications, Tyofu, Tokyo 182-8585, Japan*
**E-mail address: fuji@mrt.dis.titech.ac.jp*

(Received January 20, 2003; Accepted March 20, 2003)

Keywords: Self-organizing Formation, Virtual Spring, Triangular Lattice, Combination, Probability

Abstract. In this paper, we propose a novel method of self-organizing formation. It is assumed that elements are not connected to each other, and they can move in continuous space. The objective is to arrange elements in certain spatial pattern like a crystal, and to make the outline of the group in desired shape. For this purpose, we proposed a method by using virtual springs among the elements. In this algorithm, an element generates virtual springs between neighbor element based on information how many other elements exist in neighborhood with a certain radius. Although the elements interact locally only by virtual springs, and they do not have global information at all, they form a shape much larger than the sensory radius. By simulation study, we confirmed convergence to a target shape from a random state in very high probability. This kind of algorithm gives a new principle of self-organizing formation, and its simplicity will be useful for design of self-assembling nano machines in future.

1. Introduction

There are various phenomena in which many identical elements make the whole form by self-organization. For example, in the growth process of a crystal, many atoms or molecules form regular lattice structure according to the principle of free energy minimization. Although atoms are very small and simple, when they aggregate, they make a symmetrical crystal in a macroscopic scale. Form of a living organism is also a result of self-organization process. Beginning from fertilization, it makes a peculiar form by dividing cells repeatedly and arranging them in a certain order. Primitive elements of this process are biological cells containing equivalent genetic information. The cells determine the form, through exchanging chemical substances. Although the behavior of a cell is farther complicated than the atom in a crystal, there is a common process in which global order of the system emerges from the cooperation of homogeneous elements influencing locally.

However, in the world of the artificial things, such kind of self-organization seldom appears. It is because an artifact is usually designed along a certain purpose, and its form and structure are optimized to realize desired purposes. However, in the artificial system of the next generation, the self-organization processes will be applied more purposefully. For example, for a machine in a nano scale, it is not realistic to pick a part and assemble them one by one. Like a protein molecule, many elements must self-aggregate to form a functional component (self-assembly). Self-organization is also important to realize self-repairing artifacts. For instance, several modular robot systems capable of self-repair have been considered recently. They consist of many homogeneous mechanical modules, and it can restore itself by replacing a broken module by a new module (self-repair). In such a system, it does not know beforehand which module will break. Therefore, the self-organization algorithm in which the group of elements cooperates and forms a target shape is needed.

Formation processes of homogeneous elements can be classified roughly into the following three classes.

A. Elements are not connected to each other, and they can move in continuous space.

B. Elements are connected to each other, and they are constrained to certain discrete relative positions.

C. Elements are connected to each other, but continuous change in relative positions is allowed.

Class A is a system like a flock of birds or a school of fish, or a group of mobile robots as an artificial system. Class B is a system like a crystal or a snowflake, and its artificial example is a modular robot. Class C can be found only in natural systems such as protein aggregation or multi-cellular organisms.

Algorithms of self-organizing formation have been proposed for each class. For class A, Yamashita proposed an algorithm to align a group of homogeneous mobile robots in a circle (SUZUKI and YAMASHITA, 1999). An alignment algorithm based on nonlinear reaction-diffusion is also proposed (FUKUDA *et al.*, 2002). Methods to control not a group's form but the dynamic behavior of a group are also studied for class A (REYNOLDS, 1987; SUGAWARA *et al.*, 2000). For class B, a self-assembly algorithm for a modular robot is proposed based on local connection style among modules (MURATA *et al.*, 2001), and a self-repair algorithm using hierarchical description of the robot shape is proposed for the same modular robot (TOMITA *et al.*, 1999; FLOCCHINI *et al.*, 1999; WALTER *et al.*, 2000). About class C, although many models of developing organisms are proposed, it is thought that an artificial hardware corresponding to this class does not exist yet.

