Proceedings of the
13th International Conference on Applied Informatics
Eger, Hungary, February 19–21, 2026
DOI: 10.17048/icai.2026.182
pp. 182–191

Experimental Study of Building a Nest by an Automaton

Tamás Lukovszki, Gábor Pusztai

Eötvös Loránd University (ELTE),
Faculty of Informatics,
Hungary lukovszki@inf.elte.hu
pusztaigabor@inf.elte.hu

Abstract. Building upon the theoretical framework established by Czyzowicz et al. [3, 4], this study investigates the problem of constructing a *nest* from a set of bricks arranged in a two-dimensional grid by a robot modeled as a deterministic finite automaton. The subgraph of the grid induced by cells containing bricks is called the shape, which is initially connected. The *span* of the shape is defined as the Manhattan distance between its furthest cells. The robot starts at a cell with a brick. At each step, it can pick up a brick from its current cell if the current cell contains a brick, move to an adjacent cell, and drop a brick at an empty cell. The task of the robot is to reorganize the initial shape into the most compact shape – called the nest – with minimum possible span. In [3, 4] a finite automaton has been designed that accomplishes this task in $O(s \cdot n)$ time, where s is the span of the initial shape and n is the number of bricks. This running time is asymptotically optimal in the worst case. However, the original work omits some implementation details, and a straight-forward implementation results in inefficient runtime.

We present two algorithmic refinements of the original procedures and evaluate their performance using our interactive Java-based environment, the *NestBuilder simulator*, which facilitates visualization and enables comprehensive evaluation. Experimental results show a significant reduction in movement overhead while preserving the theoretical worst-case guarantees of the original algorithm.

Keywords: Mobile robots, Nest building, Finite automaton

2020 *Mathematics Subject Classification:* 68T40, 68U05, 68M14, 68W15

1. Introduction

The study of *mobile autonomous robot* with limited sensing and memory represents a rapidly advancing field in distributed systems, particularly regarding the design of effective algorithms that function without centralized control, prior environmental knowledge, or extensive data storage, leveraging local sensory data for decision-making. Such decentralized system can provide significant advantages in scalability, cost-effectiveness, and fault tolerance, making them indispensable for the robust execution of *construction tasks* where a simple agents must systematically gather and reorganize passive objects into predefined or compact shape geometries under strict localized sensing and memory constraints. A core requirement for these algorithms is the rigorous preservation of structural connectivity throughout the entire process. The practical implications of such tasks are vast, ranging from operations in unfavorable or inaccessible environments like in microscopic environments [1, 10], industrial logistics for hazardous material decontamination, and autonomous outer space assembly [2, 11]. Prior research has extensively investigated the reconfiguration of connected initial configurations on a grid using robots modeled as deterministic finite automata in [5–9, 12].

Czyzowicz et al. [4] showed that a deterministic finite automaton can organize an initially connected shape of bricks in the two-dimensional grid into a compact shape, called a nest, in $O(s \cdot n)$ time, where s is the span of the initial shape and n is the number of bricks, i.e. the Manhattan distance between the furthest cells in the shape. They also showed that this running time is asymptotically worst-case optimal. Despite the proven asymptotic optimality, there is a significant gap between the theoretical bound and the empirical performance. Many implementation details were omitted in the original work for simplicity, and a straight-forward implementation results in inefficient runtime.

This paper addresses these challenges by introducing the following:

- The *NestBuilder simulator*, an interactive Java-based environment for visualization and running rigorous evaluations, and additionally a Unity-based graphical player dedicated to visualization only.
- We propose two refinements of the original algorithm.
  - First, the *Optimized Rough Disc Extension* method, which is an improvement of the original extension procedure, which utilizes a localized geometric property to identify the next placement position of a new brick on the boundary of the shape.
  - Second, the *Distance-Aware Sweep* method, which is an improvement of the original sweeping procedure that separates the constructed nest from other bricks. Instead of checking $O(\sqrt{n})$ cells of the entire boundary of the nest after adding a new brick, our method only needs to check $O(1)$ cells.

