On Coordinated Motion Through a Maze

Joseph Friedman
Stanford University

John Hershberger
DEC Systems Research Center

Jack Snoeyink
Stanford University

Abstract

We look at problems of having m points that move in response to the same force (such as gravity) escape a polygonal maze of n line segments. We give an algorithm to determine the shortest sequence of tilt or rotation movements by which all the points can escape, if one exists. For a fixed number of points, m, our algorithm runs in polynomial time. If $m = \Theta(n)$, then an exponential number of moves may be necessary, and to determine if all points can escape is PSPACE-complete.

1 Introduction

Suppose we have a container that has one or more openings and has a few little objects, such as *marbles*, inside it. When we gently turn or tilt the container, the marbles inside it may move because of gravity. Our goal is to get all the marbles out through the openings in the container.

In this paper we consider this problem for a two-dimensional container whose walls are made of n line segments that intersect only at their endpoints. We call such a container a *maze* (see Figure 1). The maze contains m marbles, each of which is a single point.

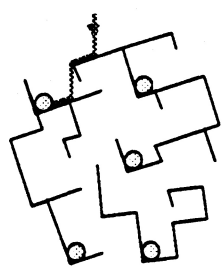

Figure 1: A maze

At any moment, the position of the marbles is determined by the *gravity vector*, which is an arbitrary direction α. The gravity vector can change according to rules specified below. Each marble responds to changes in α by moving through free space in direction α and rolling along walls to the walls' lower corners (with respect to α) until it reaches a convex corner which is a local minimum (see the top left marble in Figure 1). To remove ambiguity, when a marble hits a horizontal segment or an endpoint from which it could roll out on either of the adjacent segments, the marble always rolls to the right. Two marbles that get stuck at the same point do not interfere with each other—and since all marbles respond to the same forces, the two can be identified and treated as one. Once a marble has moved below all the segments making up the maze, we say that this marble has *escaped* the maze, and remove it from further consideration.

We consider the *rotation* paradigm in Section 2. Under this paradigm, the legal movements are continuous rotations of the maze either clockwise or counterclockwise; to change from gravity vector α to gravity vector β, one has to go through all the directions in between. We show an algorithm for determining a shortest sequence of movements by which all the marbles escape, if such a sequence exists. When the number of marbles m is fixed, our algorithm runs in time polynomial in n, the number of walls in the maze. When m is $\Omega(n)$, the algorithm requires a number of steps that is exponential in n, but uses polynomial space.

The *tilt* paradigm is a generalization of the rotation paradigm, in which the gravity vector α can change arbitrarily between moves. In Section 3 we show how to modify the rotation maze algorithm for tilt mazes.

In Section 4 we show that under both paradigms, the problem of deciding whether the marble maze has a solution is *PSPACE*-complete. Our proof is direct: for a given polynomial-space Turing machine M and a string x we construct a marble maze such that the movements in the maze simulate the computation of M given x as input.

A number of researchers have considered titling as a means of reorienting parts for robot assembly tasks. This approach was first proposed by Grossman and Blasgen [5], and has later been studied by Erdmann and Mason [3] and Natarajan [8]. Natarajan has showed that some variants of the parts orientation problem are *PSPACE*-complete.

The path taken by a single marble is the same as the path taken by a robot moving with perfect control under the compliant motion model [1, 2, 7].

Friedman, Hershberger, and Snoeyink [4] consider (among other things) robots that must stick at subgoals. Their methods can find a sequence of n tilts in $O(n^3 \log n)$ time and space by which a single marble can escape.

Canny and Reif [1] have shown that the problem of having marbles escape a maze in 3-space is NP-hard, even if a single marble is dropped into a stationary maze.

Research on multiple robots usually deals with independently-controlled robots that can interfere with each other(see, for example, [10, ch. 2&3]). Our algorithm for the marble-maze problem provides multi-step motion plans for independent compliant-motion robots that respond to the same motion commands (perhaps they share a single radio channel to their central computer).