In this paper, we assume that elements are not connected to each other, and they can move in continuous space (class A). However, our target is like class B. Namely, the objective is to arrange elements in certain spatial pattern like a crystal, and to make the outline of the group in desired shape. For this purpose, as the simplest model which can consider the size of an element, we proposed a method by using virtual springs among the elements. In this algorithm, an element generates virtual springs between neighbor elements based on how many other elements exist in the neighborhood with a certain sensory radius. Although the elements interact locally by virtual springs, and they do not have global information at all, they form a shape much larger than the radius. By simulation study, we confirmed convergence to a target shape from a random state occurs in very high

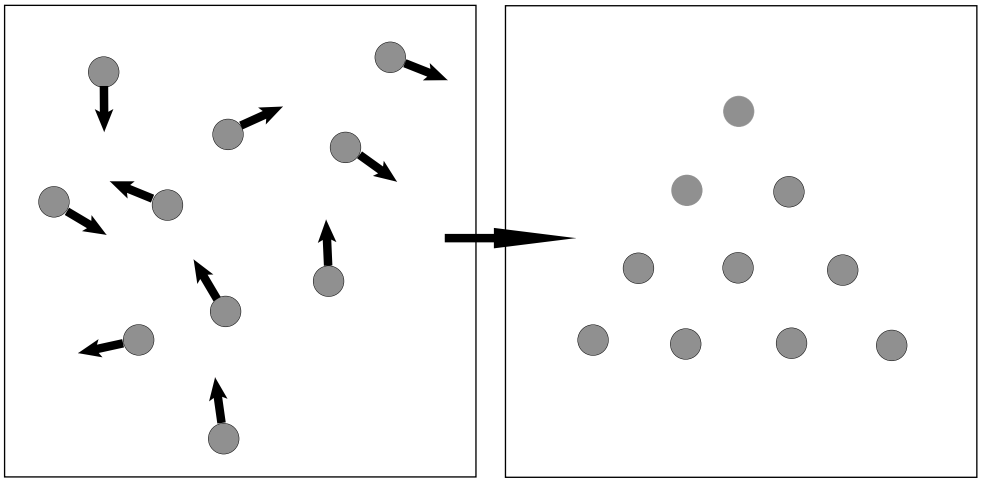

Fig. 1. Self-organizing formation.

probability. This kind of algorithm gives a new principle of self-organizing formation, and its simplicity will be useful for design of self-assembling nano-machines in future.

2. Self-organizing Formation

2.1. Problem formulation

The objective of formulation problem is to arrange elements in predetermined order without depending on their initial position and direction (Fig. 1). We assume properties of the elements as follows:

- All the elements have the same character.
- When a distance between two elements is less than a sensory radius *R*, they interact each other by a virtual spring between them.
- The number of elements which exist in *R* is countable.

2.2. Algorithm of self-organizing formation

We have developed an algorithm that utilizes the number of connection between elements under the problem setup above.

2.2.1 Connection number and virtual spring

In the algorithm, virtual springs shall be generated to all the elements that exist in *R*. We call the number of elements in the circle with a radius of *R*, number of connection. For example, elements *a*, *b*, *c*, *d*, *e* in Fig. 2 have connections 2, 4, 2, 3, 3, respectively.

Virtual springs used in the algorithm are assumed to have a large spring constant *k*. By setting natural length of virtual springs *l* slightly smaller than *R* (i.e. 90% of *R*), a triangular lattice is made among the elements (Fig. 2). This is regarded as “nominal” setting. We determined the value of *k* and *l* by trial and error.

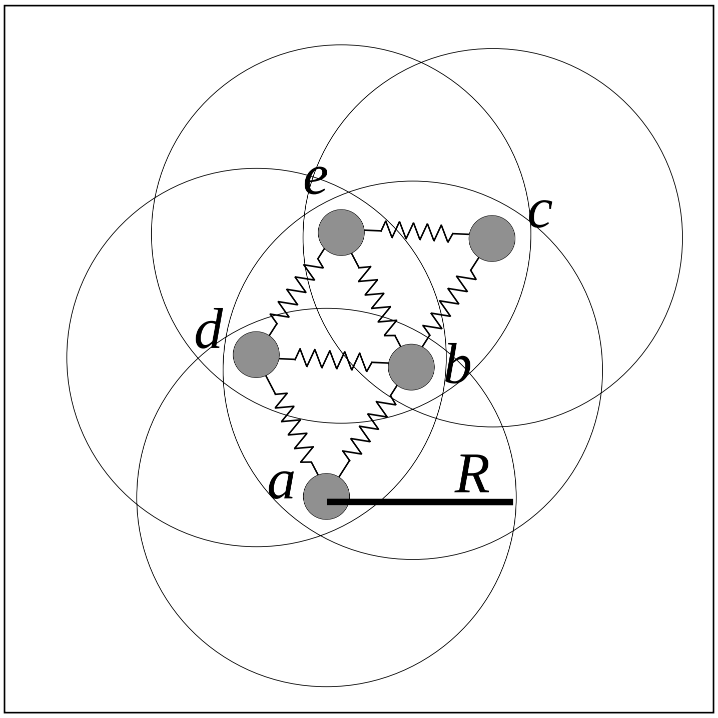

Fig. 2. The number of connection and triangular lattice.

