Autonomous Robots 10, 91–106, 2001

Modular Robot Motion Planning Using Similarity Metrics

CHIH-JUNG CHIANG AND GREGORY S. CHIRIKJIAN
Department of Mechanical Engineering, Johns Hopkins University, Baltimore, Maryland 21218, USA
chiang@caesar.me.jhu.edu
gregc@jhu.edu

Abstract. In order for a modular self-reconfigurable robotic system to autonomously change from its current state to a desired one, it is critical to have a cost function (or metric) that reflects the effort required to reconfigure. A reconfiguration sequence can consist of single module motions, or the motion of a "branch" of modules. For single module motions, the minimization of metrics on the set of sets of module center locations serves as the driving force for reconfiguration. For branch motions, the question becomes which branches should be moved so as to minimize overall effort. Another way to view this is as a pattern matching problem in which the desired configuration is viewed as a void, and we seek branch motions that best fill the void. A precise definition of goodness of fit is therefore required. In this paper, we address the fundamental question of how closely geometric figures can be made to match under a given group of transformations (e.g., rigid-body motions), and what it means to bisect two shapes. We illustrate these ideas in the context of applications in modular robot motion planning.

Keywords: metric, group, optimal assignment, morphing, pattern matching, modular robots

1. Introduction

In this paper we review and apply metrics between shapes consisting of finite sets of points in Euclidean space. We review some well-known metrics in computational geometry and compare the applicability and computational requirements of each metric in the context of self-reconfiguring robots. All of the metrics considered provide means of judging how similar geometric objects are.

The particular application explored here is modular self-reconfigurable robot motion planning. A number of researchers have explored such robots, with Fukuda and Kawauchi (1990), Fukuda et al. (1991), Kokaji (1988), and Kelmar and Khosla (1988) having paved the way. Fukuda and Kawauchi (1990), and Fukuda et al. (1991) considered cellular robotic systems in which a heterogeneous collection of independent specialized modules are coordinated. Beni and Hackwood and coworkers addressed theorerical issues relating to cellular robotic systems (Beni, 1988; Hackwood and Wang, 1988). Yim (1993, 1994) considered modular robots composed of a few basic elements which can be composed into complex systems, and used for various modes of locomotion. Murata et al. (1994) considered a 'fractal' system composed of modules with zero kinematic mobility, but which can 'walk' over each other in discrete quanta due to changes in the polarity of magnetic fields. Chirikjian (1993, 1994) introduced the concept of "metamorphic" robots consisting of a homogeneous collection of modules, and with coworkers proved certain things about the computational complexity of the self-reconfiguration process (Chirikjian et al., 1996; Pamecha et al., 1997). Chen and Burdick (1993) provided a tool for defining equivalence classes of modular robot configuration with the same shape and morphological function. Hamlin and Sanderson (1997) considered reconfigurable truss structures. Pamecha et al. (1996) implemented metamorphic robotic systems composed of several two-dimensional module designs. Murata et al. (1998) considered a system of modules that can achieve

three-dimensional self-reconfiguration. Kotay et al. (1999a) and Kotay et al. (1999b) described a three-dimensional system composed of molecule robots that can be reduced to metamorphic robots. A number of recent works have addressed modular self-reconfigurable systems to locomote over steps (Hosokawa et al., 1999) and for use as deployable structures (Lipson and Pollack, 2000).

The remainder of this paper is structured as follows. In Section 2, basic definitions and examples which will be used throughout the paper are presented. In Section 3 applications in modular robot motion planning are discussed. In Section 4 we present a new algorithm for self-reconfiguration based on recursively bisecting robot configurations. In Section 5 we present numerical results.

2. General Definitions

In this section we provide general mathematical definitions which will be used throughout this paper. Section 2.1 reviews the concept of a group, and provides several examples of geometric importance. Section 2.2 reviews the definition of a metric and provides examples. Section 2.3 examines metrics on equivalence classes generated by the action of a group on a metric space.

2.1. *Defining Properties and Examples of Groups*

A *group* $(G, \circ)$ is a set, G, together with a binary operation, $\circ$[1], such that the following group axioms are satisfied (Chirikjian and Kyatkin, 2000):

1. $g_1 \circ (g_2 \circ g_3) = (g_1 \circ g_2) \circ g_3 \; \forall g_i \in G$.
2. There exists an element $e \in G$ such that $g \circ e = g \; \forall g \in G$.
3. There exists a $g^{-1} \in G$ such that $g^{-1} \circ g = e$ for each $g \in G$.

A group is said to act on a set, X, if for all $x \in X$ and all $g \in G$, $g \cdot x$ is defined in such a way that $g \cdot x \in X$, and $(g_1 \circ g_2) \cdot x = g_1 \cdot (g_2 \cdot x) \;\; \forall g, g_1, g_2 \in G$ and $e \cdot x = x$.

A very important group in the context of robotics is the group of rigid-body motions. This group, also called the special Euclidean group or Euclidean motion group, is denoted as $SE(N)$, consists of all pairs $g = (R, \mathbf{b})$, where R is an $N \times N$ rotation matrix and $\mathbf{b} \in \mathbb{R}^N$. It has the group law $g_1 \circ g_2 = (R_1 R_2, R_1 \mathbf{b}_2 + \mathbf{b}_1)$. This group acts on $\mathbb{R}^N$ as $\mathbf{x}' = R\mathbf{x} + \mathbf{b}$. In addition to being a group, $SE(N)$ can be viewed as an $N(N+1)/2$-dimensional manifold.

Another important group is the group of permutations of n letters, denoted Π_n. It is a finite group containing $n!$ elements. We denote elements of Π_n as

$$\pi = \begin{pmatrix} 1 & 2 & \cdots & n \\ \pi(1) & \pi(2) & \cdots & \pi(n) \end{pmatrix}.$$

We will use both of these groups in subsequent sections of this paper.

2.2. *Definitions and Examples of Metrics*

Given a set $\mathcal{S}$, a *metric* is a nonnegative real-valued function, $d : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$, which has the following properties (Munkres, 1975; Zikan, 1990):

$$\forall A, B \in \mathcal{S}, \quad d(A, B) \geq 0 \text{ and } d(A, B) = d(B, A)$$
$$d(A, B) = 0 \Longleftrightarrow A = B$$
$$\forall A, B, C \in \mathcal{S}, \quad d(A, B) + d(B, C) \geq d(A, C).$$

The pair $(\mathcal{S}, d)$ is called a *metric space*.

Note that we will be dealing with metrics on sets of points (e.g., measuring distance between any two points in $\mathbb{R}^N$) *and sets of sets of points* (e.g., measuring distance between any two sets of points in $\mathbb{R}^N$). When the distinction is required, we will use S to represent a set of points and $\mathcal{S}$ to represent a set of sets. It will be clear from the context when this distinction needs to be made.

It is interesting to note that given metric spaces $(\mathcal{S}, d_1)$ and $(\mathcal{S}, d_2)$ then the following are metric spaces: (a) $(\mathcal{S}, \alpha_1 d_1 + \alpha_2 d_2)$ for $\alpha_1, \alpha_2 \in \mathbb{R}^+$; (b) $(\mathcal{S}, \max(d_1, d_2))$; (c) $(\mathcal{S}, f(d_1))$ where $f(0) = 0$, $f'(0) > 0$, $f'(x) \geq 0$ and $f''(x) \leq 0$ for all $x \in \mathbb{R}^+$. In fact, $f(x)$ need not even be differentiable, as long as it is positive and nondecreasing with nonincreasing slope. Concrete examples of acceptable functions, $f(x)$, are $f(x) = \log(1+x)$, $f(x) = x/(1+x)$, $f(x) = \min(x, T)$ for some fixed threshold $T \in \mathbb{R}^+$, and $f(x) = x^r$ for $0 < r < 1$.

One often encounters functions which are promising because they satisfy the triangle inequality, but fail one or more of the other metric properties. For example, if $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_m\}$ are sets of elements such that $a_i, b_j \in S$ for all i and j, and $\mathcal{S}$ is the set of all finite point sets with elements in S (i.e.,

$a, b \in S$ and $A, B \in \mathcal{S}$) then a *directed Hausdorff metric* on $\mathcal{S}$ is defined as:

$$h(A, B) = \max_{a \in A} \min_{b \in B} d(a, b)$$

where d is any metric on S. It is well known that the triangle inequality holds for $h(A, B)$ (see e.g., Zikan, 1990).

