Realizing Site Permutations*

Stephane Durocher† Saeed Mehrabi† Debajyoti Mondal† Matthew Skala†

Abstract

Given n fixed sites on the plane, there are several ways to determine a permutation of the sites as a function of a unit vector u or a vantage point v. Given such a scheme and a permutation π, we can ask whether there is any unit vector or vantage point for which the permutation is π. We give linear-time algorithms for this realization problem under three schemes for determining permutations: sweeping a line across the sites in a direction u; expanding a circle starting from a vantage point v; and sweeping a ray from v to give a cyclic permutation.

1 Introduction

Given an arrangement of points called *sites* on the plane, there are several ways to choose a permutation of the sites. For instance, we could sweep a line across the arrangement and enumerate the sites in the order the line touches them. We could start from some vantage point and consider the sites in order of increasing distance from the vantage point. We could instead sweep a ray from the vantage point radially through all possible angles and consider the circular ordering of the sites it encounters. Other rules are also possible. Given a set of sites S and a geometric rule for defining a permutation of S as a function of a sweep direction or vantage point, some permutations can be realized by some choice of sweep direction or vantage point, and other permutations cannot be realized. In this work we consider the algorithmic problem of recognizing realizable permutations, and describe linear-time algorithms for this problem under three different geometric rules.

Problems of this type have applications in settings that involve computing the position of an observer such as a robot [8] within its environment relative to a sequence of observations made using a directional sensor (such as a sonar, radar, or camera).

2 Definitions and notation

Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of points on the Euclidean plane, called the *sites*. Let $\mathbb{S}^1$ represent the set of directions, or unit vectors, in the plane. Assume that all points and directions in the problem are in general position: that is, no two points are coincident; no three points are collinear; no point is equidistant from two others; no four points w, x, y, and z have the relationship that the line wx is parallel to the line yz; and (in the case of sweep-line permutations) the given sweep direction u is not orthogonal to the line connecting any two points.

For any unit vector $u \in \mathbb{S}^1$ in general position relative to S, let the *sweep-line permutation* of u be the permutation of sites determined by sweeping a line orthogonal to u across the sites in the direction u and enumerating the sites in the order encountered. It would be equivalent to say that we project all the sites onto a directed line parallel to u and define the permutation by the order of the projected sites along the line.

Instead of sweeping a line in a direction, we might start from a point v called the vantage point and enumerate the sites in order of increasing distance from v to form a *distance permutation*. This derivation can be imagined as expanding a circle centred on v and enumerating the sites in the order encountered; or as sending out a sonar ping and recording the order of the echoes received.

Another way of determining a permutation would be by taking a ray starting from v and sweeping it counterclockwise through a complete rotation of 360°, enumerating the sites in the order the ray encounters them.[1] Then we obtain a cyclic permutation; that is, an equivalence class of permutations up to rotation. This *radial permutation* is analogous to scanning the sites with a rotating search light or radar beam, and recording the order in which we see them without regard for the angles or the starting orientation of the sweep.

Figure 1 illustrates the three kinds of site permutations we consider. In the figure, a line swept in the direction u encounters the sites in the order *dcba*. An expanding circle starting at v encounters the sites in the order *bdac*; and a ray originating at v and swept counterclockwise encounters them in the order *cdba*, up to a rotation that depends on the starting orientation of the sweep. For any of these schemes, given a permutation or cyclic permutation π and a set of sites, a unit vector u or vantage point v is said to *realize* π if π is the permutation determined by u or v for the given

*Work supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

†Department of Computer Science, University of Manitoba, {durocher,mehrabi,jyoti,mskala}@cs.umanitoba.ca

[1] We describe angles using degrees to avoid confusion with the symbol π used for a generic permutation.

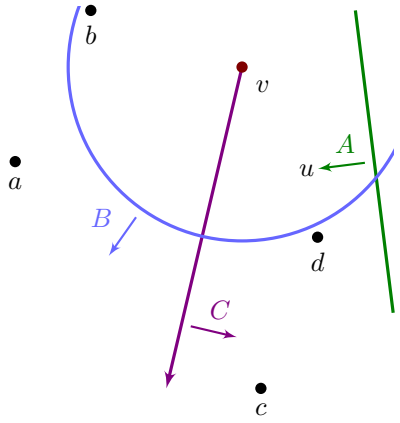

Figure 1: *A*. sweep-line permutation in direction u: *dcba*. *B*. distance permutation centred at v: *bdac*. *C*. radial permutation centred at v: *cdba*.

scheme and sites. Then π is said to be *realizable* if and only if there is a vantage point or unit vector realizing it. In this work we consider the problem of deciding whether a permutation π is realizable and, if so, computing a corresponding unit vector u or vantage point v that realizes π.