2.2.2 Tuning virtual springs

In the formation problem, we need to control not only the internal structure pattern, but also the outline of the global structure. In order to do this, we changed characteristics of the virtual springs. Namely, spring constant and natural length of a spring is defined by the numbers of connection of the elements at the both ends of the springs.

For combinations of the number of connection exist in the target form, spring constant and natural length is set to their nominal values. In addition to them, we can separate elements with a certain combination of the number of connection by setting the natural length to a value bigger than R. Combination is canceled by separation, because springs vanish when the element pushed out of R. The product of the spring constant and the spring length can tune the strength of ejection.

We also put a probability P of existence to each virtual spring. By using the probability, we can control frequency of the separation.

For example, in the situation of Fig. 3(left), assume that we want to separate a virtual spring *a*-*c* or *b*-*c*, because the combination of the number of connection 2-5 is not included in the target form. To do this, we can set the virtual spring 2-5 with large spring constant, natural length and probability of existence compared with nominal springs. As a consequent, spring *a*-*c* or *b*-*c* will be replaced by a large spring, and it pushes the elements, and then

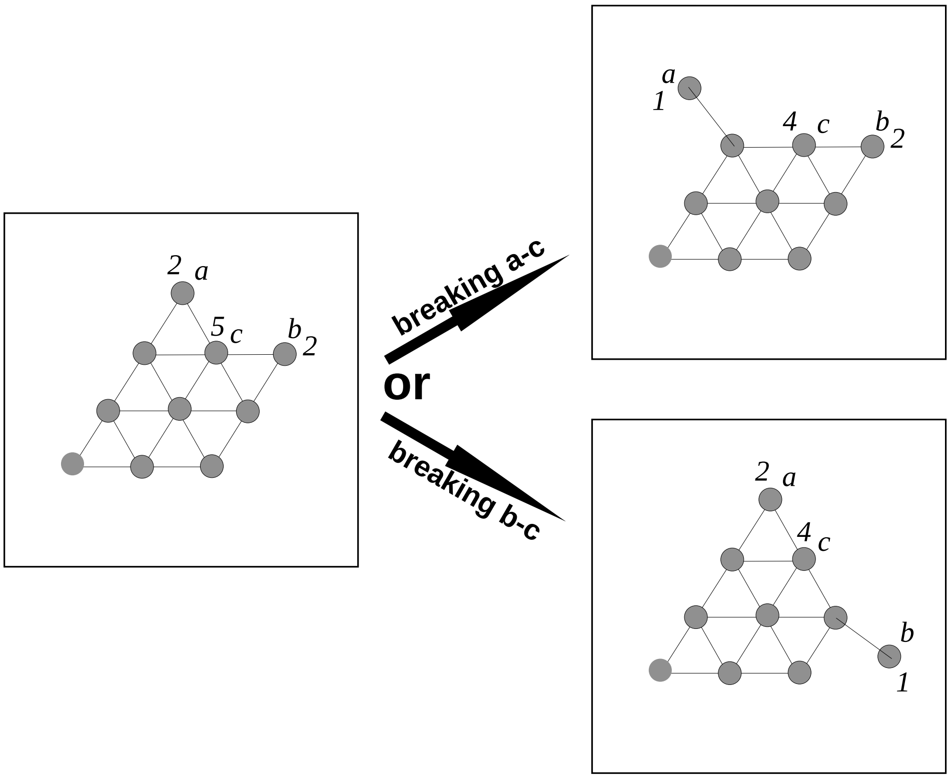

Fig. 3. Breaking unnecessary bonds.

vanishes because it goes out of range. By introducing suitable combination of springs, we can generate desired shapes.

3. Simulation

3.1. Simulation model

Elements move on a two-dimensional space. Subscript *i* is an identification number of the element.

From Fig. 4, equation of motion for the element *i* is

$$m\ddot{\mathbf{q}}_i + c\sum_{j=1}^{n_i}\left(\dot{\mathbf{q}}_i - \dot{\mathbf{q}}_j\right) + \sum_{j=1}^{n_i} k_{n_i - n_j}\left(1 - \frac{l_{n_i - n_j}}{\left|\mathbf{q}_i - \mathbf{q}_j\right|}\right)\left(\mathbf{q}_i - \mathbf{q}_j\right) + d\dot{\mathbf{q}}_i + e\mathbf{q}_i = 0 \qquad (1)$$

where, position of the element $\mathbf{q}_i$ is defined as a vector, n_i is the number of connection of