Since in general $h(A, B) \neq h(B, A)$ we may "repair" this by defining the undirected Hausdorff distance as

$$H(A, B) = \max(h(A, B), h(B, A)). \tag{1}$$

The undirected Hausdorff distance does satisfy the metric properties (see e.g., Alt et al., 1988).

A metric that we have shown to be useful in the context of modular robot motion planning, which we refer to as the *optimal assignment metric*, is defined as (Chirikjian et al, 1996; Pamecha et al., 1997):

$$\delta^{(p)}(A, B) = \min_{\pi \in \Pi_n} \sqrt[p]{\sum_{i=1}^{n} d^p\left(a_i, b_{\pi(i)}\right)}, \tag{2}$$

where Π_n is the set of all possible matchings/permutations of labels of the n points and A and B each have n points.

2.3. Metrics on Equivalence Classes

In this subsection we examine how metrics can be used not only to provide a measure of distance between objects in space, but also to determine how similar two objects are.

A metric $d : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ is said to be *G-invariant* if $d(A, B) = d(g \cdot A, g \cdot B) \forall g \in G$ for some group G and all $A, B \in \mathcal{S}$ under the closure condition $g \cdot A$, $g \cdot B \in \mathcal{S}$.

For example, if $S = \mathbb{R}^N$ and $A, B \in \mathcal{S}$ are sets of points in $\mathbb{R}^N$, then the undirected Hausdorff metric, $H(A, B)$, and the optimal assignment metric, $\delta^{(p)}(A, B)$, are $SE(N)$-invariant when $d(a, b) = d_2(a, b)$ (the Euclidean metric). That is, these metrics on point sets inherit invariance under rotation and translation from the underlying metric.

Two objects A and B are said to be *similar*[2] (or equivalent under the action of a group G) if $d(A, g \cdot B) = 0$ for some $g \in G$ and the G-invariant metric $d(\cdot, \cdot)$.

Recall that the concept of "equivalence" of objects A and B (denoted $A \sim B$) satisfies the properties of reflexivity ($A \sim A$), symmetry (if $A \sim B$ then $B \sim A$), and transitivity (if $A \sim B$ and $B \sim C$, then $A \sim C$). An *equivalence class*, $[A] \in \mathcal{S}/\sim$,[3] determined by $A \in \mathcal{S}$ is defined as $[A] = \{A' \in \mathcal{S} : A' \sim A\}$. Any set $\mathcal{S}$ can be decomposed into disjoint equivalence classes (Munkres, 1975).

We note that the function

$$d_G(A, B) = \min_{g \in G} d(A, g \circ B) \tag{3}$$

is a metric on the set of all equivalence classes of $\mathcal{S}$ (which has elements $[A] = \{A' \in \mathcal{S} : d(A, g \circ A') = 0 \text{ for some } g \in G\}$) when $d(A, B)$ is a G-invariant metric on $\mathcal{S}$. See Chirikjian and Kyatkin (2000) for a proof of this.

3. Applications in Modular Robot Motion Planning

This section describes the application of the optimal assignment metric $\delta^{(1)}(\cdot, \cdot)$ to the motion planning of modular robots. We begin by reviewing the concept of a "metamorphic" robot, illustrate the locomotion of such robots, and discuss their configuration space. Finally, applications of the optimal assignment metric to this problem are discussed.

3.1. Defining Properties and an Example

A *metamorphic* system (Chirikjian, 1994) is a collection of independently controlled robotic modules, each of which has the ability to connect, disconnect, and climb over adjacent modules. Each module allows power and information to flow through itself and to its neighbors. A change in the metamorphic robot topology results from the locomotion of each module over its neighbors. Thus a metamorphic system has the ability to dynamically self-reconfigure.

Metamorphic systems can be viewed as a large swarm (or colony) of connected robots which collectively act as a single entity. What distinguishes metamorphic systems from other reconfigurable robots is that they possess all of the following properties:

1. All modules have the same physical structure, and each must have complete computational and communication functionality.
2. Symmetries in the mechanical structure of the modules must be such that they fill planar and spatial regions with minimal gaps.

3. The modules must each be kinematically sufficient with respect to the task of locomotion, i.e., they must have enough degrees of freedom to be able to 'walk' over adjacent modules so that they can reconfigure without outside help.
4. Modules must adhere to adjacent modules, e.g., there must be electromechanical or electromagnetic connectors between modules which can carry load.

Potential applications of metamorphic systems composed of a large number of modules include : (1) obstacle avoidance in highly constrained and unstructured environments; (2) 'growing' structures composed of modules to form bridges, buttresses, and other civil structures in times of emergency; (3) envelopment of objects, such as recovering satellites from space.

One design which satisfies all the properties mentioned above is based on square modules. The square modules can completely fill the plane without any gaps. The centers of the square modules form a Cartesian lattice and each module can be viewed as part of a lattice structure. The fact that the centers of the square modules form a Cartesian lattice is important since we can adapt this concept and expand it to the spatial case with a cubic module design.

Each module, as shown in Fig. 1, carries male or female connectors on each of its edges. Because of the symmetry of the modules, the locomotion always results in edges with opposite polarity (i.e., male/female connectors) meeting with each other. This symmetry is maintained over the entire structure.

The male connector consists of two parallel plates with space in between. There is one driving gear and two rollers mounted on the bottom plate. An H-shaped mating link called the *shuttle* with racks attached to both sides are driven by the driving gear. The shuttle remains connected to the male connector while sliding back and forth in the rails formed by the tracks of both male and female connectors.

The female connector consists of three parallel plates with seven gears mounted on the middle plate. The two outermost gears mesh with the rack of the shuttle of its neighboring male connector. One of these two gears is actuated by a D.C. motor and serves as a driving gear for the female connector. The top and bottom plates, called the *jaws*, can slide open or close along three guiding rods perpendicular to the plates. This opening and closing jaw movement is actuated by a D.C. motor. The jaws remain closed unless there is another module coming from the direction perpendicular to the edge of the female connector. In that case, the jaws open so that the shuttle of the incoming module can come in. The shuttle is locked in position when the jaws close, thus completing the mating process. In the case of releasing the connection, we have to follow the above sequence in reverse order. A total of six D.C. motors are needed in a single module, two each for operating the driving gears

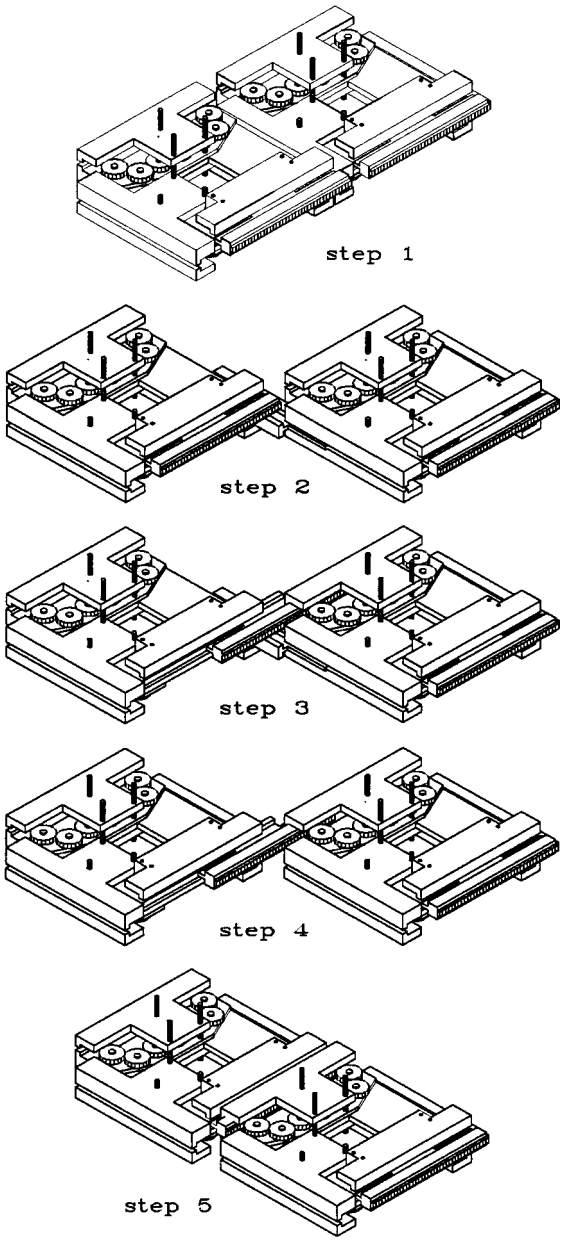

Figure 1. Motion sequence demonstrating the diagonal transformation.

in the male connectors, the driving gears in the female connectors, and the jaws opening/closing mechanisms.