If the vantage point v is sufficiently far from the sites in the direction opposite to u, then the expanding circle centred on v when it passes over the sites is equivalent to a line orthogonal to u and sweeping in the direction u. Similarly, if the vantage point v is sufficiently far from the sites in a direction 90° counterclockwise from u, then the sweeping ray from v when it passes over the sites is equivalent a line sweeping in the direction u. We can thus make the following observation.

Observation 1 *Every sweep-line permutation for an arrangement of sites is also realized as a distance permutation and a radial permutation.*

Throughout our algorithmic results we assume a real RAM model of computation, in which we can perform basic arithmetic operations in unit time. This is a standard assumption for computational geometry algorithms in general; and in particular, the linear-time linear programming algorithm of Megiddo [5], which we use, is only linear-time under the assumption it can complete in constant time the multiplication and division operations needed to find the intersections of lines given as input. Analysing the algorithms under some other model to force a superlinear result would be primarily an exploration of the complexity of arithmetic in general without giving specific insight into these algorithms.

3 Previous work

The cyclic sequence of sweep-line permutations formed by a site arrangement as we rotate the sweep direction through a full circle is called an *allowable sequence*, and allowable sequences are well-studied. Goodman and Pollack pioneered the use of allowable sequences in characterizing the order type of the sites [4]. The allowable sequence for a site arrangement is closely connected to the oriented matroid associated with the site arrangement, and that connection leads to many combinatorial insights [2].

Chávez, Figueroa, and Navarro introduced distance permutations in a database context, as a way of classifying points in high-dimensional general metric spaces to support efficient proximity queries [3]. Note that this kind of permutation (possibly with a tiebreaking assumption added to handle degenerate cases) is defined for any space with a real distance function—it need not even be a metric. Skala proved bounds on the number of distinct distance permutations that can occur as a function of the number of sites in various spaces, including an exact count for Euclidean spaces [6].

Bieri and Schmidt studied radial permutations as well as sweep-line permutations and a variation on radial permutations in which a line is swept instead of a ray [1]. Noting that the number of radial permutations realized by a site arrangement is $\Theta(n^4)$ (which follows from the number of bisectors and the fact that k lines in general position on the plane divide the plane into $\Theta(k^2)$ cells), they give an algorithm to generate all the permutations in $\Theta(n^4)$ time—interesting because the naive size of the output would be $\Theta(n^5)$. To achieve the faster running time, they order the permutations in such a way that each except the first differs from some previous permutation by one swap of adjacent elements; then the swaps can be found in $O(n^4)$ time. Tovar, Freda, and LaValle studied radial permutations in the context of robot navigation; assuming a robot with a sensor that detects the radial permutation of landmarks as seen from its current location, they show how the robot can achieve navigational goals [8].

4 Bisectors and Voronoi diagrams

The sweep-line method of finding a permutation implicitly divides the set of possible directions into intervals corresponding to the realizable permutations. Similarly, the distance and radial permutations correspond to cells of a Voronoi-like diagram in the plane. These divisions are shown in Figure 2. Note that the unbounded cells for distance and radial permutations correspond to the permutations realized by points at infinity, and thus to the sweep-line permutations (Observation 1).

Every pair of sites s_i and s_j determines a *bisector*: a set of points where the ordering of s_i and s_j is not

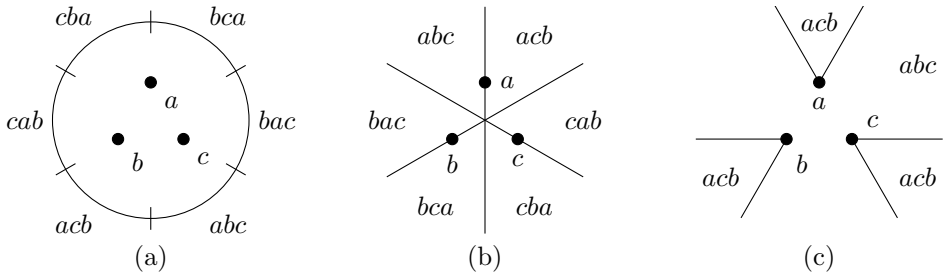

Figure 2: Division of space by permutation schemes: (a) sweep-line, (b) distance, (c) radial.