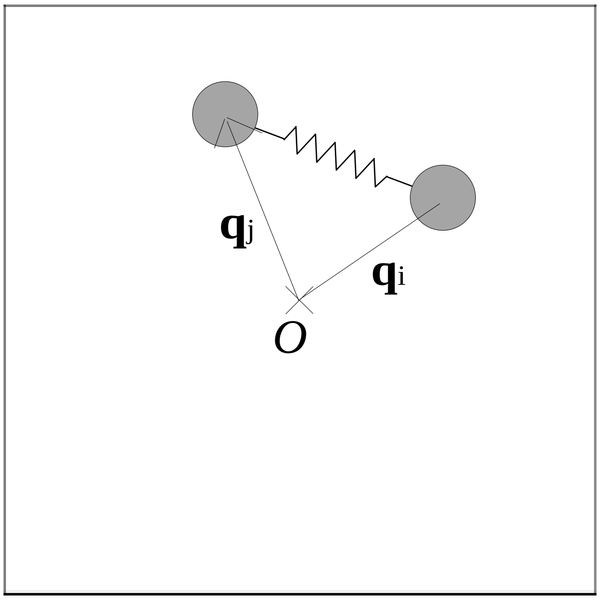

Fig. 4. Movement of element *i*.

elements *i*. Identification number of the element which exists in neighbor *R* of element *i* is denoted as *j*, and $\mathbf{q}_j$, n_j are position and the number of the connection of the element *j*, respectively. *m* is mass of the element, and *c*, $k_{n_i\text{-}n_j}$, $l_{n_i\text{-}n_j}$, *d* denote damper coefficient, spring coefficient, natural length of spring, and friction coefficient, respectively. We assume the friction force $d\dot{\mathbf{q}}_i$ and the additional force $e\mathbf{q}_i$ to aggregate the elements to the origin. These are the force proportional to the distance between the element and the origin of the plane. These are added to increase the efficiency of convergence, but they are weak enough so that the formation is not affected.

In this model, elements are basically "passive" without virtual springs, but become "active" according to the force of virtual springs. This equation of motion is integrated by Euler method. Differential Eq. (1) is discretized and update is performed on all the elements in every step. Outline of the whole procedure is as follows.

1. For all the element *i*,
   (a) calculate distance between elements, and select the element *j* in *R*.
   (b) For all the element *j* in *R*,
      i. by using n_i, n_j, determine $k_{n_i\text{-}n_j}$, $l_{n_i\text{-}n_j}$ and $P_{n_i\text{-}n_j}$.
      ii. generate random number *rand*(0‘1), and if $rand < P_{n_i\text{-}n_j}$, apply the force of virtual spring.
2. Calculate new position and velocity of the element *i* by Euler method.

3.2. Triangle formation

We simulated the formation process of a triangle. We changed a set of parameters such

Table 1. Virtual springs for triangle formation.

n_i - n_j	$k_{n_i\text{-}n_j}$	$l_{n_i\text{-}n_j}$	$P_{n_i\text{-}n_j}$
2-4	200	90	1
4-2	〃	〃	〃
4-4	〃	〃	〃
4-6	〃	〃	〃
6-4	〃	〃	〃
6-6	〃	〃	〃
2-5	250	120	0.7
3-4	300	120	0.005
3-3	300	120	0.001
Others	200	120	0.0001

as, spring constant, natural length, and generating probability according to combination of connection number. For each set of parameters, formation process is simulated 1000 times, and the rate of success and an average step number are evaluated. Here, throughout the simulation we assumed m = 10, r = 5 (radius of an element body), d = 2, c = 2, e = 0.3, R = 100, and $|\dot{\mathbf{q}}_{init}|$ = 2 (initial velocity), where $\dot{\mathbf{q}}_{init}$ has a random direction.

As a result, the best parameter set was selected for the number of elements n = 15 (see Table 1) and it was applied to various size of triangles (n = 6, 10, 21).

3.2.1 Parameter tuning for triangle formation

Combinations n_i-n_j contained in the target triangle are 2-4, 4-2, 4-4, 4-6, 6-4, and 6-6. Because these are structural members, we put nominal property ($k_{n_i\text{-}n_j}$ = 200, $l_{n_i\text{-}n_j}$ = 90) for them. $P_{n_i\text{-}n_j}$ (generation probability of the springs) is set to 1.

These springs are not sufficient to control the global outline shape of the system. We need to add other kind of springs to reshape outline to the target shape. We introduced springs with a larger spring constant and larger natural length to cut a specific unnecessary connection (lower half of Table 1).

Frequency of breaking the connections that are not included in a triangle is also important.