3.2. *The Locomotion Process*

The reconfiguration of metamorphic robots with square modules takes place by the locomotion of modules around each other while remaining connected to each other at all times. There are two types of such 'sliding' motion: (a) Vertical or horizontal motion, which involves the motion of a mobile module in the vertical or horizontal direction; (b) Diagonal motion, which involves the motion of a moving module in the diagonal direction.

The locomotion procedure for vertical or horizontal motion is first to identify the mating pair of connectors and actuate the driving gear of the female connector to move the module in the desired direction. Once the mobile module is making the initial connection with the new neighboring module, the driving gear of the female connector in the new mating pair of connectors is actuated until the mobile module reached the designated position.

The motion sequence demonstrating the locomotion procedure for diagonal motion is shown in Fig. 1, steps 1 to 5. Step 1 shows the original position of the two adjacent modules; the left one, the mobile module, is about to move to its diagonal direction to the top of the fixed module. In step 2 the driving gear of the female connector drives the mobile module halfway up then the driving gear of the male connector takes over and moves the mobile module one full module distance up. In step 3 the driving gear of the male connector in the mobile module slides the shuttle one half of a module distance to the right to make connection with the female connector for the fixed module. In step 4 the connection between the first pair of connectors releases and the shuttle moves back up to its normal position. In step 5 the driving gear for the male connector brings the shuttle back to its normal position while the driving gear for the female connector moves the mobile module further right to the final position. This completes one diagonal move of the module.

3.3. *Configuration Space of Metamorphic Robots*

The configuration space of a metamorphic robot can be viewed as a graph where each vertex represents a single configuration of the robot, and each edge represents the

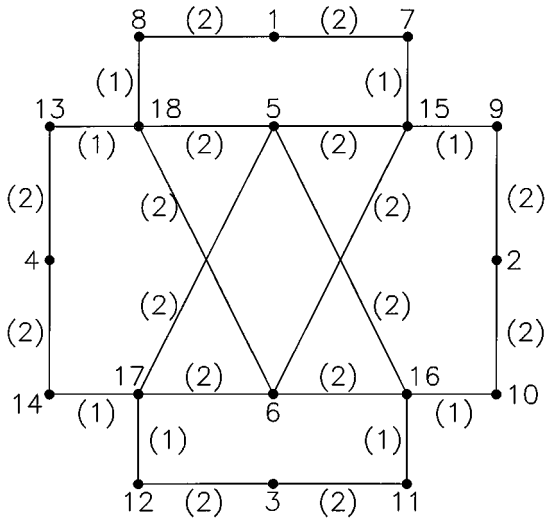

Figure 2. The C-space graph showing the neighboring relations of the configurations.

fewest moves required to get from one configuration to the next closest one. In the case of square modules, we count a diagonal move as two moves, and sliding from one lattice space to an adjacent one as a single move. The graph shown in Fig. 2 is then the configuration space for a metamorphic robot with two square modules which may move relative to a fixed (hatched) square base. Each vertex in this graph corresponds to one of the configurations shown in Fig. 3.

The motion planning problem for metamorphic robots can be formulated as the shortest path in this configuration space graph. The only problem with this approach is that the graph size grows exponentially in the number of movable modules. Thus, it is not practical to do a graph search when there are many movable modules, and a heuristic approach is required. In practice, the optimal assignment metric

$$\delta^{(1)}(A, B) = \min_{\pi \in \Pi_n} \sum_{i=1}^{n} d_1(a_i, b_{\pi(i)})$$

performs quite well as an estimate of the actual number of module moves required to get from one configuration to another, where $d_1(\cdot, \cdot)$ is the Cartesian lattice distance (which reflects module motion constraints).

A good lower bound on the minimal number of moves required to reconfigure a metamorphic robot is also obtained by using the lattice metric and optimal assignment (Pamecha et al., 1997). This lower bound is based on the fact that the minimal number of single-module moves required for a single module motion in a

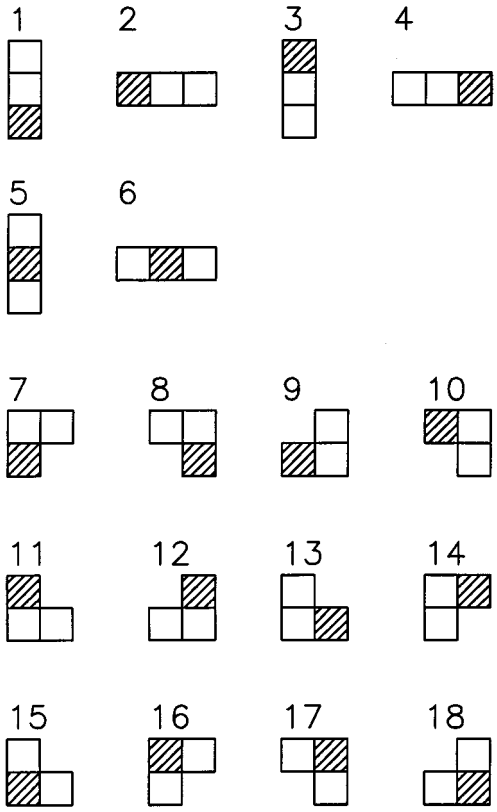

Figure 3. Configurations of a metamorphic robot with 2 movable square modules.

lattice will be no less than the lattice distance between the initial and final spaces. Furthermore, if it were possible to track the sequence of motions of an optimally reconfiguring metamorphic robot, we could compute the lattice distance between each module in its initial and final lattice spaces, and the sum would be a lower bound on the total number of module motions. Since this is not possible, we will assign modules in two configurations in such a way that the sum of the lattice distances between matched modules is minimized over all possible matchings. This minimal sum will be at most the aforementioned lower bound. Since this *is* something that can be computed relatively efficiently, this is the lower bound we will use. Thus the optimal assignment metric can be used as both a cost function, and lower bound on the number of single-module moves required to complete the reconfiguration process. A detailed comparison of this in the context of the metamorphic robot motion planning problem for the case of hexagonal modules can be found in Pamecha et al. (1997).

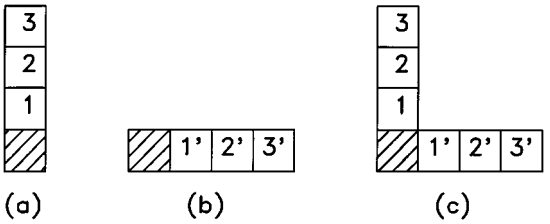

Figure 4. (a) present configuration, (b) new configuration, (c) module labeling.

3.4. An Example Calculation

In this subsection we use one example to demonstrate how an optimal assignment is calculated.

Consider the example in Fig. 4. Figure 4(a) shows the present configuration, Fig. 4(b) shows the new configuration and Fig. 4(c) shows a labeling of the modules in the two configurations.