- Experimental evaluation across 100 randomly generated initial configurations indicates that these enhancements yield an improvement of 18.75% in expansion and a 92% in sweeping procedures.

2. Model and Algorithmic Framework

Adopting the formal model and terminology by Czyzowicz et al. [4], the problem is defined within an infinite oriented $\mathbb{Z}\times\mathbb{Z}$ grid, represented as a set of unit square cells in the two-dimensional plane, with all cell sides vertical or horizontal. Each cell has 4 adjacent cells, North, East, South, and West of it. A cell is either *full*, i.e. contains a single brick, or *empty*. For any two cells $c = (x, y)$ and $c' = (x', y')$, the distance $d(c, c')$ is defined as the Manhattan distance (L_1 distance) $d(c, c') = |x-x'|+|y-y'|$. The subgraph of the grid induced by full cells is initially connected. The subgraph induced by the full cells is called the *current shape*. The Manhattan distance between the furthest cells of the shape is called its *span* s.

We are given a mobile robot starting in an arbitrary cell of the initial shape. The robot has no knowledge about the shape, of its size or its span. At each step, it can pick up a brick from its current cell (if it does not carry any brick at this time), move to an adjacent cell, and it can drop a brick at the current cell if it carries a brick and the cell is empty. The robot's objective is to rearrange the initial shape into a *nest*. The nest is a *disc* or a *rough disc*, see subsection 3.2 for details. A *nest* of size n is a shape that has the minimum span among all shapes of size n. While the grid is theoretically infinite, the operational area of the robot is limited by the initial configuration. Given that the algorithm operates through searching for a suitable brick and extending the nest with that brick, the robot remains within a limited distance of the initial shape.

In [4], the robot knows whether each cell at distance at most $r = 8$ from its current cell is full or empty. During the nest building, the robot maintains a special, one-cell component M, called the *marker*. The robot also maintains a *swept region* around the nest. This region is a gap where all cells within a distance of 7 from the nest are empty except for the cell of the marker. The marker signals the robot that it got back in the vicinity of the rough disc. Using the marker allows the robot to avoid storing the nest's coordinates. During the execution of the nest building algorithm, the robot does not ensure that the full cells outside of the rough disc and of the marker form one component. Any component that is different from the rough disc and from the marker is called a *free component*. The robot must identify and retrieve a *free brick*, which is defined as a brick of a free component whose removal does not disconnect that component.

The robot is formally modeled as a deterministic automaton. It may remember a constant number of bits by encoding them in its states (see [4] for a detailed description). The high-level idea of the nest building algorithm is the following.

First, the robot performs some preprocessing by establishing the marker, the initial rough disc, and the swept region. Then it performs the main loop. In each iteration, the robot performs three procedures:

1. FindNextBrick: Finding the next free brick b in a free component. This procedure is not discussed in this article; see [4] for details.

2. ReturnToMarker: Returns to the marker with b. See [4] for details.

3. ExtendRoughDisc: Extending the rough disc with b and by calling Sweep, restoring the property that there are no full cells within a Manhattan distance of 7 from the rough disc, except for the marker. See subsection 3.2 and subsection 3.3 for details and improvements.

The authors show that this algorithm can be implemented with a finite-state robot that builds a nest in $O(s \cdot n)$ time steps.

3. Implementation and Practical Refinements

While the original algorithm proposed by Czyzowicz et al. [4] provides a worst-case optimal complexity of $O(s \cdot n)$, the authors noted that many specific implementation details were omitted for the sake of simplicity. Our work bridges this gap by developing a functional simulator and introducing refinements that improve the performance without sacrificing the theoretical guarantees of the finite-state automaton model.

3.1. The NestBuilder Simulator

To evaluate the construction algorithm, we developed the *NestBuilder* simulator using the Java programming language. A primary challenge in simulating a robot navigating an infinite grid $\mathbb{Z} \times \mathbb{Z}$ is the management of memory and coordinates. Our environment manages this infinite grid through a dynamic $n \times n$ collection that expands elastically as the robot moves further away from the origin.

