Rearranging agents in a small space using global controls

Yinan Zhang^{1,*}, Xiaolei Chen^{2,*}, Hang Qi^{3} and Devin Balkcom^{4}

***Abstract*— This paper explores a problem of reconfiguration of a large set of agents using global control signals: *right*, *left*, *up* and *down*. A field of obstacles guides the reconfiguration by limiting motion in the various directions. This paper extends work by Becker *et al.* ([4], [5] and [7]), and shows there exists a workspace a constant factor larger than the number of agents that enables complete rearrangement for a rectangle of agents.**

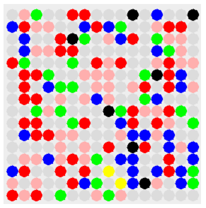
I. INTRODUCTION

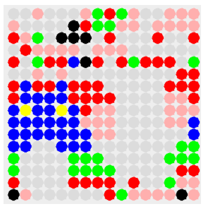
This paper explores the problem of re-configuring a group of simple agents using a sequence of global controls which every agent must follow. If the global control *right* is issued, then all agents move to the right, unless they are stopped by an obstacle or another stationary agent. The other available controls are *left*, *up* and *down*. Figure 1 shows a grid with obstacles generated by our algorithm for rearranging 16×16 matrices of agents, and its application to rearrange an initially randomized Super Mario image.

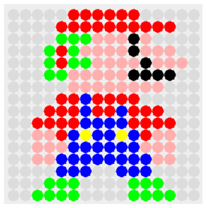
This paper is primarily inspired by a collection of prior papers by Becker *et al.* ([4], [5] and [7]) that use the same model, and show that by placing obstacles cleverly, sequences of global controls can allow essentially arbitrary re-configuration of the agents. The primary contribution of this paper is the observation that with different placements of obstacles, the total physical space requirements (the board that the agents live on), may be much smaller than in previous work. Specifically, general re-arrangement of an $n \cdot m$ rectangle of agents requires a board that is only a constant factor larger than the number of agents. For a specific $n \cdot m$ matrix of agents, constant-time rearrangement can be done in a $O(n \cdot m \cdot (n + m))$ workspace.

There are two approaches to placing obstacles. In the first, simpler approach, *task-specific design*, the obstacles are placed based on the specific start and desired goal arrangement of the agents. This approach seems impractical because a new board must be designed for each arrangement task. In the second approach, *multi-purpose design*, the obstacles are placed in fixed locations, and more complex sequences of actions make use of those obstacles to rearrange the agents. This approach, while more general, does have the disadvantage that the reconfiguration algorithms so far discovered act in serial on each agent or small groups of agents, so that action sequences might be quite long. A primary goal of future work is to parallelize the rearrangement process, so that a much smaller number of moves is required.

* Co-first author
^{1,2,3,4}Department of Computer Science, Dartmouth College, NH, USA


(a) A randomized arrangement of Super Mario image.


(b) Mario image rearrangement 66% done.


(c) The target rearrangement of Super Mario image.

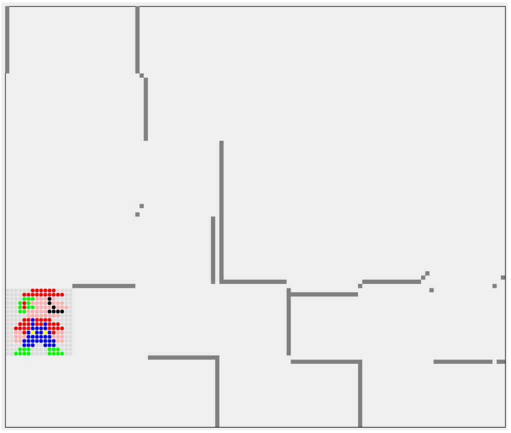

(d) The space designed for rearranging the Super Mario image. Dark gray squares are obstacles

Fig. 1: Rearranging a Super Mario image (16×16) in a grid space using Bubble Sort with limited global controls. The grid space size is linear to the number of elements in the Super Mario image. Rearrangement is completed after 553583 moves.

A. *Task-specific design*

When agents are initially in a single contiguous row, we provide an algorithm to place obstacles in the space such that these agents reorder themselves within that row, using a sequence of four actions.

Using the technique for rearranging a row of agents as a subroutine, we derive an algorithm for rearranging a specific $n \cdot m$ agent matrix in a $O(n \cdot m \cdot (n+m))$ space using constant number of controls. This algorithm first moves every agent into its target row in five moves, and then simultaneously rearranges every row of agents in six steps.