The matrix $\mathbf{D} = [d_{ij'}]$ formed by the distances between various modules is shown in Eq. (4).

$$\mathbf{D} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1' & 2' & 3' \end{array} \\ \begin{pmatrix} 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{pmatrix} \end{array}. \tag{4}$$

Performing column operations (subtracting l_p, the minimum element of each column, from each column respectively), we get the matrix in Eq. (5). Similarly performing the row operations, we get the *reduced matrix* in Eq. (6).

$$\mathbf{D}' = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 2 & 2 \end{pmatrix} \tag{5}$$

$$\bar{\mathbf{D}} = \begin{pmatrix} \boxed{0} & 0 & 0 \\ 0 & \boxed{0} & 0 \\ 0 & 0 & \boxed{0} \end{pmatrix} \tag{6}$$

The *reduced matrix* $\bar{\mathbf{D}}$ contains several combinations of three independent 0′*s* any of which solves the problem and gives the value of $\delta^{(1)}(A, B)$ by summing entries in the original matrix in the same places as the independent zeroes.

Choosing the boxed solution above, the value of $\delta^{(1)}(A, B)$ is given as,

$$\delta^{(1)}(A, B) = d_{11'} + d_{22'} + d_{33'} = 2 + 4 + 6 = 12.$$

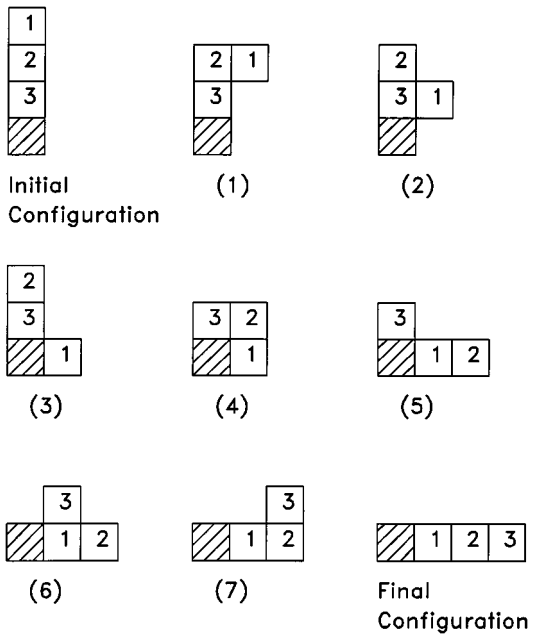

Figure 5. A complete reconfiguration sequence involving two serial configurations.

The minimal value is achieved by matching modules with the same subscripts in the above expression.

Figure 5 shows a complete reconfiguration sequence involving two serial structures. For each move, the moving module moves into an empty space while remaining connected to at least one other module. Also, at each time, only one module is allowed to move.

Since we are counting a diagonal move as two moves, the above reconfiguration sequence takes 12 moves in total. It is also the minimum number of moves required to realize the reconfiguration process since $\delta^{(1)}(A, B) = 12$, and we know this is a lower bound on the minimal number of moves.

4. Using Metrics for Bisecting Configurations

The motivation for bisecting configurations is to divide the motion planning problem into smaller pieces by generating a certain number of intermediate configurations and use them as consecutive goal configurations to improve the overall performance and avoid some local minima during normal motion planning processes. Theoretically, if we keep bisecting configurations the distance between consecutive intermediate configurations will be down to either 1 or 2 (within a single module motion). By doing so, we can put the starting and final configurations together with these intermediate configurations in sequence thus solving the motion planning problem without going through the simulated annealing algorithm used in Pamecha et al. (1997).

4.1. *The Basic Idea*

This problem is very similar to the shape interpolation problem of two given sets of points. In shape interpolation, we first find an optimal assignment between the two sets of points then calculate the "average" set of points in the middle.

However, there are several essential aspects in bisecting configurations that mark the differences between these two problems. First of all, we have to make sure the new configuration generated is connected to meet the kinematic property of a metamorphic robotic system. Secondly, the mid-configuration must have the same number of modules as the starting and final configurations. Since all modules must be placed on lattice points only, it is possible that there is more than one average of matched pairs occupying the same position. In such cases, we have to put back more modules until the numbers of modules agree. Thirdly, we have to be sure that the mid-configuration is properly "centered", i.e., the distance between starting configuration and mid-configuration and the distance between mid-configuration and final configuration should be approximately the same.

A "straightforward" bisection, which simply takes the average of each optimally assigned pair of modules, will not work. A bisection following this method usually will result in overlapping module paths and/or modules not sitting on the lattice. For an illustration of a straightforward bisection, consider the following example. Figure 6(a) shows the initial configuration of a robot with 3 movable modules, Fig. 6(b) shows the final configuration and Fig. 6(c) shows the overlap view of the two configurations.

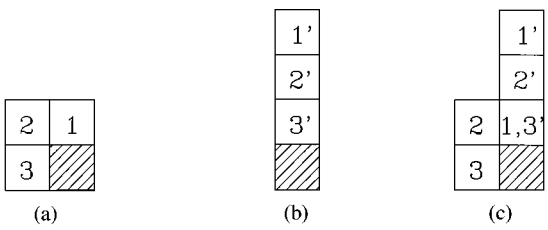

Figure 6. Straightforward bisection example: (a) initial configuration, (b) final configuration, (c) overlap.

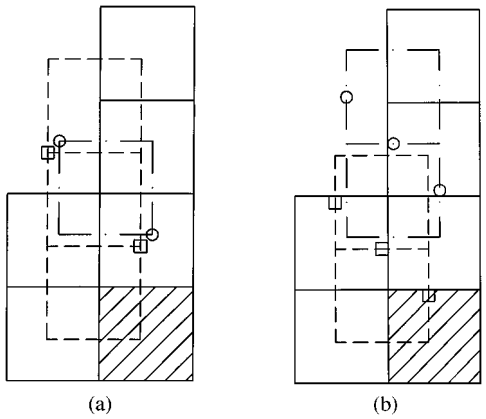

Figure 7. Optimally assigned paths for two possible optimal assignments: (a) $1 \to 3', 2 \to 2', 3 \to 1'$; (b) $1 \to 3', 2 \to 1', 3 \to 2'$.

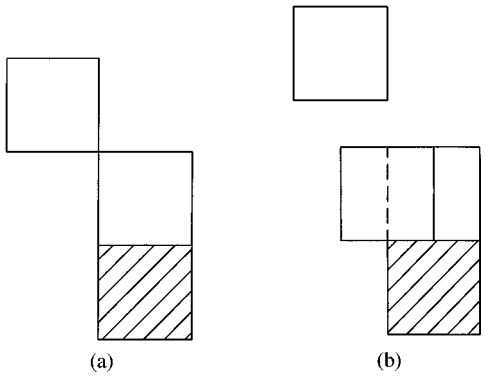

Figure 8. One possible bisected configuration for each optimal assignment.

There are two sets of possible optimal assignments for the given initial and final configurations:

$$\begin{aligned} &(a) 1 \to 3', 2 \to 2', 3 \to 1' \\ \text{and} \quad &(b) 1 \to 3', 2 \to 1', 3 \to 2' \end{aligned}$$

Figure 7 shows the respective optimally assigned paths of the two cases. Figure 8 shows one possible bisected configuration for each matching. From Fig. 8(a) we can see that a straightforward bisection can have overlapped modules and the bisected configuration is not connected. From Fig. 8(b) we see that not all modules are on the lattice. Clearly, this method does not work.

In the next subsection, a method to *grow* the mid-configuration according to an algorithm for distributing a weight index to proper lattice grid points is described. Any mid-configurations generated by this method will meet the three characteristics discussed above.

4.2. *Weight Distribution Algorithm*

The first step in bisection configurations is to find an optimal assignment between the starting and final configurations. For each pair of matched modules, calculate the average of the x and y coordinates to give the coordinates of a module in the middle. Since each module has to fall on the square lattice, non-integer module coordinates are not accepted. We have to find a way to distribute the unit weight associated with the module with non-integer coordinate to several adjacent lattice points when this happens.

Let the unit weight of one module be 16 (it will become clear later why 16 is chosen). There are four possible combinations of the (x, y) coordinates of the averaged position of each matched pair.