If the spring with the combinations such as n_i = 4, 5, and 6, is generated for separation in high probability, it will never converge. On the contrary, if the probability is too low, then other formations easily appear. We found that moderate generation probability for 3-3 and 3-4 is efficient to form a triangle.

Deadlock shapes usually contain obtuse vertices (120°), and the connection number of them is 3. We, therefore, cut connections including n_i = 3 in low probability. However, since n_i = 3 also appears during the desired formation process of a triangle, the probability must be small enough. Combination 2-5 has a particular function. It gives an element freedom of motion along the edge, instead of completely cutting it off. We set a very small probability to all the other springs not contained in the above list. This is effective to prevent deadlocks in general.

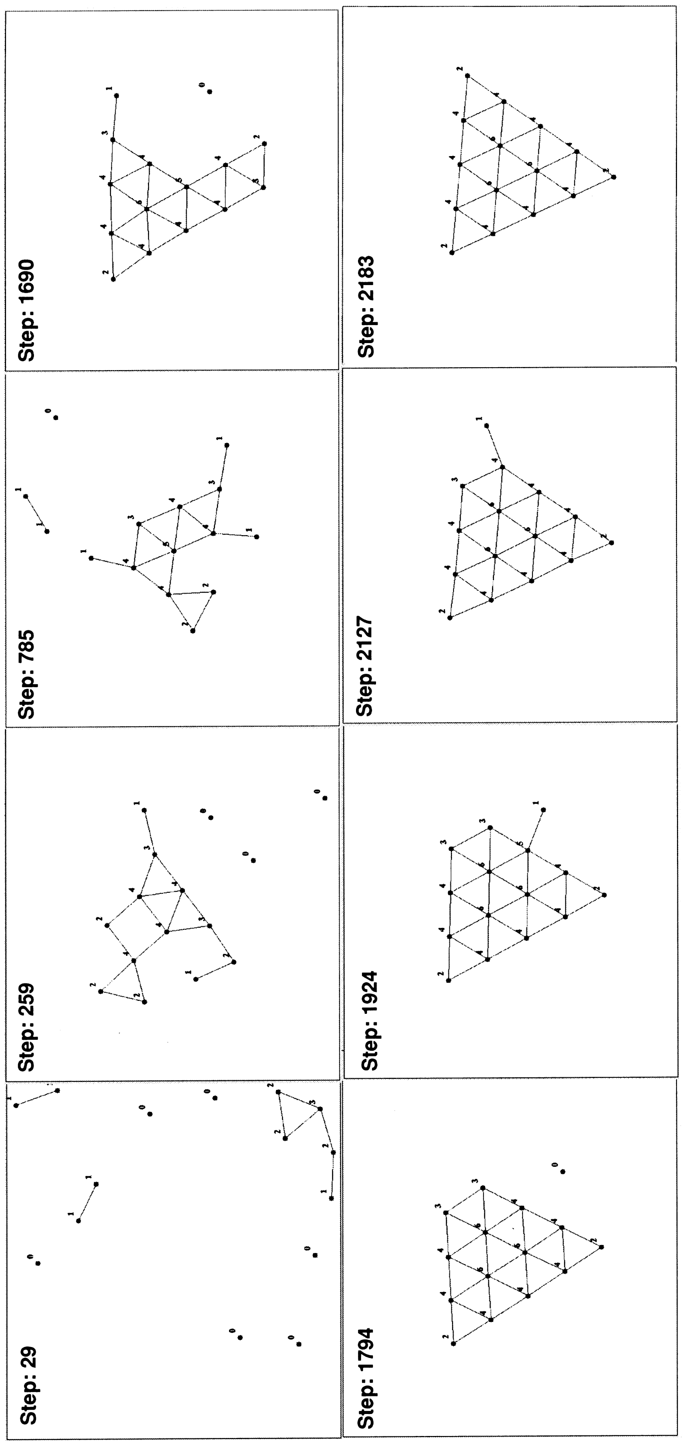

Fig. 5. Simulated formation process of a triangle.

Table 2. Simulation results by size of target.