Furthermore, to ensure rigorous stress-testing of the robot's ability to recover connectivity, we utilize *Kruskal's* algorithm with randomly generated weights during the setup phase. This allows us to generate complex, randomized, yet initially connected starting shapes of size z, ensuring the robot must navigate a challenging configuration of bricks to gather all the bricks into a nest.

Beyond the core Java-based simulation engine, this research incorporates a supplementary visualization environment developed in Unity. While the Java simulator is primarily used to execute the algorithm and generate statistical data, the Unity environment serves as a specialized tool for visualising and replaying pre-recorded simulation sequences. The program is deployed via *WebGL*, which provides an extra perspective on the robot's movement and the creation of the nest. The architecture of this visualization tool is built upon a modular simulation framework previously developed and published by us [13], which was specifically designed for modeling autonomous agent behaviors and construction tasks in grid-based environments.

3.2. Optimized Rough Disc Extension

A *disc* D of radius $r \geq 0$ centered at c is defined as the set of cells whose distance from c is at most r. The set of cells of D that are at a distance of i, $0 \leq i \leq r$ from c is called the i-th *layer* of D. A disc of radius $r \geq 0$ contains $n_r = 1 + 4 \cdot \sum_{i=1}^{r} i = 2r^2 + 2r + 1$ cells and has a span $2r$. When the total number of bricks n satisfies $n_r < n < n_{r+1}$, the structure is classified as a *rough disc*. This structure is formed by extending D with a set of cells F of $n - n_r$ bricks, which are placed in the set of cells adjacent to D starting at the northern neighbor of the easternmost cell and proceeding counterclockwise. Let R be a connected shape formed by the full cells. The *boundary* ∂R of R is defined as the set of cells of R that are adjacent to at least one empty cell.

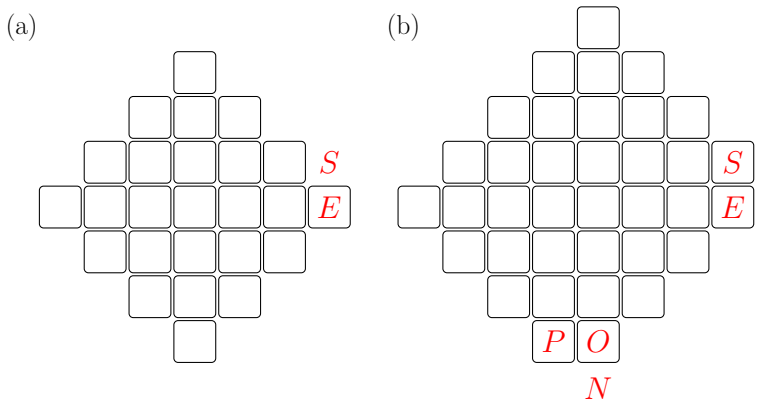


Figure 1. Visual representation of the localized geometric heuristic used to identify the target placement position for new bricks. (*a*) Disc: The robot identifies the easternmost cell (E) and checks its northern neighbor (S); if S is empty, the nest is recognized as a completed disc, and a new layer begins at S. (*b*) Rough Disc: For an incomplete layer, the robot identifies the connected pair of bricks P (the previously placed brick) and O (the brick to which the new one will connect) on the boundary to locate the target placement position (N).

In [4], in the original ExtendRoughDisc procedure, the robot remembers in its state whether the current nest is a disc or a rough disc. Adding a brick to a disc, it walks to the starting position S of a new layer, which is defined as the northern neighbor of the easternmost cell E of the nest. For a rough disc, it is required to perform a traversal along the boundary ∂R to find the correct location for the brick. The time required for this operation is the size of the boundary of a rough disc, which is $O(\sqrt{n})$, which is $O(s)$.

The original work provides limited specification regarding the precise localization of the placement position. We provide a method to identify this position based on a simple geometric property of the boundary of the nest. During a counterclock-

wise traversal of the nest, the robot evaluates the connectivity of the bricks on the boundary ∂R. This approach makes it unnecessary to remember whether the nest is a disc or a rough disc.