B. *Multi-purpose design*

We present a design of a $O(n \cdot m)$ space for rearranging arbitrary agent matrix that is $n \cdot m$ or smaller. We apply the Bubble Sort algorithm to rearrange an agent matrix. We show both shifting positions of agents and swapping between two agents at fixed positions can be done in a space of size linear to the number of agents. The combination of shifting and swapping allows bubble-sorting an agent matrix in a space that is a constant factor larger than the number of agents. The required sequence of controls is $O((n \cdot m)^2)$ in length.

Figure 1 shows a grid space with obstacles generated by our algorithm for rearranging an initially randomized Super Mario image (16×16 agent matrix).

II. Related work

Global control of robots. Becker *et al.* [4] examined the same model of particle swarms and proved it is NP-hard to decide whether a given initial configuration can be transformed into a desired target configuration, if the obstacles are fixed. Becker *et al.* designed an algorithm to construct AND and OR logic gates, when allowed to place obstacles, and also provided an algorithm to place obstacles in a $O(N^2)$ space to perform arbitrary permutations, where N is the number of robots. [5] later proved a stronger result: the problem of finding an optimal control sequence is PSPACE-complete. Shad *et al.* [22] extended the idea of building logic gates using unit-size swarm robots with global control into building a binary memory and showed it is possible to have nano-robots perform arbitrarily complex operations without using external computational devices.

Collecting objects in a compact manner has been studied using a similar model. Dhagat and O'Rourke [10] considered the problem of pushing square objects in a grid world. Mahadev *et al.* [16] provided algorithms to concentrate under-actuated swarm robots despite obstacles. Akitaya *et al.* [1] considered the problem of sweeping a line to compact a set of objects setting in a grid world, and showed that deciding whether the set of objects can be pushed to form a square is NP-hard.

Swarm robots. Over the past several years, researchers have built swarms of biologically-inspired small-scale robots; papers include [3], [15], [17], [12], [9]. Caprari *et al.* [8] built programmable ultra-low-power robots. Kornienko *et al.* [13] used the Jasmine micro-robots to study the re-embodiment of biological aggregation behavior of honeybees. Rubenstein *et al.* [18] built a thousand-robot swarm, Kilobot. Becker [6] later studied manipulation of large populations of simple robots with common input signals using kilobots. Rubenstein *et al.* [19] also used kilobots to study object transportation. Manipulation of droplets in lab-on-a-chip designs is an emerging application [23], and global controls may serve as one approach.

Robot construction and robot assembly. Seminal work on minimalist manipulation by Erdmann and Mason used tray tilting and interaction of a part with obstacles to coerce the part into a desired configuration [11]. Rubenstein *et al.* [20] designed an algorithm to control kilobots to self-assemble shapes in 2D. Kotay *et al.* [14] designed a robotic module, groups of which aggregate into 3D structures. Rus and Vona's early work [21] on the Crystalline robot presented an algorithm to do self-reconfiguration. Arbuckle *et al.* [2] allowed identical memory-less agents to construct and repair arbitrary shapes in the plane. In the authors' own work [24] on assembly of interlocking structures, nine kinds of blocks are used to build large-scale voxelized models such that all blocks are interlocked and the whole structure is rigid as a whole. Zhang *et al.* work [25] dissembles divisible material to extract an atomic part.

III. Rearranging a row of agents

In this section, we discuss the simplest case of rearranging a line of n horizontally connected unit-size agents in a grid world. Rearranging a vertical array of agents can be viewed as a 90-degree rotation of a horizontal array. These cases will later be used to solve more complicated cases when there are several rows of agents.

We present an algorithm for placing $2n$ obstacles to rearrange agents using a sequence of just four controls: *up*, *right*, *down*, *left*. We will show the space complexity of this design is $O(n^2)$. To better illustrate the process, we represent each agent using a ball labeled by a unique number from 1 to n; our goal is to sort these numbers in ascending order.

In this paper, the $(0, 0)$ position of a matrix is at the bottom left corner. Coordinate (x, y) means the position at the *x-th* column and *y-th* row.

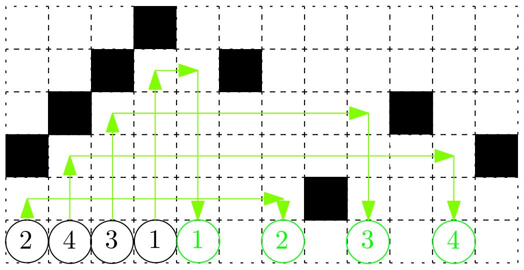


Fig. 2: Rearranging a row of n agents using $2n$ obstacles in a $(n + 2) \cdot (3n)$ space. Green spheres are in desired order.

A. The algorithm

The general idea of the approach is to first separate every agent into different rows using an *up* command, then move them to sorted relative positions using a *right* command and finally re-move them to a same row using a *down* command. See Figure 2.