Size of target triangle	6	10	15	21
Success rate (%)	100	100	100	83.7
Steps in best case	100	160	313	257
Steps in worst case	31684	55774	42364	178253
Average	4423	7447	7662	27508

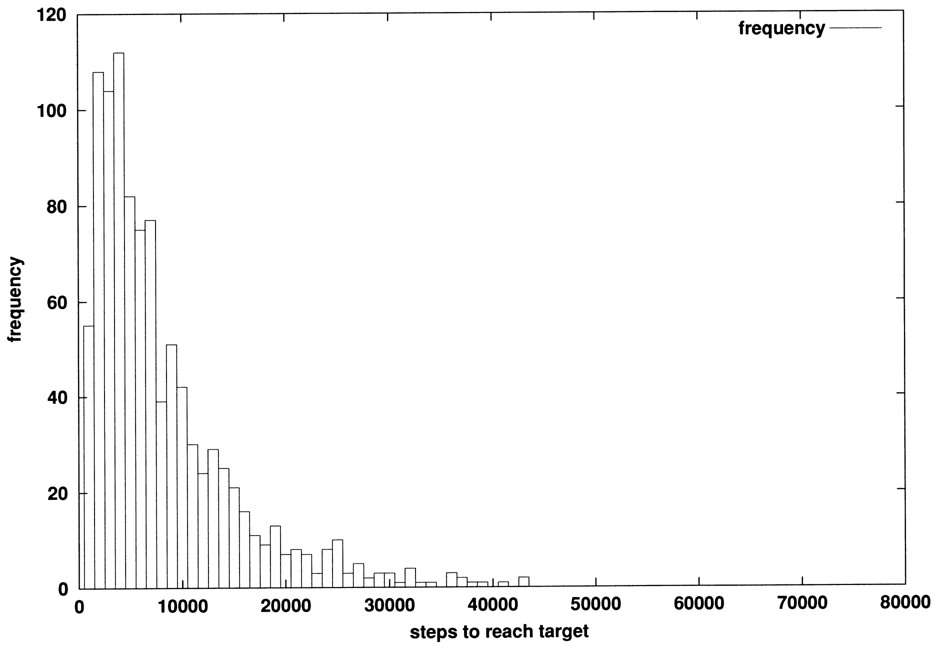

Fig. 6. Histogram of steps to reach target (n = 15).

3.2.2 Simulation results

Simulation results are shown in Table 2 and Fig. 5. The rate of convergence was evaluated by sizes of targets. We used the parameter set tuned for a triangle with 15 elements for all of them.

In Fig. 6, the peak of the necessary step for convergence appeared in the left of the plot shows that it is basically a random walk process. It is a process without memory, because the number of activation for each rule is linearly increased against convergence time (Fig. 7).

Since those simulations were obtained by the parameters for 15 elements, convergence for 6 and 10 elements were worse. It will be improved if parameters are tuned according to that target size. Although for the target larger than 15, we need longer convergence time, the rate of a success did not get worse so much, since the target shape is the only stable state.

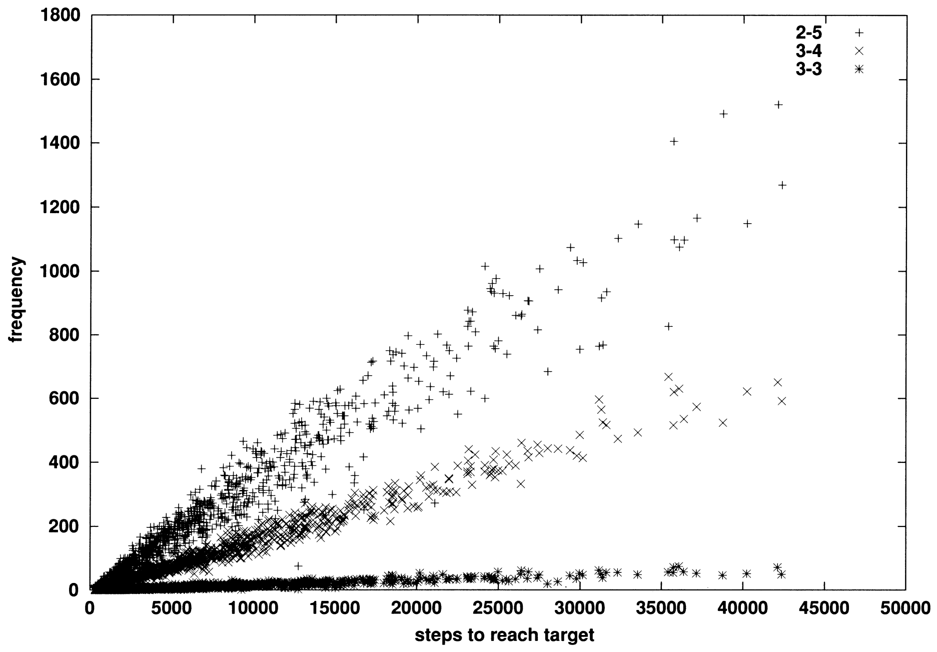

Fig. 7. Frequency of separation by spring 2-5, 3-4, 3-3 (n = 15).