1. Both x and y coordinates are integers. In this case, the total weight of 16 is placed at a single point (x, y).
2. Only the x coordinate is an integer. In this case, the weight is distributed equally between 2 adjacent points $(x, y - 1/2)$ and $(x, y + 1/2)$ so that each point has a weight of 8.
3. Only the y coordinate is an integer. In this case, the weight is distributed equally between 2 adjacent points $(x - 1/2, y)$ and $(x + 1/2, y)$ so that each point has a weight of 8.
4. Neither coordinate is an integer. In this case, the weight is distributed equally among 4 adjacent points $(x, y - 1/2)$, $(x, y + 1/2)$, $(x - 1/2, y)$ and $(x + 1/2, y)$ so that each point has a weight of 4.

See Fig. 9 for an illustration of proper weight distribution.

If, in any case, the base module $(0, 0)$ has non-zero weight, we should redistribute the weight assigned to it to its 4 neighboring points with each of the 4 points gets one fourth of the original weight. The unit weight is chosen to be 16 so that after such redistribution the minimal weight of any point will still be at least 1.

After we have finished n such weight distributions (n is the number of modules in the starting configuration) for the n pairs of matched modules, we have a set of lattice grid points with variable density of weight index. We will then grow the mid-configuration from the base module according to the overall weight distribution pattern. We first initialize the mid-configuration to contain the base module only. In the first step, we generate a neighbor list containing all the neighboring points of the current mid-configuration. Any new

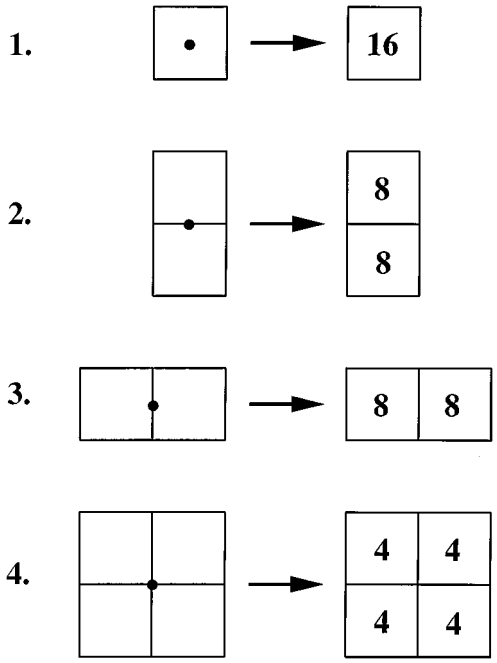

Figure 9. Four cases of weight distribution.

module added can only be picked from this neighbor list to ensure overall connectivity. In step two, we check all the points in the neighbor list to pick a point (x_α, y_α) with maximal density α. If α is greater than zero, we will just add (x_α, y_α) to the mid-configuration and increase the total number of modules by 1. If the module count is n then the mid-configuration is complete, otherwise we have to go back to step 1 to start the next iteration. If α is zero, that means all the candidate points have zero weight index. Since new modules can only be picked from this list, we need a new measure to assign a new index to each of the points in the neighbor list to justify the selection of any specific point from the list. The new measure is achieved by assigning a neighbor index to each point in the neighbor list. The neighbor index is calculated by summing the weight index of its 4 neighboring points. We then check all the points in the neighbor list to pick a point (x_β, y_β) with maximal neighbor index β. We add the point (x_β, y_β) to the mid-configuration based on the assumption that a point with larger neighbor index has a better chance of being picked as part of mid-configuration. We then increase the total number of modules by 1. If the module count is n the mid-configuration is complete, otherwise we go back to step 1 to start the next iteration.

In summary, starting with the base module only, the algorithm adds one module at a time to the mid-configuration at the previous step. The key point is to maintain the connectivity of mid-configuration at all time so we don't need the connectivity check after the procedure. The algorithm stops when the total number of modules of the mid-configuration agrees with that of the starting or final configuration.

Below pseudocode is provided which implements the above discussion.

Weight Distribution Algorithm for Generating Weight Pattern

Step 0
- Start with a mid-configuration containing the base module (0, 0) only
- Module Count = 0
- Go to step 1

Step 1
- Generate the neighbor list from the neighbor of mid-configuration
- Go to step 2

Step 2
- Go through the neighbor list to pick a point (x_α, y_α) with maximal density α.
- If $\alpha > 0$, go to step 3a,
- Else go to step 3b

Step 3a
- Add (x_α, y_α) to mid-configuration
- Increase Module Count by 1
- If Module Count = n, go to step 4, else go back to step 1

Step 3b
- Assign a neighbor index to each point in neighbor list by summing the weight index of its 4 neighboring points
- Go through the neighbor list to pick a point (x_β, y_β) with maximal neighbor index β.
- Add (x_β, y_β) to mid-configuration
- Increase Module Count by 1
- If Module Count = n, go to step 4, else go back to step 1

Step 4
- Mid-configuration is complete. Stop.

5. Simulation Results

5.1. *Examples of Bisecting Configurations*

In this subsection, we will use the bisecting techniques described in the previous section to find the mid-configuration of any two given initial and final

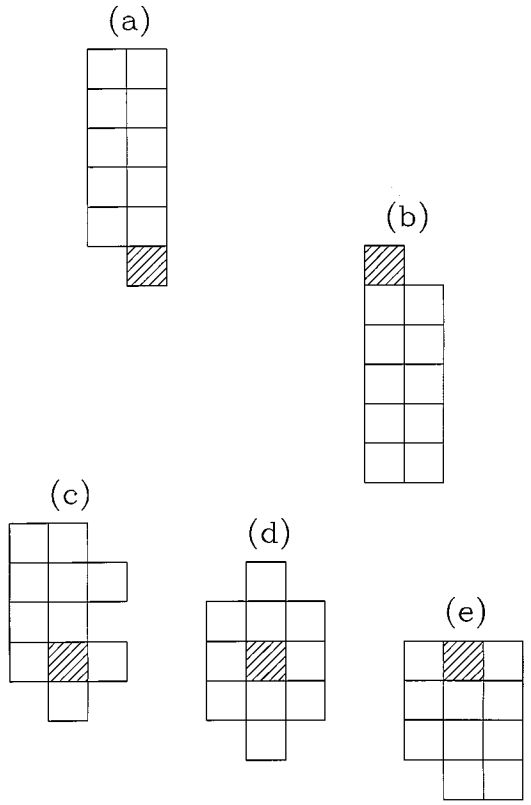

Figure 10. Example 1: (a) initial configuration, (b) final configuration, (c) 1/4 configuration, (d) mid-configuration, (e) 3/4 configuration.

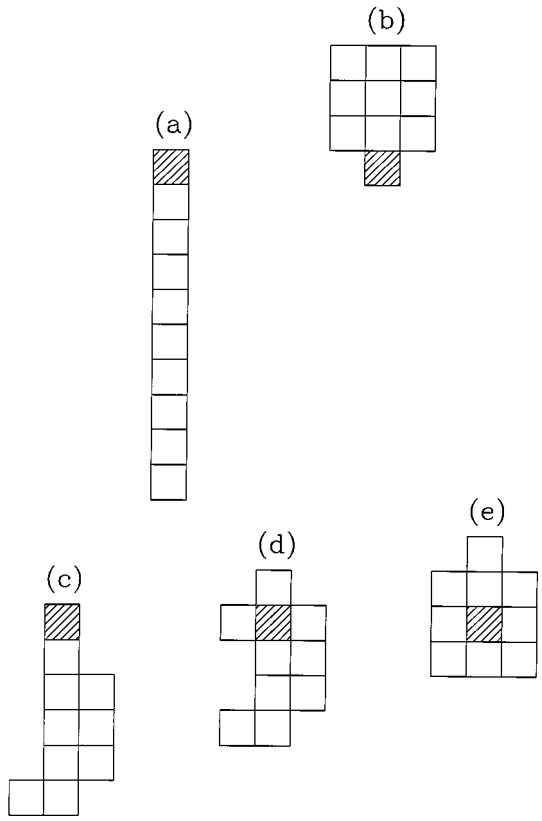

Figure 11. Example 2: (a) initial configuration, (b) final configuration, (c) 1/4 configuration, (d) mid-configuration, (e) 3/4 configuration.

configurations. In the current simulation codes, we first ask for the user to input the initial and final configurations in two separate windows. The codes will first generate the mid-configuration of the two given starting configurations then generate the 1/4 configuration by bisecting with the initial and the mid-configuration and generate the 3/4 configuration by bisecting with the mid-configuration and the final configuration. At the end of the simulation, the codes will output the 1/4, mid-, and 3/4 configurations to three separate windows. This is a 2-level bisection. For a k-level bisection of the configurations, we will have $2^k - 1$ intermediate configurations. Figures 10 to 12 show the results of the simulation with three sets of initial and final configurations.