uniquely defined. If we imagine a point wandering continuously through the space (like Tovar, Freda, and LaValle's robot [8]), the permutation it observes will change by a swap of adjacent elements each time it crosses a bisector. For sweep-line permutations, the bisector of s_i and s_j consists of the two unit vectors parallel to the line between s_i and s_j. For distance permutations, it is the set of all points equidistant from s_i and s_j, which is the line orthogonally bisecting the segment that connects the two sites. For radial permutations, it is the line connecting s_i and s_j, with the segment between them removed. The radial bisector is unusual because it can be said to cut the plane into just one piece: with two sites, only one permutation exists up to rotation, so there is only one cell. Radial bisectors become more meaningful once there are three or more sites.

Examination of these divisions of space leads to simple counts or bounds on the number of permutations realized. For sweep-line permutations, the bisectors each consist of two points, and distinct bisectors never coincide when sites are in general position, so it is trivial that the number of intervals and thus permutations for n sites is $2\binom{n}{2}$. For distance permutations, $\binom{n}{2}$ bisectors and the quadratic bound on number of cells formed by lines in general position gives an upper bound of $O(n^4)$ permutations; Skala notes that bisectors are not in general position because of transitivity, and gives an exact recurrence for the number of permutations, as well as generalizing the question to higher dimensions of Euclidean space; in d dimensions the number of permutations is shown to be $\Theta(n^{2d})$ [6]. For radial permutations, the same kind of argument gives an obvious $O(n^4)$ upper bound, but the possibility for a permutation's cell to be non-convex or even disconnected (as in Figure 2(c)) complicates matters. Bieri and Schmidt state as a theorem (without detailed proof) that the upper bound is achieved by some arrangement of n sites for every n [1].

5 Radial permutations in the dual space

For each site s_i in $s_1, s_2, \ldots, s_n$, define a line s_i^* as follows: let (x_i, y_i) be the coordinates of s_i, and then let s_i^* be the line dual to s_i, defined by $y = x \cdot x_i - y_i$. Let $v = x_v, y_v$ be a point in the plane, not equal to any of the sites, and similarly define its dual line v^* by $y = x \cdot x_v - y_v$. These points and lines are shown in Figure 3. The vantage point v was chosen to be the origin for convenience in making and understanding the figure; its image in dual space is the x axis. The sorted sequence of the segments (v, s_i) around v corresponds to an ordered sequence of intersections between the lines v^* and s_i^*, as a line connecting two points in primal space corresponds to the intersection of two lines in dual space.

Let L be the vertical line passing through v. L divides the plane into two half-planes. In Figure 3(a), the right half-plane contains s_3, s_5, and s_6, and the left half-plane contains s_1, s_2, and s_4. In Figure 3(b), the crossing points for s_3^*, s_5^*, and s_6^* (shown in white) appear consecutively right to left. Similarly, the crossing points for s_1^*, s_2^*, and s_4^* (shown in black) appear consecutively left to right. We can concatenate the two lists to obtain a radial permutation π of the sites around v: $s_1, s_2, s_4, s_3, s_5, s_6$.

The dual space naturally suggests another sequence of the sites, that found by examining all the crossings (not black and white separately) along the line. From right to left that sequence is $s_1^*, s_2^*, s_3^*, s_5^*, s_4^*, s_6^*$. In the primal space it corresponds to rotating a line, not a ray, passing through v, starting at vertical and then 180° counterclockwise until it becomes vertical again, and enumerating the sites in the order the line encounters them. This variation of radial permutations corresponds to the *undirected stars* described by Streinu [7]. If we add a sign to each element in the sequence describing whether it was hit by the head or the tail of the line during the radial sweep, the result is a *directed star* (or simply a *star*) as described by Streinu; in Figure 3, using Streinu's notation, the star would be $12\bar{3}\bar{5}4\bar{6}$, where x and $\bar{x}$ denote that element x was met by the head or tail end, respectively, of the rotating line. Given a directed star, it is straightforward to construct the corresponding radial permutation in linear time.