2 The Rotation Paradigm

Recall that under the rotation paradigm we allow only continuous rotations of the maze. Equivalently, we allow rotation of the gravity vector α either clockwise (CW) or counter-clockwise (CCW). We assume that when marbles are moving, the rotation stops. In this section, we describe an algorithm that computes the shortest sequence of rotations required to get all the marbles out of the maze.

The algorithm traverses a *state graph*, in which each node captures some marble configuration inside the maze (a *state*). The state graph contains an arc from node u to node v if and only if the state corresponding to u changes to the state corresponding to v by a CW or a CCW rotation. The algorithm simply performs a shortest-path search from the node representing the initial marble configuration to the node representing the empty maze.

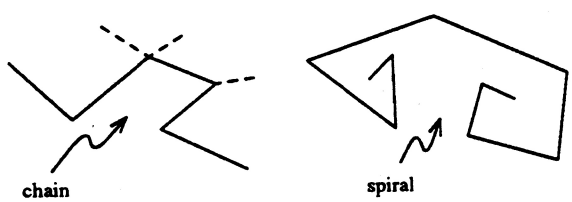

Figure 2: Two chains; one is a spiral

First, however, we notice that marbles stay in *spirals* in rotation mazes and that we can build marble traps. For example, no marble can escape from the spiral shown in Figure 2 because marbles that leave the chain return to the chain.

Let us now return briefly to the state graph. We partition the segments of S into $s \leq n$ spirals, then we describe the marble configuration in the maze by an s-bit binary vector that contains at most m 1's: when a bit is 1, the corresponding spiral contains a marble. The number of nodes in the state graph is

$$v = \sum_{i=0}^{m} \binom{s}{i} \leq \sum_{i=0}^{m} \binom{n}{i}.$$

When m is fixed, $v = O(n^m)$; when m is $\Omega(n)$, then $v = O(2^n)$.

A breadth-first search obtains the shortest sequence of rotations in $O(vm)$ time and space. Unfortunately, when $m = \Omega(n)$, this algorithm requires exponential space.

Alternatively, we can use an approach similar to the one used by Savitch [6, 9] and keep the space requirements down, at the cost of increasing the running time. When $m = \Omega(n)$ marbles, the space requirement is $O(n^2)$ and the running time is $2^{O(n^2)}$.

3 The Tilt Paradigm

Recall that in the tilt method we allow the direction of the gravity vector to change arbitrarily from one move to the next. In spite of the increased freedom, the maze may still contain marble traps, as illustrated in Figure 3. Once inside the trap, a marble can only reach other corners of the trap.

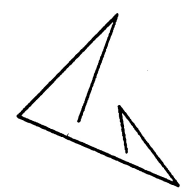

Figure 3: A tilt trap

The algorithm is basically the same as outlined in Section 2. However, it is not enough to partition the environment into spirals for the state graph, so we use all the convex corners of the maze instead. Precisely, if $s \leq n$ is the number of convex corners of the maze, the nodes in the state graph are s-bit binary vectors with at most m 1's, and a 1-bit corresponds to a marble in the appropriate corner. The number of nodes v is $O(n^m)$ for a fixed m and $O(2^n)$ for $m = \Omega(n)$, as before.

Let us determine the out degree of a node in the state graph.

Lemma 3.1 *For any marble at a point p in a maze with n segments, there are $O(n)$ intervals of directions such that a compliant motion in all directions in an interval ends at the same convex corner.*

We conclude that the out degree of a node in the state graph is $O(mn)$. We can compute all the neighbors of a particular node by merging the precomputed n ranges of each of the (at most) m convex corners containing marbles. The merge runs in $O(mn \log m)$ time. Therefore, the complexity of a breadth-first search algorithm for the tilt paradigm is $O(vmn \log m)$ time and $O(vm)$ space, and the complexity of a Savitch-style algorithm is $O(v^{\log v+1} \log v \cdot mn \log m)$ time and $O(\log^2 v + mn)$ space.