3.3. *Formation of other shapes*

We applied the same algorithm to other shapes such as a linear ladder and a hexagon.

For a ladder, we assume $n = 15$, $m = 10$, $r = 5$, $d = 0$, $c = 2$, $e = 0$, $R = 100$, and for a hexagon $n = 19$, $m = 10$, $r = 5$, $d = 2$, $c = 2$, $e = 0.3$, $R = 100$.

Combinations in the ladder are 2-3, 3-2, 2-4, 4-2, 3-4, 4-3, 4-4, and combinations in the hexagon are 3-6, 6-3, 4-6, 6-4, 6-6. $k_{n_i\text{-}n_j}$, $l_{n_i\text{-}n_j}$ and $P_{n_i\text{-}n_j}$ were set to the same value as 3.2. We designed other springs as shown in Tables 3 and 4.

By preliminary simulation, we succeeded in forming these shapes as is shown in Figs. 8 and 9. However, the tuning of the parameters was difficult, and the convergence was much slower than the case of triangle. In order to improve the efficiency, we need to add some springs to remove deadlock states.

4. Conclusion

In this paper, under the constraint that the elements locally interact with neighbors within radius R, we proposed an algorithm to form a shape much larger than R by using virtual springs, where, the properties of the spring were designed based on the number of connection of the element.

In the proposed algorithm, it is necessary to tune parameters for the specific shape and the specific number of elements. In order to obtain the optimal parameter automatically, we

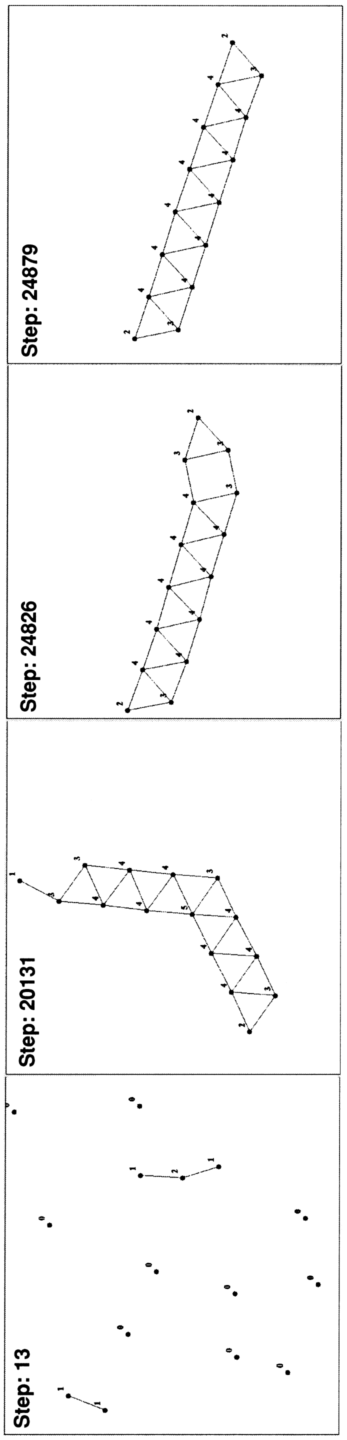

Fig. 8. Formation process of a ladder.

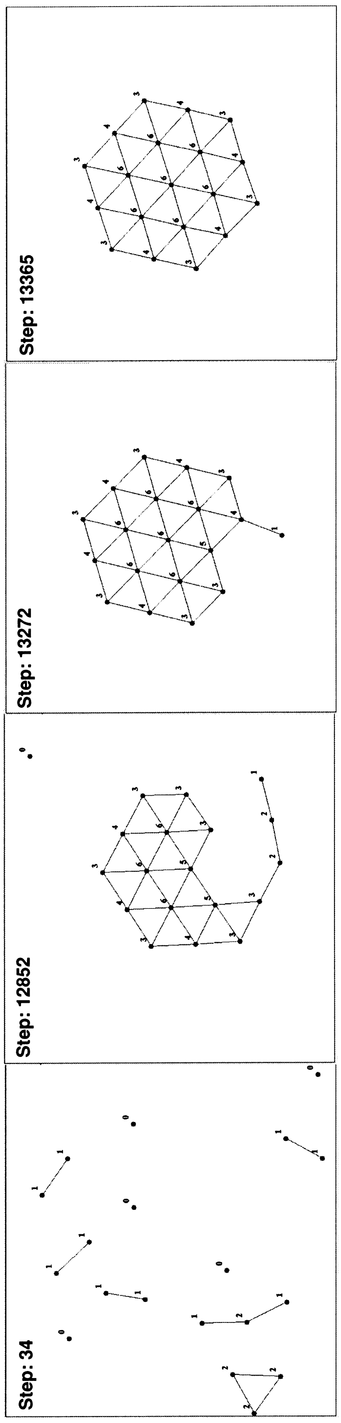

Fig. 9. Formation process of a hexagon.