- When the robot reaches the easternmost cell E of the nest, determinable during counterclockwise traversal, when the next ∂R brick would have increased y but non-increased x coordinate. The robot checks the northern neighbor cell S of E. If S is an empty cell, then the robot recognizes the nest as a disc and starts a new layer at the position S.

- Otherwise, the nest is a rough disc. Then there are exactly two positions where bricks of ∂R are connected to each other (the cells share a common side). One occurs at the starting location S, and its southern neighbor E. The other is at P (the previously placed brick of set ∂R) and its neighbor O. Let N be the empty cell (where the next brick will be placed) adjacent to O, such that N is on the right side of $\overrightarrow{PO}$. Note that $P = S$ or $O = E$ are possible. If $P = S$ then we started a new layer with the previous brick. If $O = E$ then we complete a disc layer after placing the brick at N. The pairs E, S and P, O of bricks on the boundary of the nest are identified during the boundary traversal by moving on top of the bricks. When the robot encounters a connected pair, it checks whether it corresponds to the starting position S, E. If so, the robot continues traversing until it finds the pair P, O. Otherwise, the pair P, O is localized.

3.3. Distance-Aware Sweeping

The Sweep procedure maintains the gap invariant: it ensures that all bricks belonging to a free component C are at a Manhattan distance of at least 7 from the rough disc D. In [4], the original algorithm maintains the gap by requiring the robot to perform a full counterclockwise traversal of the boundary of the nest after every brick placement, checking every cell $c \notin D \cup M$ (M is the marker) within a radius of 7 from its current position.

Our optimization is based on the geometric properties of the free components. Let D be the current nest, which is a rough disc. For D, the region D_{+7} is defined by a Manhattan distance of ≤ 7 from D. $D_{+7} \setminus D$ forms a swept region, which must not contain any bricks. Upon extending the nest by placing a new brick at the cell b, new cells will be included in the swept region, which might contain bricks that have to be relocated. Mathematically, if the nest is extended to $D' = D \cup \{b\}$, then the new distance from any cell c to the nest is defined as $d(c, D') = \min(d(c, D), d(c, b))$. This implies that the invariant $d(c, D') > 7$, for any brick $c \notin D'$, can only be violated if $d(c, b) \leq 7$. All such bricks are visible from b. This means that it is sufficient to check a constant number of cells around b instead of traversing the whole boundary for maintaining the invariant. This significantly reduces the time for each sweep operation.

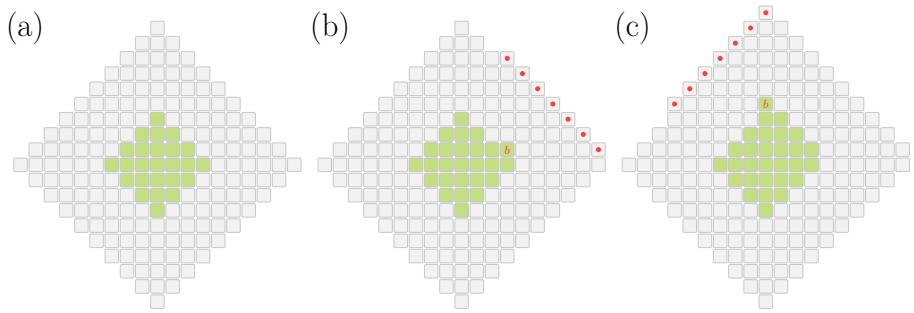


Figure 2. Illustration of the optimized sweeping process, which maintains the gap around nest D, such that $d(c, D) > 7$ for any brick $c \notin D$. (a) The initial nest is surrounded by a swept region where no bricks are permitted. (b) and (c) The nest is extended by placing a new brick at the cell b. Only a constant number of cells (indicated by dots) need to be checked to see whether they contain bricks.

When a new brick is placed at b, only cells c with $d(c, b) = 7$ need to be checked to see if they contain a brick, since any brick which is not in D is of a distance at least 8 from D before extending the nest with b.