4 Simulating Computations

In this section we investigate the ability of marble mazes (under both paradigms) to simulate Turing machines. Throughout this section, let M be an arbitrary fixed Turing machine with the following properties:

- The alphabet of M is $\{B, 0, 1\}$. B is the blank symbol, and is not part of the language recognized by M (in other words, the input tape is always a finite contiguous sequence of 0's and 1's surrounded by B's);
- M uses a single one-way infinite tape, and starts the computation on the leftmost nonblank tape cell;
- There is a function $S(n) \geq n$ such that in any accepting computation of M on an input of length n, M scans at most $S(n)$ cells.[1]

The main result of this section is the following theorem. The theorem holds under both paradigms.

Theorem 4.1 *Using the above notation, for any input string x over $\{0,1\}^*$ we can construct a marble maze $L(x)$ such that all the marbles can escape $L(x)$ if and only if M accepts x. Our construction uses $O(\log S(n))$ space, and produces a marble maze with $O(S(n))$ edges and $O(S(n))$ marbles.*

Furthermore, if M accepts x in t steps, then the shortest solution of $L(x)$ uses $\Theta(t)$ movements.

By way of remark, the second part of this theorem applied to a simple machine that counts in binary implies an exponential lower bound on the number of movements needed to solve marble mazes. The first part and the previous sections imply the following corollary:

Corollary 4.2 *The problem of determining whether a marble maze L has a solution is PSPACE-complete.*

[1] We also require that $\log S(n)$ be space-constructible (see [6, p. 297]).

A Conceptual Overview of the Construction

Fix x, the input string. The segments making up the marble maze $L(x)$ depend only on the *length* of x (and on the transition table of the machine M). The initial placement of the marbles in $L(x)$ specifies the *contents* of x.

The maze $L(x)$ consists of a *trap* L-cell, followed by $S(n) - 1$ *basic* L-cells, followed by a *head* L-cell, an additional $S(n) - 1$ basic L-cells and another trap L-cell. The basic L-cells are all identical; the trap L-cells and the head L-cell are each as big as a basic L-cells, but have a different structure. (See Figure 4.) The L-cells correspond to the cells of M's tape. The head L-cell always corresponds to the cell that M's head scans, and therefore the correspondence between L-cells and tape cells keeps shifting right or left as M carries out its computation on x. The maze $L(x)$ contains a total of $S(n)$ marbles, and at any given moment these marbles occupy a chunk of $S(n)$ consecutive L-cells, one marble per L-cell.

The trap L-cells at both ends guard against "spill-overs" which can happen when a non-accepting computation attempts to consume more than $S(n)$ tape cells. In this case, a marble enters the trap cell and the maze has no solution. From now on, let us refer to a non-trap L-cell simply as an L-cell.

When the marbles are at rest, they are in special corners of the read or write halves, called *pockets*. The read half of every L-cell records the contents of the corresponding tape cell as well as the current state of the machine; in other words, the read half of every L-cell has a pocket for every pair (q, a) of state and alphabet symbol. The write half records, in addition to this, the symbol under M's head, namely, a pocket for every triple (q, h, a). We discuss the particular implementation of pockets in Sections 4 and 4. Figure 5 shows the pockets used in a simulation step—due to space contraints, we omit the description.

Let us conclude this conceptual overview with a look at the end and the beginning of the computation. The construction of $L(x)$ must allow the marbles to escape when M reaches an accepting state. We can achieve this by replacing the all pockets that correspond to symbols in accepting states with openings in the read halves of all the L-cells.

If the initial state of M is q_0, and the input string $x = a_0 \ldots a_{n-1}$, the initial marble configuration inside $L(x)$ has a marble in the pocket (q_0, a_i) of the read half of the ith L-cell to the right of the head L-cell, and a marble in the pocket (q_0, B) any the other L-cells to the right of the input.