Assume the first element of the array sits in the bottom-left corner ($(0, 0)$ position) of a grid space. If an agent is in the i-th position of the initial array and should be moved to the j-th position of the sorted array (indices start from 0), two obstacles will be placed, in the $(i, i+2)$ and $(n+2(j+1)-1, i+2)$ positions.

Although this algorithm deals only with a row of agents, it can easily be extended to rearrange a specific row or column of an agent matrix. In section V, we will see how an extension of this algorithm can be used to shift agent positions in a matrix.

B. Space complexity

In the above algorithm, separating n agents into different rows requires $(n+2)\cdot n$ space. Moving each ball to its correct position requires at most $(n+2)\cdot 2n$ space. So the total space is at most $(n+1)\cdot 3n$. Because we place two obstacles for each agent, the total number of obstacles is at most $2n$.

IV. Rearranging an agent matrix using eleven controls

We apply the technique mentioned in previous section to arrange agents in a square matrix. In this section, we present an algorithm that designs a grid space for a given $n\cdot m$ agent matrix and re-configure agent positions in a constant number of control commands. Figure 8 is an example of rearranging a 7×7 size heart image.

A. The algorithm

The idea is to take advantage of the fact that all row rearrangements require the same controls so we can rearrange rows simultaneously. Figure 3 is an example for a 2×3 agent matrix.

We describe the algorithm for placing obstacles in the grid space in two parts. Let the initial agent matrix be M, and the rearranged matrix be M'. (All indices start from 0.)

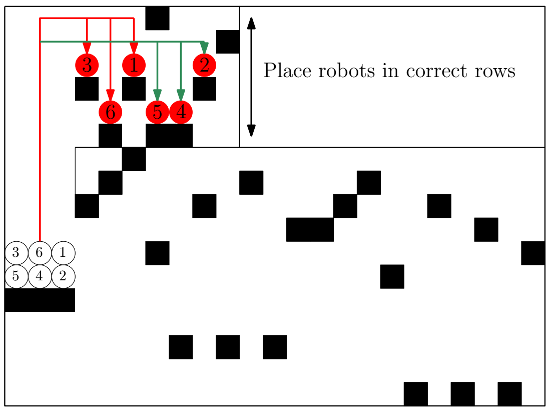
1) **Place agents in correct rows.** First place obstacles at $(m+m*i, 6n+m+2-i)$ for $i\in[1,n]$ to separate rows. Then for each agent, if at $M[x,y]$ and to be moved to $M'[x',y']$, place an obstacle at $(m+x, 3n+m+2y'+2)$. Thus every agent can be transported to its desired row. We denote the matrix after completing the first step as M''.
2) **Rearrange agent arrays.** We place m obstacles at position $(i,2n)$ for $i\in[0,m-1]$ to stop agents from the last *down* command. Similarly, we place obstacles to separate rows at $(2m+(3m+1)\cdot i, 3n-i)$ for $i\in[0,n-1]$. Then for each agent, if at $M''[x,y]$ and to be moved to $M'[x',y]$, place three obstacles at $(m+(3m+1)\cdot(n-1-y)+x, 3n+1+x+y)$, $(2m+(3m+1)\cdot(n-1-y)+2x'+2, 3n+y+(m-1-x'))$ and $(2m+(3m+1)(n-1-y)+2x'+1, 2y)$.

B. The complexity

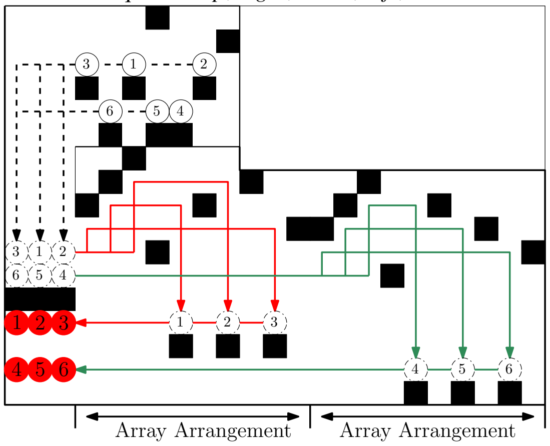
As we can see from the algorithm described above, the width of the board is $m+(3m+1)\cdot n$, and the height of the board is $6n+m+2$. The space complexity of the board is $O(m\cdot n\cdot(m+n))$. There are $2n+6m$ obstacles.