5.2. *Motion Planning with Intermediate Configurations*

In this subsection, we will apply the idea of bisecting configurations to the motion planning problem

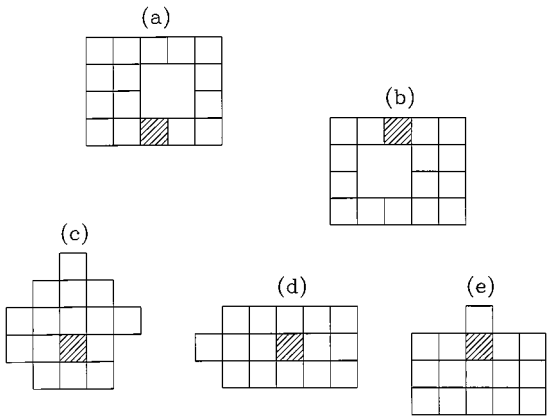

Figure 12. Example 3: (a) initial configuration, (b) final configuration, (c) 1/4 configuration, (d) mid-configuration, (e) 3/4 configuration.

of metamorphic robots with square modules. To generate motion between configurations we use a variant on the simulated annealing algorithm in Pamecha et al. (1997). We use the three sets of initial and final

Table 1. Results for configurations of Fig. 10. Distance between initial and final configurations is 70.

Bisection	Avg. moves	Max. moves	Min. moves
No	115.5	288	80
1-level	111.3	202	78
2-level	98.3	170	80
3-level	98.0	136	82

Table 2. Results for configurations of Fig. 11. Distance between initial and final configurations is 69.

Bisection	Avg. moves	Max. moves	Min. moves
No	131.8	277	85
1-level	102.1	133	81
2-level	92.6	111	87
3-level	91.0	103	85

Table 3. Results for configurations of Fig. 12. Distance between initial and final configurations is 52.

Bisection	Avg. moves	Max. moves	Min. moves
No	194.7	308	92
1-level	153.5	244	88
2-level	115.3	158	92
3-level	115.0	176	90

configurations in Figs. 10 to 12 to run the simulations. We ran ten trials of motion planning simulations on each set with no bisection, 1-level, 2-level, and 3-level bisections respectively and record the total distance traveled (or the total number of moves made). The intermediate configurations are used as consecutive goal configurations during the simulation. The results for the three cases are shown in Tables 1 to 3.

As can be seen from the tables, more levels of bisection yields better results in all three cases. Although the improvement for a 3-level bisection over a 2-level bisection is not obvious, the performance of a 2-level bisection is much better than the ones with no or only 1-level of bisection. Generally, when more intermediate configurations are used as consecutive goal configurations during the motion planning, the chances of reaching a local minima are reduced, thus cutting down oscillations and reducing the total number of moves made. The fact that there is no significant improvement for 3-level bisections over 2-level bisections in the examples can be viewed as the numbers of local minima between intermediate configurations in both cases are similar.

5.3. *Branch Motion*

Instead of moving only one module in each reconfiguration step, we would like to explore the possibility for multiple-module motion or branch motion. There are several general questions regarding the branch motion:

1. How do we recognize the branches in the configurations?
2. How do we check the mobility for the branches?
3. We need to evaluate the cost or efficiency for both single module and branch motions, i.e., how and when do we want to make a branch motion instead of a single module motion at a certain step?

Since the reconfiguration process for metamorphic robots using square modules consists of translations only, the shape and orientation for the branches shall remain the same before and after the motion. If we try to enumerate and examine every possible branch in the whole configuration without setting an upper limit of the number of modules in a branch, the problem will become unmanageable. To simplify the problem we only look at two-module branches as a start.

In this context, two branches A and B are similar if they are translationally equivalent. I.e., if $d_G(A, B)$ in Eq. (3) is zero, where G is the group of translations in the square lattice.

Not every branch is movable due to the kinematic constraints, especially for those branches close to the base module. However, for every movable branch, at least one of the modules must be a movable module under single module motion. With this in mind, when we try to look for appropriate branch candidates in the whole configuration, we don't need to go through each and every one of the modules. Instead, we generate a list consisting of movable modules in the single module motion. For every module in the list we then check for possible branch candidates. A two-module branch is formed by attaching a module to any module in the list in one of its four neighboring positions. If the attached module is also a member of the original configuration then the branch is a valid branch candidate, otherwise the proposed branch does not exist.

There are two types of two-module branches: vertical and horizontal. For each type of a vertical or horizontal branch, there are twelve different motion patterns: up,

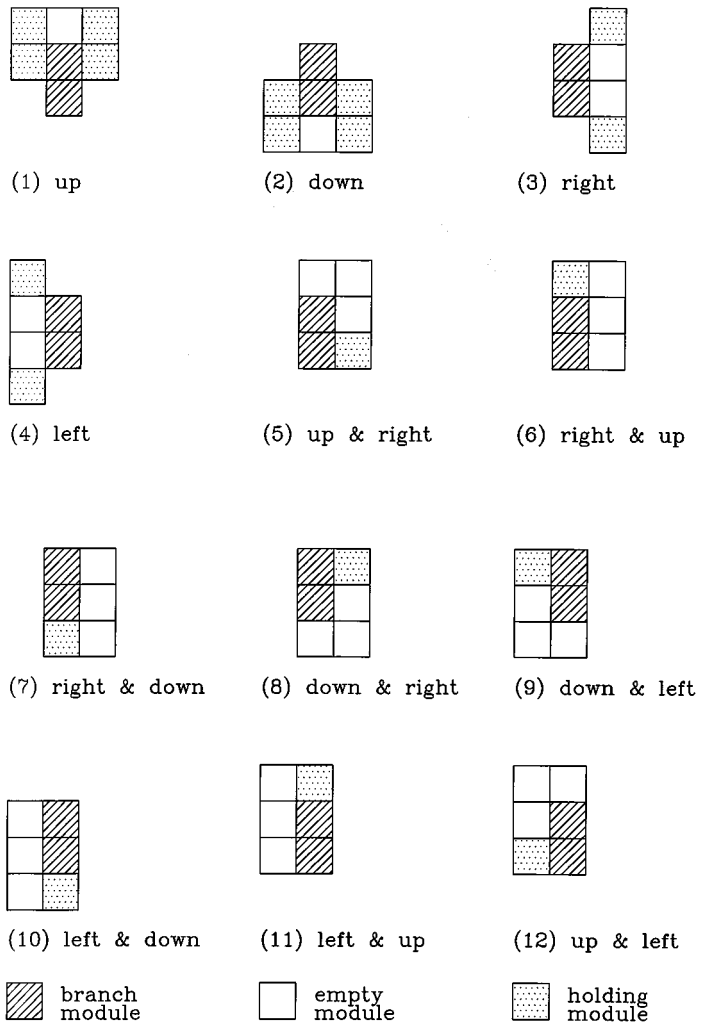

Figure 13. Twelve motion patterns for a vertical branch.

down, right, left, up and right, right and up, right and down, down and right, down and left, left and down, left and up, and up and left. Depending on the type of the branch and the intended motion pattern there will be some more conditions the branch candidate must satisfy to have a valid branch motion. For example, if a given vertical branch with module coordinates (x, y) and $(x, y+1)$, respectively, is to move up in the y-direction, the new branch module coordinates will be $(x, y+1)$ and $(x, y+2)$, respectively. Therefore, the position $(x, y+2)$ in the lattice must be empty before the branch motion. In addition, there should be at least one module available for the branch to hold on to during the motion. In this case, the holding module can be any of the four positions: $(x+1, y+2)$, $(x-1, y+2)$, $(x+1, y+1)$, and $(x-1, y+1)$. The final condition is that after the branch motion, the new configuration must remain connected.