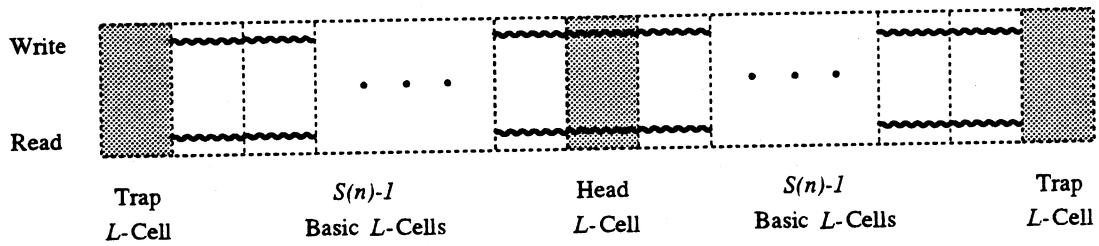

Figure 4: The conceptual structure of the maze $L(x)$

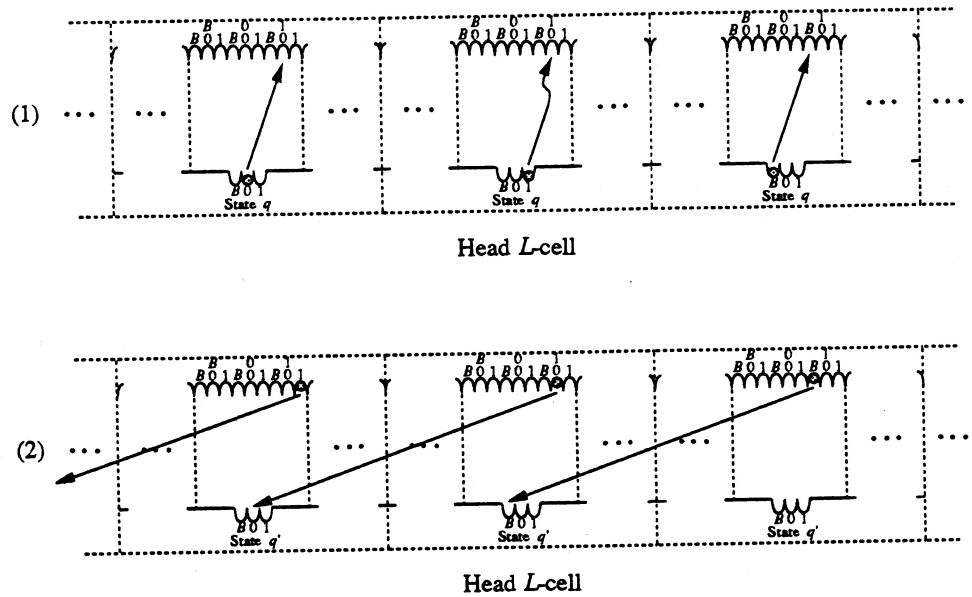

Figure 5: A simulation of a step of M.

Gadgets for the Tilt Paradigm

Let us concentrate on the tilt paradigm, and describe how to construct traps, pockets, and the head L-cell's near-traps. The trap cells have a tilt trap very much like the one shown in Figure 3 in both the read and write halves. In other L-cells, a pocket is simply a convex corner.

Figure 6 shows how to merge three pockets in the write half of the head L-cell so that they guide the marble into position (q, h, h'), as described above. In Figure 7 we show how to implement the near-traps on the pockets of the read half in the head L-cell. These near-traps force the marble in the pocket to go in a very narrow range of directions, and trap the marble for any other direction.