4. Experimental Evaluation and Conclusion

To empirically validate the proposed optimizations, we conducted a series of systematic tests using the *NestBuilder* simulator across a diverse range of initially connected fields. The experimental setup involved 100 randomly generated connected configurations, where starting shapes were generated via Kruskal's algorithm with randomized weights to facilitate a rigorous stress test of the ability of the robot to navigate complex configurations while maintaining connectivity. To ensure that our results are independent of the specific shape and scale of the initial configuration, we evaluated several types of initial shapes (tree, line, spiral) within a grid of three sizes: 10, 20, and 40. The total number of bricks varied depending on the types of the shapes. For sparse configurations, such as tree and line, the number of bricks ranged from 21 to 47 for the 10×10 grid, from 48 to 95 for the 20×20 grid, and from 111 to 165 for the 40×40 grid. The dense spiral (more precisely, an Archimedean spiral according to the L_∞ metric, consisting of vertical and horizontal line segments, with a line width of 1 and a gap between the lines of 1) contained 55, 231, and 861 bricks in the corresponding grids. The data presented reflect the average performance of all experimental runs.

4.1. Evaluation of Optimized Rough Disc Extension

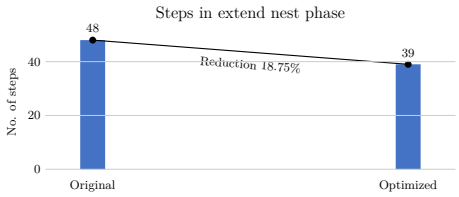
The algorithm described in subsection 3.2 was tested against the original boundary-traversal method in [4].

Our method achieved an improvement of 18.75% in the number of steps during the nest extension phase, see Figure 3a. Specifically, the average number of movements in the ExtendRoughDisc procedure was reduced from an average of 48 to an average 39 steps.

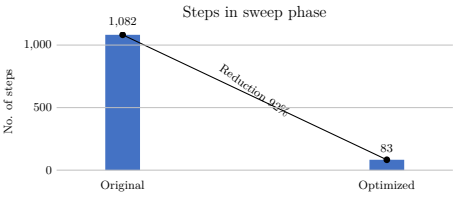
4.2. Evaluation of Distance-Aware Sweeping

The Distance-Aware Sweeping method provided the most substantial performance improvement by utilizing the geometric property that the swept region around the nest D is only affected by the cell b, where the new brick is placed. While the original algorithm in [4] required a full boundary traversal of the nest D, which takes $O(n)$ steps, and checking the cells within a distance of 7 cells in every step of the boundary traversal, our method only needs to check $O(1)$ cells around the new brick placed at b.

This improvement resulted in a massive 92% reduction in the number of steps, see Figure 3b. In the sweep phase, the average number of movement steps decreased from an average of 1082 steps to an average of 83. This result shows that our approach effectively eliminates redundant examination of cells that could not have changed since the last iteration.



(a) Comparative analysis of movement efficiency between the original traversal method and our Rough Disc Extension procedure. Our approach achieves an 18.75% reduction in the number of movements required for the ExtendRoughDisc procedure (from an average of 48 movements to an average 39).



(b) Comparative analysis of movement efficiency of the original sweeping method and the Distance-Aware Sweeping method. Our approach achieves a 92% reduction in the number of steps required during *Sweep* procedure (from an average 1082 steps to 83.)

Figure 3. Experimental Evaluation.

5. Conclusion and Future Work

This work presents empirical practical improvements of the algorithm by Czyzowicz et al. [4] for the construction of a nest by an autonomous robot modeled as a finite

automaton. The original algorithm has a running time of $O(s \cdot n)$, which has been shown to be asymptotically optimal for this model. Through the development of the *NestBuilder* simulator and modifications of the original algorithm, we have demonstrated that significant improvements can be achieved in runtime without sacrificing the memory requirements of a deterministic finite automaton.

Future research:

- **Target Structure Diversity:** We aim to characterize which other classes of target shapes can be built starting from any connected shape.
- **Parallelization Challenges:** The model is highly effective for single-robot nest construction, but has a high limitation for the marker-based system for multi-robot coordination. It is a challenging task to develop algorithms for multiple robots that enable collaborative nest building without destroying the connectivity of the free components.