V. Rearranging an agent matrix in a $O(N)$ space

Although the previous section provides a fast algorithm to rearrange a given agent matrix, it requires a space that is relatively large and requires designing the grid of obstacles for the particular input. In this section, we present a complete algorithm to rearrange arbitrary agent matrices. Inspired by the Bubble Sort algorithm, we treat the matrix as a folded array, and sort the array by swapping adjacent agents. We show a design for swapping first, then show how to shift positions of agents so we only need to swap between two fixed positions of a matrix. Each grid space design is for $n\cdot m$ matrices but works for any smaller matrices. We combine these two parts and present a general grid space capable of rearranging arbitrary matrices smaller than $n\cdot m$.



(a) The first step moves every agent to its final using a 5-control sequence: *up*, *right*, *down*, *left*, *down*.



(b) With agents in their correct rows, we now rearrange all rows together using a 5-control sequence: *right*, *up*, *right*, *down*, *left*.

Fig. 3: A board design to rearrange a 2×3 matrix in constant steps. The rearrangement is done in two parts; one moves agents to correct rows, another rearranges all rows at the same time.

A. Swapping adjacent agents

We design a board to swap the last two agents in a row using 3 commands: *up*, *right*, *down*. The design uses the first command to separate two target agents and other agents. The second command (*right*) changes the relative positions of the target agents, and finally the last command puts them into the same row again. Figure 4 shows how swapping is done for a row of four agents.

The placement of obstacles for an array of n agents can be described as follow: 1. place obstacles at $(i,2)$ for $i\in[0,n-3]$, 2. place obstacles at $(n-2,3)$, $(n-1,4)$, $(2n-2,1)$, $(2n,3)$ and $(2n+1,2)$. The width of the grid space is $2n+2$, and the height is 5. A total number of $n+3$ obstacles are introduced into the space.

For a $m \cdot n$ matrix, if we only swap the last two elements in the first row, the design is similar, but we need to separate the first row from others; see Section V-C.

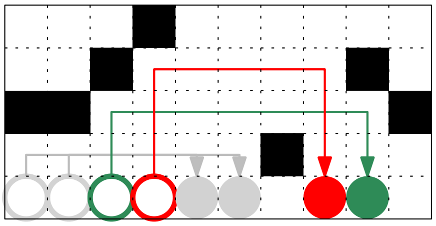


Fig. 4: Swapping the green and red agents. Other agents remain in the same order. This design requires $5 + (n - 1)$ obstacle and $5 \cdot (2n + 1)$ space, where n is the number of agents in the array.

B. Shifting agent positions

Because the swapping design only allows exchanging agents in two fixed adjacent positions (the last two elements of a row), to swap other adjacent agents, we need to first do a shift operation. As we treat an agent matrix as a folded array of N agents, shifting means rearranging agents $[a_0, a_1, a_2, \ldots, a_{n-1}]$ to $[a_1, a_2, a_3, \ldots, a_{n-1}, a_0]$. This operation can be done in four steps: 1. pop the first element of the first column, 2. push the element back to the end of the first column, 3. pop the first column 4. and push the column to the end. If a column of agents is treated as a whole agent, steps 3 and 4 can also be viewed as a rearrangement of a single line of agents. So our design is a combination of two agent line rearrangements. Figure 5 gives an example of shifting a 3×3 matrix.

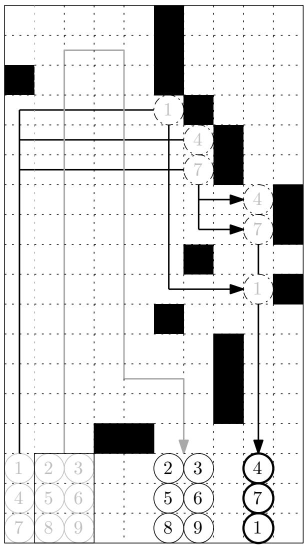


Fig. 5: Shifting all agents to predecessor positions. The first agent is shifted to the end. The shifting can be viewed as popping the first element of the first column and pushing it back to the end of the column, then popping and pushing the first column to the end.

We place obstacles in the following way:

1) One obstacle at at $(0, 4n + 3)$ to separate the first column from other columns.
2) $m - 1$ obstacles at $(m + i, n)$ for $i \in [0, m - 2]$ and n obstacles at $(2m - 1, 4n + 3 + i)$ for $i \in [0, n - 1]$ for elements not in the first column
3) Three obstacles at $(2m, 4n + 2)$, $(2m - 1, 2n + 1)$ and $(3m, 2n + 2)$ for the first element of the first column.
4) For other elements of the first column, put $n - 1$ obstacles at (3m - 2, 4n + 2 - i) for $i \in [1, n - 1]$, $n - 1$ obstacles at $(3m, 3n + 2 - j)$ for $j \in [1, n - 1]$ and one obstacle at $(3m - 3, 2n + 3)$.