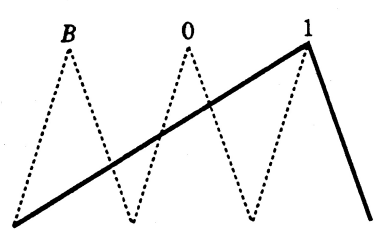

Figure 6: Redirecting the head marble

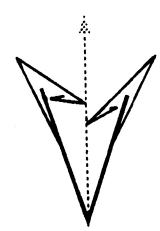

Figure 7: A near-trap

Gadgets for the Rotation Paradigm

The trap cells employ a trap similar to the spiral shown in Figure 2. The other gadgets for rotational mazes are a little bit more complicated to construct than their tilt counterparts. Figure 8 shows a pocket in the read half of a basic L-cell. The figure shows the rest position of the marble inside the pocket; it is easy to see that when the marble rolls down into the pocket through the entrance at the top, it ends up in this rest position. The marble can leave

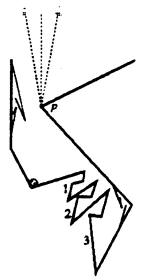

Figure 8: Rot. pocket

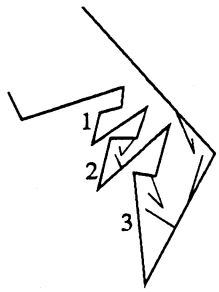

Figure 9: Spiral selector

the pocket using two rotations: first, a CCW rotation which causes the marble to fall into one of the three spirals marked 1, 2, and 3 in Figure 8; then, a CW rotation of nearly 180° causes the marble to leave the point marked p in one of three directions. In the head L-cell, we place traps on exactly two of the spirals 1, 2, and 3, as illustrated in Figure 9. These traps force the marble to leave point p in a direction which corresponds to the spiral without the trap.

The write pocket of an L-cell is adjusted so that the marble in the pocket will roll out into the read pocket (q', a) in the adjacent L-cell or get trapped if the maze is rotated the wrong way.

Conclusion of the Proof

We have established the correctness of the simulating maze $L(x)$. It is easy to see that the complexity claims made in Theorem 4.1 are also true: The complexity of a single L-cell is constant, and $L(x)$ contains $O(S(n))$ L-cells. We can construct all these L-cells using only $O(\log S(n))$ space using a binary counter that counts up to $S(n)$. We can compute the size of the counter since $\log S(n)$ is space-constructible. This concludes the proof of Theorem 4.1.

Acknowledgements

We thank Digital Equipment Corporation for supporting this research and Avraham Melkman and Ashok Subramanian for discussions on this problem.

References

[1] J. F. Canny and J. Reif. New lower bound techniques for robot motion planning problems. In *Proceedings of the 28th IEEE Symposium on Foundations of Computer Science*, pages 49–60, 1987.

[2] B. R. Donald. The complexity of planar compliant motion under uncertainty. *Algorithmica*, 5:353–382, 1990.

[3] M. A. Erdmann and M. Mason. An exploration of sensorless manipulation. *IEEE Journal on Robotics and Automation*, 4(4):1–9, Aug. 1988.

[4] J. Friedman, J. Hershberger, and J. Snoeyink. Input sensitive compliant motion in the plane. To appear in The Second Scandinavian Workshop on Algorithm Theory (SWAT 90), 1990.

[5] D. D. Grossman and M. W. Blasgen. Orienting mechanical parts by computer-controlled manipulator. *IEEE Transactions on Systems, Man and Cybernetics*, SMC 5(5):561–565, 1975.

[6] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[7] T. Lozano-Pérez, M. T. Mason, and R. H. Taylor. Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research*, 3(1), 1984.

[8] B. K. Natarajan. *On Moving and Orienting Objects.* PhD thesis, Cornell University Department of Computer Science, Ithaca, N.Y., 1986.

[9] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and Systems Sciences*, 4(2):177–192, 1970.

[10] J. T. Schwartz, M. Sharir, and J. Hopcroft, editors. *Planning, Geometry, and Complexity of Robot Motion.* Ablex Series in Artificial Intelligence. Ablex, Norwood, New Jersey, 1987.