Figures 13 and 14 show the physical relationships among branch modules, empty module positions and holding modules for different motion patterns with vertical branch and horizontal branch, respectively. The corresponding coordinates are listed in Tables 4 and 5.

Now that we know how to check the mobility of a branch candidate, the next question is when do we make a branch motion instead of a single module motion at a certain step.

Let's denote the distance between the current configuration and the final configuration as δ and the distance after a certain move as δ'. For a single module motion,

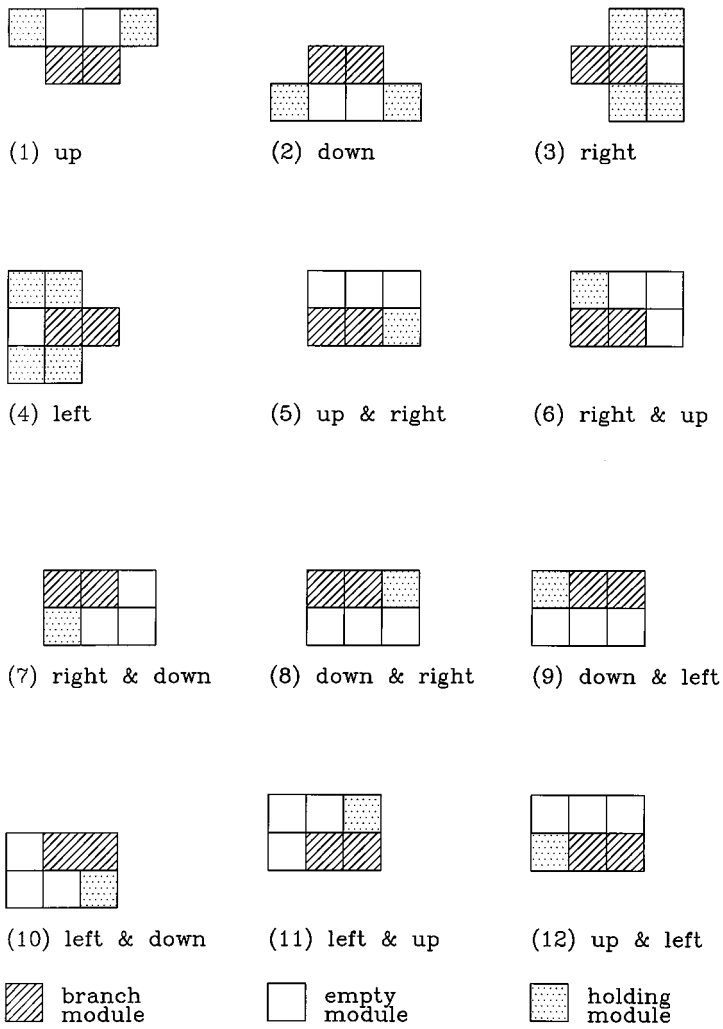

Figure 14. Twelve motion patterns for a horizontal branch.

$\Delta\delta = \delta' - \delta$ can be $+2, +1, 0, -1$ or -2 whereas for a branch motion, $\Delta\delta$ can be $+4, +2, 0, -2$ or -4. Potentially, a branch motion can decrease the distance more quickly and possibly can prevent the reconfiguration process from getting stuck in some local minima. So at a certain step, if there is some branch motion available that will decrease the distance, i.e., $\Delta\delta$ is either -2 or -4, it is preferable to make the branch motion.

In summary, we have to first generate a list consisting of movable modules. We then check for branch candidates from every module in the list with its neighboring modules. If any branch candidate along with some type of motion patterns can decrease the distance we will make the proposed branch motion. If no branch motion can decrease the distance after we have searched through all possible branch candidates then a single module motion is made. After each step, whether it's branch motion or single module motion, the movable module list is updated and the reconfiguration process continues until the final configuration is reached.

Figure 15 shows an example for the test of the branch motion code. Figure 15(a) is the initial configuration and Fig. 15(b) is the final configuration. The simulation results after ten trials of the branch motion code and the single module motion code are shown in Table 6.

We also test the three sets of initial and final configurations in Figs. 10 to 12 of the previous section. The results are listed in Tables 7 to 9.

As can be seen from tables, the branch motion code performs better than the single module motion code

Table 4. List of Coordinates for different motion patterns for a vertical branch with module coordinates (x, y) and $(x, y+1)$.

Intended motion	Empty module positions	Holding modules
Up	$(x, y+2)$	$(x+1, y+2), (x-1, y+2)$ $(x+1, y+1), (x-1, y+1)$
Down	$(x, y-1)$	$(x+1, y), (x-1, y)$ $(x+1, y-1), (x-1, y-1)$
Right	$(x+1, y+1), (x+1, y)$	$(x+1, y+2), (x+1, y-1)$
Left	$(x-1, y+1), (x-1, y)$	$(x-1, y+2), (x-1, y-1)$
Up and right	$(x+1, y+1), (x+1, y+2)$ $(x, y+2)$	$(x+1, y)$
Right and up	$(x+1, y+1), (x+1, y+2)$ $(x+1, y)$	$(x, y+2)$
Right and down	$(x+1, y), (x+1, y-1)$ $(x+1, y+1)$	$(x, y-1)$
Down and right	$(x+1, y), (x+1, y-1)$ $(x, y-1)$	$(x+1, y+1)$
Down and left	$(x-1, y), (x-1, y-1)$ $(x, y-1)$	$(x-1, y+1)$
Left and down	$(x-1, y), (x-1, y-1)$ $(x-1, y+1)$	$(x, y-1)$
Left and up	$(x-1, y+1), (x-1, y+2)$ $(x-1, y)$	$(x, y+2)$
Up and left	$(x-1, y+1), (x-1, y+2)$ $(x, y+2)$	$(x-1, y)$

Table 5. List of Coordinates for different motion patterns for a horizontal branch with module coordinates (x, y) and $(x+1, y)$.

Intended motion	Empty module positions	Holding modules
Up	$(x, y+1), (x+1, y+1)$	$(x-1, y+1), (x+2, y+1)$
Down	$(x, y-1), (x+1, y-1)$	$(x-1, y-1), (x+2, y-1)$
Right	$(x+2, y)$	$(x+1, y-1), (x+2, y-1)$ $(x+1, y+1), (x+2, y+1)$
Left	$(x-1, y)$	$(x-1, y-1), (x, y-1)$ $(x-1, y+1), (x, y+1)$
Up and right	$(x+1, y+1), (x+2, y+1)$ $(x, y+1)$	$(x+2, y)$
Right and up	$(x+1, y+1), (x+2, y+1)$ $(x+2, y)$	$(x, y+1)$
Right and down	$(x+1, y-1), (x+2, y-1)$ $(x+2, y)$	$(x, y-1)$
Down and right	$(x+1, y-1), (x+2, y-1)$ $(x, y-1)$	$(x+2, y)$
Down and left	$(x-1, y-1), (x, y-1)$ $(x+1, y-1)$	$(x-1, y)$
Left and down	$(x-1, y-1), (x, y-1)$ $(x-1, y)$	$(x+1, y-1)$
Left and up	$(x-1, y+1), (x, y+1)$ $(x-1, y)$	$(x+1, y+1)$
Up and left	$(x-1, y+1), (x, y+1)$ $(x+1, y+1)$	$(x-1, y)$

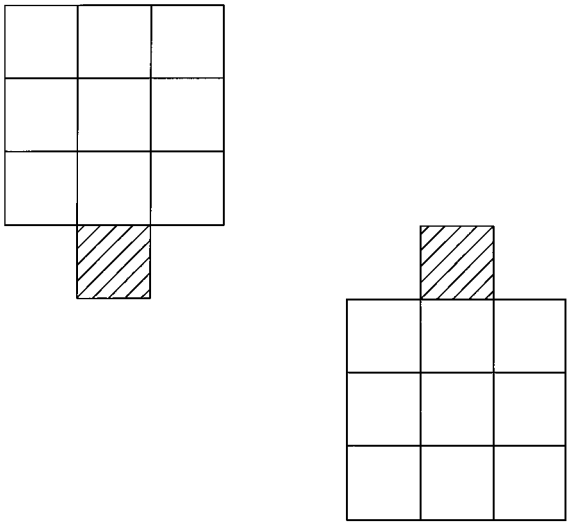

Figure 15. Example 4: (a) initial configuration, (b) final configuration.

in all examples tested. So the condition we imposed on searching for branch motion candidates that decrease the distance is valid. An interesting observation is to compare the results for motion planning with intermediate configurations and with branch motions. For the three examples in Figs. 10 to 12, only configurations in Fig. 11 yields better results using bisected configurations. Both Figs. 10 and 12 yield better results using branch motion.