In the example, the space is $3m + 1$ wide and $5n + 3$ tall, so the space complexity is also $O(n \cdot m)$. A total of $4n + m + 1$ obstacles are required.

C. The design of a $O(N)$ workspace

Finally, we combine two spaces: one for swapping and one for shifting. Figure 6 is an example for a 3×3 agent matrix. The left part of the space is a shifting room and the right part is a swapping room. The bottom of the right part is designed to constrain changes of relative positions of agents not in the first row. Although this board is generated for 3×3 matrices, it also works for smaller matrices. We can transform any matrix A into B using a Bubble Sort algorithm. We treat each matrix as a folded 1D array, the first element is the top-left element while the last is at bottom-right. Label each agent in matrix B as a sorted array in ascending order. If the last two elements of the first row are in descending order, swap, unless these two elements are the end and the head of matrix B. Otherwise, shift. The matrix is sorted if we have done $n \cdot m$ times of consecutive shifts.

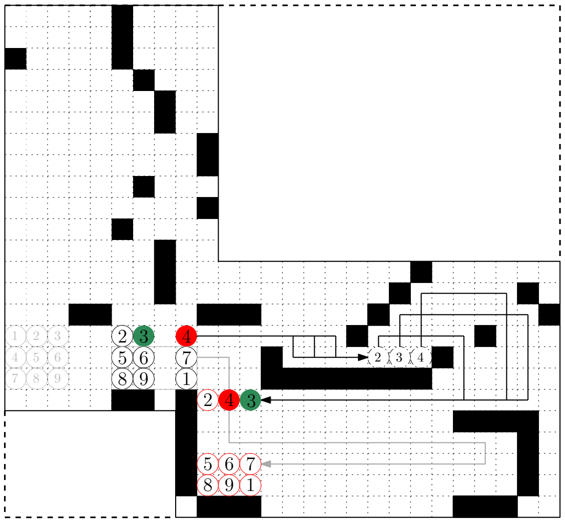
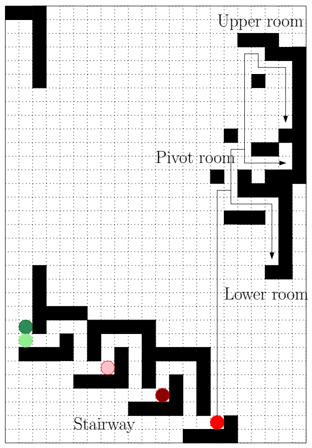
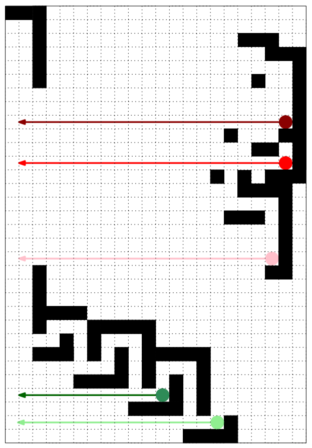


Fig. 6: The design of a $O(n \cdot m)$ size board for rearranging arbitrary agent matrices smaller than $n \cdot m$. The board has two parts: the left part does shifting and the right part does swapping. Here the agent labeled 3 and 4 exchanged positions, others remain the same position in the matrix.



(a) The stairway ensures single-agent exit. At the exit of the stair way, an agent can be moved to go to pivot room, lower room or upper room.



(b) After re-arranging a segment, move all agents to the left, then the just rearranged segment will be pushed to the top of the row.

Fig. 7: A board design to perform quick sort for a row of agents. The board is able to perform pivoting for any sub-sequences of agents and shift them to the top of the row.

D. Time and space complexity

The shifting part of the space is $O(n)$ tall and the swapping part is $O(m)$ wide. Both parts have $O(n)$ obstacles. So the space complexity is $O(n{\cdot}m)$. We are using a Bubble Sort algorithm to transform a matrix, and Bubble Sort requires $O(N^2)$ number of swaps, where $N = mn$. Each swap is followed by a shift operation. To place an agent to its correct position, the algorithm has to do at most N shifts to position the agent to swap positions, and the worst case will do N swaps. So the total number of shifts is $N{\cdot}N{+}N{\cdot}(N{-}1)/2$. The total number of operations is $O(N^2)$, which is the time complexity. Thus, we have Theorem 1.

Theorem 1: Transforming a matrix A into matrix B can be done by a set of $O(n)$ obstacles in a $O(n \cdot m)$ grid space. The transformation requires at most $O((n \cdot m)^2)$ moves.

Simply holding $n \cdot m$ agents requires a $n \cdot m$-size grid; our grid design also uses $O(n \cdot m)$ space, achieving a tight bound.