References

[1] A. T. Becker, S. P. Fekete, P. Keldenich, D. Krupke, C. Rieck, C. Scheffer, A. Schmidt: *Tilt Assembly: Algorithms for Micro-factories That Build Objects with Uniform External Forces*, Algorithmica 82.2 (2020), pp. 165–187, doi: 10.1007/S00453-018-0483-9.

[2] M. K. Ben-Larbi, K. F. Pozo, T. Haylok, M. Choi, B. Grzesik, A. Haas, D. Krupke, H. Konstanski, V. Schaus, S. P. Fekete, C. Schurig, E. Stoll: *Towards the automated operations of large distributed satellite systems. Part 1: Review and paradigm shifts*, Advances in Space Research 67.11 (2021), pp. 3598–3619, doi: 10.1016/j.asr.2020.08.009.

[3] J. Czyzowicz, D. Dereniowski, A. Pelc: *Building a Nest by an Automaton*, in: 27th Annual European Symposium on Algorithms, ESA 2019, 2019, 35:1–35:14, doi: 10.4230/LIPIcs.ESA.2019.35.

[4] J. Czyzowicz, D. Dereniowski, A. Pelc: *Building a Nest by an Automaton*, Algorithmica 83.1 (2021), pp. 144–176, doi: 10.1007/S00453-020-00752-0.

[5] S. P. Fekete, E. Niehs, C. Scheffer, A. Schmidt: *Connected assembly and reconfiguration by finite automata*, arXiv preprint arXiv:1909.03880 (2019), doi: 10.1007/s00453-022-00995-z.

[6] S. P. Fekete, R. Gmyr, S. Hugo, P. Keldenich, C. Scheffer, A. Schmidt: *CADbots: Algorithmic Aspects of Manipulating Programmable Matter with Finite Automata*, Algorithmica 83.1 (2021), pp. 387–412, doi: 10.1007/S00453-020-00761-Z.

[7] S. P. Fekete, E. Niehs, C. Scheffer, A. Schmidt: *Connected Reconfiguration of Lattice-Based Cellular Structures by Finite-Memory Robots*, Algorithmica 84.10 (2022), pp. 2954–2986, doi: 10.1007/S00453-022-00995-Z.

[8] R. Gmyr, K. Hinnenthal, I. Kostitsyna, F. Kuhn, D. Rudolph, C. Scheideler: *Shape recognition by a finite automaton robot*, in: 43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2018, pp. 52–1, doi: 10.4230/LIPIcs.MFCS.2018.52.

[9] R. Gmyr, K. Hinnenthal, I. Kostitsyna, F. Kuhn, D. Rudolph, C. Scheideler, T. Strothmann: *Forming tile shapes with simple robots*, Natural Computing 19.2 (2020), pp. 375–390, doi: 10.1007/s11047-019-09774-2.

[10] W. Hu, G. Z. Lum, M. Mastrangeli, M. Sitti: *Small-scale soft-bodied robot with multimodal locomotion*, Nature 554.7690 (2018), pp. 81–85, DOI: 10.1038/nature25443.

[11] B. Jenett, C. Gregg, D. Cellucci, K. Cheung: *Design of multifunctional hierarchical space structures*, in: Aerospace Conference, IEEE, 2024, pp. 1–10, DOI: 10.1109/AERO.2017.7943913.

[12] E. Niehs, A. Schmidt, C. Scheffer, D. E. Biediger, M. Yannuzzi, B. Jenett, A. Abdel-Rahman, K. C. Cheung, A. T. Becker, S. P. Fekete: *Recognition and Reconfiguration of Lattice-Based Cellular Structures by Simple Robots*, in: International Conference on Robotics and Automation, (ICRA), IEEE, 2020, pp. 8252–8259, DOI: 10.1109/ICRA40945.2020.9196700.

[13] G. Pusztai, T. Lukovszki: *Multi-Robot Simulation Framework for Construction Problems*, in: Proceedings of the Intelligent Robotics FAIR 2025, IntRob '25, Association for Computing Machinery, 2025, pp. 124–131, ISBN: 9798400715891, DOI: 10.1145/3759355.3759631.