Table 3. The generating virtual springs for a ladder.

n_i-n_j	$k_{n_i\text{-}n_j}$	$l_{n_i\text{-}n_j}$	$P_{n_i\text{-}n_j}$
2-3	200	90	1
3-2	″	″	″
2-4	″	″	″
4-2	″	″	″
3-4	″	″	″
4-3	″	″	″
4-4	″	″	″
2-5	250	120	0.7
Others	250	120	0.001

Table 4. The generating virtual springs for a hexagon.

n_i-n_j	$k_{n_i\text{-}n_j}$	$l_{n_i\text{-}n_j}$	$P_{n_i\text{-}n_j}$
3-4	200	90	1
4-3	″	″	″
3-6	″	″	″
6-3	″	″	″
4-6	″	″	″
6-4	″	″	″
6-6	″	″	″
2-4	250	120	0.7
2-5	250	120	0.7
3-3	300	120	0.0005
Others	300	120	0.0001

can use genetic algorithms for instance. However, since it is a random process that does not have memory, it is difficult to drastically improve convergence speed. To extend this algorithm we need to introduce some additional properties such as nonlinear characteristics of the virtual springs and internal state of the element. For instance, by introducing the state, it becomes a system that has memory. By using the memory, we can not only improve convergence but assemble hierarchical structures, and more complicated shapes will be formed by such an extension.

As a future work, we need to prove that various formations are possible by this algorithm that does not exist in natural crystals. We also plan to redesign the algorithm in simpler form aiming at hardware embodiment by a cluster of small mobile robots. The simplicity of the formation algorithm may also lead us to applications such as nano/ molecular machines such as self-assembling of DNA molecules (MAO *et al.*, 2000; TURBERFIELD *et al.*, 2000).

REFERENCES

FLOCCHINI, P., PRENCIPE, G., SANTORO, N. and WIDMAYER, P. (1999) *Hard Tasks for Weak Robots: The Role of Common Knowledge in Pattern Formation by Autonomous Mobile Robots*, ISAAC, 93–102, LNCS 1741, Springer.

FUKUDA, T., IKEMOTO, Y., ARAI, F. and HIGASHI, T. (2002) Graduated spatial pattern formation, in *Proc. DARS 2002*, pp. 340–349.

MAO, C., LABEAN, T. H., REIF, J. H. and SEEMAN, N. C. (2000) Logical computation using algorithmic self-assembly of DNA triple-crossover molecules, *Nature*, **407**, 28 September.

MURATA, S., YOSHIDA, E., KUROKAWA, H., TOMITA, K. and KOKAJI, S. (2001) Self-repairing mechanical system, *Autonomous Robots*, **10**, 7–21.

REYNOLDS, C. W. (1987) Flocks, herds, and schools: A distributed behavioral model, in *Computer Graphics*, **21**(4) (SIGGRAPH '87 Conference Proceedings), pp. 25–34.

SUGAWARA, K., SANO, M., YOSHIHARA, I., ABE, K. and WATANABE, T. (2000) Collective behavior of multi-robot system with simple interation, in *Proc. 5th Int. Symp. on Artificial Life and Robotics*, pp. 725–727.

SUZUKI, I. and YAMASHITA, M. (1999) A theory of distributed anonymous mobile robots—Formation and agreement problems, *SIAM J. Computing*, **28**, 4, 1347–1363.

TOMITA, K., MURATA, S., KUROKAWA, H., YOSHIDA, E. and KOKAJI, S. (1999) A self-assembly and self-repair method for a distributed KUROKAWAsystem, *IEEE Transactions on Robotics and Automation*, **15-6**, 1035–1045.

TURBERFIELD, A. J., YURKE, B. and MILLS, A. P., Jr. (2000) DNA Hybridization Catalysts and Molecular Tweezers, DNA Based Computers V: DIMACS Workshop DNA Based Computers, June 14–15, 1999, DIMACS Series Vol. 54, pp. 171–182.

WALTER, J., WELCH, J. and AMATO, N. (2000) Distributed reconfiguration of metamorphic root chains, in *Proc. of 19th ACM Annual Symp. on Principles of Distributed Computing (PODC 2000)*, pp. 171–180, Portland, Oregon.