E. Sorting a row of agents with fewer actions and $O(N)$ space

Our analysis and experiments (Table I) shows a relatively large number of operations to complete a matrix rearrangement, which is expected from the worst case $O(N^2)$ time complexity of Bubble Sort. Can we instead perform an operation like quicksort?

Figure 7a is an example of a design capable of doing *partition* operations of a quicksort.

In our design, a space has five parts: the left-most is *entrance* part where agents are kept in a row, the lowest eight rows is *stairway* which ensures only one agent is able to exit and be further rearranged. The right side of the space is partitioned into three parts: a *pivot room* in the middle that holds the pivot agent, an *upper room* which holds agents labeled larger than the pivot and a *lower room* that holds agents labeled smaller than the pivot. After exiting the stairway, an agent can be controlled to enter pivot room; if it is the pivot agent, or the upper room or lower room independently (Figure 7a). When a segment of the row is pivoted, move *left* to recompose all agents into a single row. This operation also puts the rearranged segment onto the top of the left-over agents (Figure 7b).

After the partition step of the quicksort algorithm, all numbers smaller than the pivot element will be placed into one side of the pivot, and others will be placed into the other side of the pivot.

The stairway design is fixed for any length row of agents. So its width and height are both fixed. The upper room, pivot room and upper room also have fixed width of five.

An interesting note is that selecting the median element of the partition to pivot around in each partitioning step avoids the worst-case behavior of quick-sort. From a computational perspective, finding the median is itself an expensive operation, so the pivot is typically selected randomly or from one side of the sub-array to be partitioned. However, in our case, we are not particularly concerned with computational time needed to generate the sequence of action, but rather with the physical number of moves. Thus, it is quite reasonable to select the median for pivoting, allowing the modified quick-sort to use something like $\Theta(n \log n)$ moves.

Example: Assume we are given an array [8,7,6,5,4,3,2,1]. First step chooses 4 as pivot and partitions the array into two parts: [3,2,1],[4],[8,7,6,5]. Shift the lower part and pivot to the end: [8,7,6,5],[3,2,1,4]. Now partition [8,7,6,5] with pivot 6, yielding: [5],[6],[8,7],[3,2,1,4]. Shift the lower part and pivot: [8,7],[3,2,1,4],[5],[6]. Partition [8,7] with pivot 7: [7],[8],[3,2,1,4],[5],[6]. Shift the lower part and pivot: [3,2,1,4],[5],[6],[7],[8]. Partition [3,2,1,4],[5],[6],[7],[8] with pivot 2: [1],[2],[3,4],[5],[6],[7],[8]. The array is sorted!

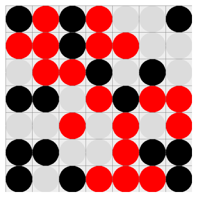
Although we have described the partitioning step, which is the core of quick-sort, quick-sorting rectangular arrays of agents is left for future work.

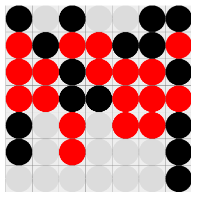
VI. Examples and Results

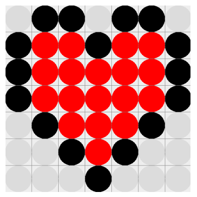
We ran several experiments with our design of constant move space and Bubble Sort space. We tested different cases of different size matrices including a heart image, a duck image, and a Super Mario image.

A. Constant-move grid design

Figure 8 shows an example of rearranging a heart image (7×7 size) using a 51×161 grid. In the initial image (Figure 8a), agent positions are randomized. After running the first 5 commands, every agent is moved into its target row, as shown in Figure 8b. The next 5 commands rearrange all 7 rows simultaneously, and the last command recomposes all rows into a matrix. Figure 8c is the goal configuration of all agents, and Figure 8d demonstrates the board generated by our algorithm based on the randomized agent positions.


(a) A randomized arrangement of a heart image.


(b) Agents are placed in target rows after 5 moves.


(c) The target rearrangement of the heart image.

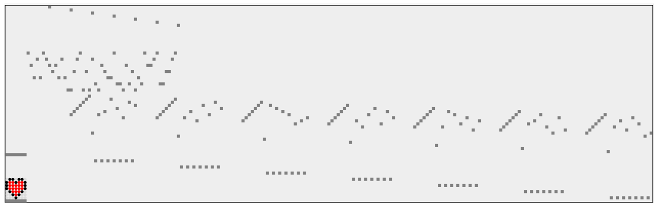

(d) The space designed for rearranging the 7×7 heart image. Dark gray squares are obstacles

Fig. 8: Rearranging a heart image (7×7) in a grid space in eleven moves. Initially, agent positions are randomized in Figure 8a. After 5 moves, all agents are moved to their target positions, Figure 8b. Figure 8c shows the final result of a rearranged heart image.