Table 6. Results for configurations of Fig. 15. Distance between initial and final configurations is 36.

Motion type	Avg. moves	Max. moves	Min. moves
Single module	59.6	78	48
Branch	51.4	72	42

Table 7. Results for configurations of Fig. 10. Distance between initial and final configurations is 70.

Motion type	Avg. moves	Max. moves	Min. moves
Single module	115.5	288	80
Branch	87.2	102	82

Table 8. Results for configurations of Fig. 11. Distance between initial and final configurations is 69.

Motion type	Avg. moves	Max. moves	Min. moves
Single module	131.8	277	85
Branch	94.4	173	77

Table 9. Results for configurations of Fig. 12. Distance between initial and final configurations is 52.

Motion type	Avg. moves	Max. moves	Min. moves
Single module	194.7	308	92
Branch	88	176	70

6. Conclusions

A number of metrics on geometric objects and equivalence classes of geometric objects were reviewed. We formulated a technique that uses these metrics to bisect two shapes. We showed how these concepts can be used to generate sequences of intermediate shapes that interpolate two given shapes. The motivation for this work was modular robot motion planning, and applications in this area were discussed. In future work, we intend to combine the ideas of branch motion and bisected configurations as well as searching for longer or differently shaped branches.

Acknowledgment

This work was supported by NSF Award IIS 9731720; and a 1994 Presidential Faculty Fellow's Award.

Notes

1. A binary operation acts in such a way that $g_1 \circ g_2 \in G$ for all $g_1, g_2 \in G$.
2. Note this definition is a generalization of the standard concept.
3. Often the notion E_A is used to denote an equivalence class, whereas we use $[A]$. The notation $S/\sim$ reflects the fact that an equivalence relation divides a set into disjoint equivalence classes.

References

Alt, H., Melhorn, K., Wagener, H., and Welzl, E. 1988. Congruence, similarity and symmetries of geometric objects. *Discrete Comput. Geom.*, 3:237–256.

Beni, G. 1988. Concept of cellular robotic systems. In *IEEE Int. Symposium on Intelligent Control*, Arlington, VA, August 24–26, 1988.

Chen, I. and Burdick, J. 1993. Enumerating non-isomorphic assembly configurations of a modular robotic system. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Yokohama, Japan, pp. 1985–1992.

Chirikjian, G.S. 1993. Metamorphic hyper-redundant manipulators. In *Proceedings of the 1993 JSME International Conference on Advanced Mechatronics*, Tokyo, Japan, August 1993, pp. 467–472.

Chirikjian, G.S. 1994. Kinematics of a metamorphic robot system. In *Proceedings of the 1994 IEEE Int. Conf. on Robotics and Automation*, San Diego, CA, May 1994, pp. 449–455.

Chirikjian, G., Pamecha, A., and Ebert-Upoff, I. 1996. Evaluating efficiency of self-reconfiguration in a class of modular robots. *J. of Robotic Systems*, 13(5):317–338.

Chirikjian, G.S. and Kyatkin, A.B. 2000. *Engineering Applications of Noncommutative Harmonic Analysis*, CRC Press: Boca Raton, Florida.

Fukuda, T. and Kawauchi, Y. 1990. Cellular robotic system (CEBOT) as one of the realization of self-organizing intelligent universal manipulator. In *Proceedings of the 1990 IEEE Conference on Robotics and Automation*, pp. 662–667.

Fukuda, T., Kawauchi, Y., and Hara, F. 1991. Dynamic distributed knowledge system in self-organizing robotic systems; CEBOT. In *1991 IEEE Conference on Robotics and Automation*, pp. 1616–1621.

Hackwood, S. and Wang, J. 1988. The engineering of cellular robotic systems. In *IEEE Int. Symposium on Intelligent Control*, Arlington, VA, August 24–26, 1988.

Hamlin, G.J. and Sanderson, A.C. 1997. TETROBOT: A modular approach to parallel robotics. *IEEE Robotics and Automation Magazine*, 4(1):42–50, March 1997.

Hosokawa, K., Fujii, T., Kaetsu, H., Asama, H., Kuroda, Y., and Endo, I. 1999. Self-organizing collective robots with morphogenesis in a vertical plane. *JSME International Journal Series C-Mechanical Systems Machine Elements and Manufacturing*, 42(1):195–202.

Kelmar, L. and Khosla, P.K. 1988. Automatic generation of kinematics for a reconfigurable modular manipulator system. In *Proc. 1988 IEEE Conf. on Robotics and Automation*, pp. 663–668.

Kokaji, S. 1988. A fractal mechanism and a decentralized control method. In *Proc. U.S.-Japan Symp. Flexible Automation*, pp. 1129–1134.

Kotay, K., Rus, D., Vona, M., and McGray, C. 1999a. The self-reconfiguring molecule: Design and control algorithms. In *1999 Workshop on Algorithmic Foundations of Robotics*.

Kotay, K. and Rus, D. 1999b. Locomotion versatility through self-reconfiguration, *Robotics and Autonomous Systems*, 26:217–232.

Lipson, H. and Pollack, J.B. 2000. Towards continuously reconfigurable self-designing robotics. In *Proc. 2000 IEEE International Conference on Robotics and Automation*, San Francisco, CA, April 2000, pp. 1761–1766.

Munkres, J.R. 1975. *Topology: A First Course*, Prentice Hall, Inc.,: Englewood Cliffs, NJ.

Murata, S., Kurokawa, H., and Kokaji, S. 1994. Self-assembling machine. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, San Diego, CA, pp. 441–448.

Murata, S., Kurokawa, H., Yoshida, E., Tomita, K., and Kokaji, S. 1998. A 3-D self-reconfigurable structure. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, Leuven.

Pamecha, A., Chiang, C.-J., Stein, D., and Chirikjian, G.S. 1996. Design and implementation of metamorphic robots. In *Proceedings of the 1996 ASME Design Engineering Technical Conference and Computers in Engineering Conference*, Irvine, CA.

Pamecha, A., Ebert-Uphoff, I., and Chirikjian, G.S. 1997. Useful metrics for modular robot motion planning. *IEEE Transactions on Robotics and Automation*, 13(4):531–545.

Yim, M. 1993. A reconfigurable modular robot with many modes of locomotion. In *Proceedings of the 1993 JSME International Conference on Advanced Mechatronics*, Tokyo, Japan, pp. 283–288.

Yim, M. 1994. New locomotion gaits. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, San Diego, CA.

Zikan, K. 1990. The theory and applications of algebraic metric spaces. Ph.D. Thesis, Stanford University.

Gregory S. Chirikjian was born August 16, 1966 in New Brunswick, New Jersey, USA. He received the B.S.E. degree in engineering mechanics, the M.S.E. degree in mechanical engineering, and the B.A. degree in mathematics, all from The Johns Hopkins University, Baltimore, MD, in 1988. He then received the Ph.D. degree from the California Institute of Technology, Pasadena, CA, in 1992. Since the summer of 1992, he has been with the Department of Mechanical Engineering, Johns Hopkins University, where he is now an Associate Professor. Dr. Chirikjian is a 1993 NSF Young investigator, a 1994 Presidential Faculty Fellow, and a 1996 recipient of the ASME Pi Tau Sigma Gold Medal. His research interests include the kinematic analysis, motion planning, design, and implementatation of "hyper-redundant," "metamorphic," and "binary" manipulators. In recent years Dr. Chirikjian has expanded the scope of his research to include applications of group theory in a variety of engineering disciplines.