B. Bubble Sort grid

We ran our algorithm to generate $O(N)$ grid space and bubble-sort matrices of 3×3, 6×6, 9×9, 13×13 (duck image) and 16×16 (Super Mario image). For each matrix, we randomize its initial arrangement for 10 times and do bubble-sorting for each random arrangement. The following chart (Figure 9) and table (Table I) shows the average number of shifts, swaps, and moves, and their standard deviations.

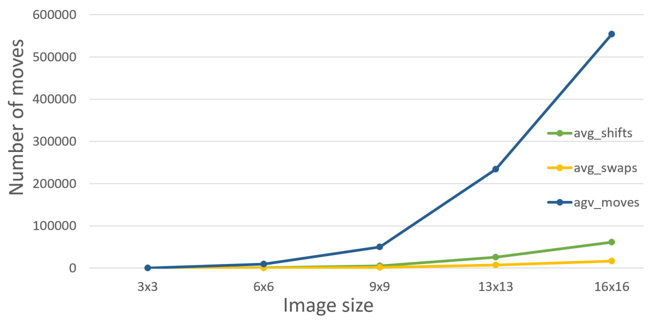
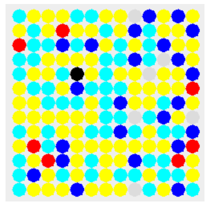


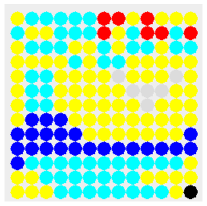
Fig. 9: Average number of shifts, swaps and moves for rearranging 3×3, 6×6, 9×9, 13×13 (duck image) and 16×16 matrices.

Figure 10 and 1 shows the process of rearranging an duck image (13×13) and a Super Mario image (16×16). They both use the same 100×119 grid space generated for 16×16 agent matrices. When 66% of commands are executed, the last several rows of both images are sorted and we are able to see the shape of the final image.

TABLE I: Bubble sort experiment results with different sized matrices.

	shifts	stdev	swaps	stdev	moves	stdev
3×3	38.7	15.326	13.7	5.697	382.9	149.674
6×6	1047.6	93.646	316.7	33.869	9768.3	778.590
9×9	5508	402.291	1555.1	62.836	50154.1	2388.549
13×13	25941.5	1403.254	7129.4	482.695	234072.4	11498.934
16×16	61593.6	2023.678	16809.8	635.403	554469.4	14164.014


(a) A randomized arrangement of duck image.


(b) Duck image rearrangement 66% done.

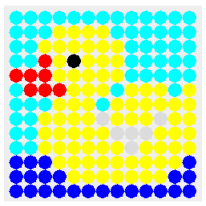

(c) The target rearrangement of duck image.

Fig. 10: Rearranging a duck image (13×13) in a $O(N)$ size board using Bubble Sort with limited global controls.

VII. Conclusions and Future Work

This paper studies the time complexity and space complexity of rearranging any 2D agent matrices using limited global control signals. It is possible to rearrange arbitrary $n \cdot m$ agent matrices in a $O(n \cdot m)$ space, which is linear to the space requirement to simply hold all the agent units. We also presented a complete algorithm to place obstacles in the space and to control agents to finish rearrangement. Realizing the large number of controls required using minimum required space, we showed an algorithm to manipulate a given agent matrix in constant steps but in a $O(n \cdot m \cdot (n + m))$ space. Our work can be viewed as a further discussion of the reconfiguring problems raised in [4].

In the future, we hope to extend our work from simple 2D rectangular agent matrices to arbitrary shapes. Of even greater interest is whether rearrangement can be done quickly. There are two approaches to speeding up rearrangement: 1) more efficient serial algorithms for mechanical sorting, and 2) parallel approaches that exploit the simultaneous nature of the force-field actions. Our work on quick-sort shows a promising approach to finding more efficient serial algorithms for rearrangement, but we are particularly interested in designing parallel algorithms.

References

[1] Hugo Akitaya, Greg Aloupis, Maarten Löffler, and Anika Rounds. Trash compaction.

[2] DJ Arbuckle and A AG Requicha. Self-assembly and self-repair of arbitrary shapes by a swarm of reactive robots: algorithms and simulations. *Autonomous Robots*, 28(2):197–211, 2010.

[3] Andrew T Baisch and Robert J Wood. Pop-up assembly of a quadrupedal ambulatory microrobot. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1518–1524. IEEE, 2013.

[4] Aaron Becker, Erik D Demaine, Sándor P Fekete, Golnaz Habibi, and James McLurkin. Reconfiguring massive particle swarms with limited, global control. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, pages 51–66. Springer, 2013.

[5] Aaron Becker, Erik D Demaine, Sándor P Fekete, and James McLurkin. Particle computation: Designing worlds to control robot swarms with only global signals. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 6751–6756. IEEE, 2014.

[6] Aaron Becker, Golnaz Habibi, Justin Werfel, Michael Rubenstein, and James McLurkin. Massive uniform manipulation: Controlling large populations of simple robots with a common input signal. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 520–527. IEEE, 2013.

[7] Aaron T Becker, Erik D Demaine, Sándor P Fekete, Hamed Mohtasham Shad, and Rose Morris-Wright. Tilt: The video-designing worlds to control robot swarms with only global signals. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 34. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[8] Gilles Caprari, Patrick Balmer, Ralph Piguet, and Roland Siegwart. The autonomous micro robot" alice": a platform for scientific and commercial applications. In *Micromechatronics and Human Science, 1998. MHS'98. Proceedings of the 1998 International Symposium on*, pages 231–235. IEEE, 1998.

[9] Yufeng Chen, E Farrell Helbling, Nick Gravish, Kevin Ma, and Robert J Wood. Hybrid aerial and aquatic locomotion in an at-scale robotic insect. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 331–338. IEEE, 2015.

[10] Arundhati Dhagat and Joseph ORourke. *Motion planning amidst movable square blocks*. PhD thesis, Smith College, Northampton, Mass., 1992.

[11] Michael A. Erdmann and Matthew T. Mason. An exploration of sensorless manipulation. *IEEE Journal of Robotics and Automation*, 4(4):369–379, 1988.

[12] Michael Karpelson, Benjamin H Waters, Benjamin Goldberg, Brody Mahoney, Onur Ozcan, Andrew Baisch, Pierre-Marie Meyitang, Joshua R Smith, and Robert J Wood. A wirelessly powered, biologically inspired ambulatory microrobot. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 2384–2391. IEEE, 2014.

[13] S Kornienko, R Thenius, O Kornienko, and T Schmickl. Reembodiment of honeybee aggregation behavior in artificial microrobotic system. *Adaptive Behavior (accepted for publication)*.

[14] Keith Kotay, Daniela Rus, Marsette Vona, and Craig McGray. The self-reconfiguring robotic molecule. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 1, pages 424–431. IEEE, 1998.

[15] Kevin Y Ma, Pakpong Chirarattananon, Sawyer B Fuller, and Robert J Wood. Controlled flight of a biologically inspired, insect-scale robot. *Science*, 340(6132):603–607, 2013.

[16] Arun V Mahadev, Dominik Krupke, Jan-Marc Reinhardt, Sándor P Fekete, and Aaron T Becker. Collecting a swarm in a grid environment using shared, global inputs. In *Automation Science and Engineering (CASE), 2016 IEEE International Conference on*, pages 1231–1236. IEEE, 2016.

[17] Cagdas D Onal, Robert J Wood, and Daniela Rus. An origami-inspired approach to worm robots. *IEEE/ASME Transactions on Mechatronics*, 18(2):430–438, 2013.

[18] Michael Rubenstein, Christian Ahler, and Radhika Nagpal. Kilobot: A low cost scalable robot system for collective behaviors. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3293–3298. IEEE, 2012.

[19] Michael Rubenstein, Adrian Cabrera, Justin Werfel, Golnaz Habibi, James McLurkin, and Radhika Nagpal. Collective transport of complex objects by simple robots: theory and experiments. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 47–54. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[20] Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014.

[21] Daniela Rus and Marsette Vona. Crystalline robots: Self-reconfiguration with compressible unit modules. *Autonomous Robots*, 10(1):107–124, 2001.

[22] Hamed Mohtasham Shad, Rose Morris-Wright, Erik D Demaine, Sándor P Fekete, and Aaron T Becker. Particle computation: Device fan-out and binary memory. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 5384–5389. IEEE, 2015.

[23] V. Shekar, M. Campbell, and S. Akella. Towards automated optoelectrowetting on dielectric devices for multi-axis droplet manipulation. pages 1431–1437, Karlsruhe, Germany, May 2013.

[24] Yinan Zhang and Devin Balkcom. Interlocking structure assembly with voxels. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 2173–2180. IEEE, 2016.

[25] Yinan Zhang, Emily Whiting, and Devin Balkcom. Assembling and disassembling planar structures with divisible and atomic components. In *Algorithmic Foundations of Robotics X - Proceedings of the Workshop on the Algorithmic Foundations of Robotics, WAFR, San Francisco, USA*, 2016.