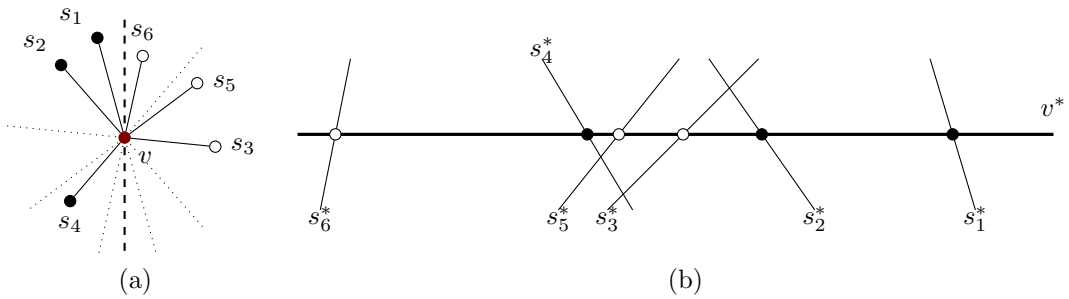

Figure 3: A radial permutation in (a) primal and (b) dual spaces.

6 Results

The bisectors for sweep-line permutations suggest a simple linear-time algorithm for realizing permutations; in fact, because the cells are simply intervals around the circle, we not only compute a single unit vector to realize the permutation, but also completely describe the set of all such vectors in the same asymptotic time.

Theorem 1 *There exists a linear-time algorithm that given sites $s_1, s_2, \ldots, s_n$ and a permutation π on the indices, finds the set of all directions u for which the sweep-line permutation is π.*

Proof. By transitivity, it suffices to enforce the $n-1$ constraints that the sweep line reaches $s_{\pi(1)}$ before $s_{\pi(2)}$, $s_{\pi(2)}$ before $s_{\pi(3)}$, and so on. Each of those constraints corresponds to an interval of allowed values for u in $\mathbb{S}^1$. Each interval is open and has length 180°; therefore the intersection of any two of them is a single, possibly smaller, interval; and by associativity we can compute the intersection of all of them in $O(n)$ time. □

In the case of distance permutations, linear time does not allow us to examine all of the quadratic number of bisectors; but because the bisectors correspond to the transitive "less than" relation on distances, we can obtain all the necessary information by examining a linear-sized subset of them.

Theorem 2 *There exists a linear-time algorithm that given sites $s_1, s_2, \ldots, s_n$ and a permutation π on the indices, finds a vantage point v for which the distance permutation is π, if such a v exists.*

Proof. By transitivity, it suffices to enforce the $n-1$ constraints that v is closer to $s_{\pi(1)}$ than to $s_{\pi(2)}$, closer to $s_{\pi(2)}$ than to $s_{\pi(3)}$, and so on. Each of those corresponds to a half-plane (linear) constraint. By the linear-time two-dimensional linear programming algorithm of Megiddo [5], we can find a point v satisfying all the constraints in $O(n)$ time. □

Radial permutations present a greater challenge, primarily because we are seeking not a single permutation but an equivalence class of permutations. We begin by proving a connection between the realization problem and linear programming.

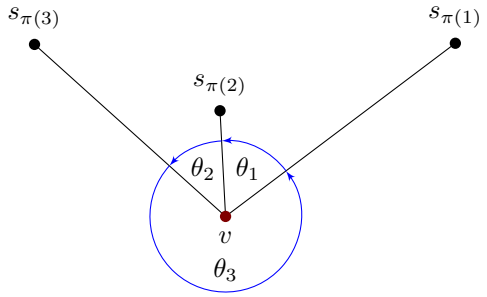

Figure 4: Angles measured around v.

Lemma 3 *There exists a linear-time algorithm that given sites $s_1, s_2, \ldots, s_n$ and a permutation π on the indices, constructs a set of linear constraints such that any vantage point v for which the radial permutation is π up to rotation satisfies all, or all but one, of the constraints.*

Proof. Where v is the vantage point, for each integer $1 \le i \le n$, let θ_i denote the angle measured counterclockwise around v from the ray pointing at $s_{\pi(i)}$ to the ray pointing at $s_{\pi(j)}$, where $j = (i \bmod n)+1$, as shown in Figure 4. Let $\Theta = \sum_{i=1}^{n} \theta_i$, that is, the sum of all the θ_i.

For each pair of successive sites $s_{\pi(i)}$ and $s_{\pi(j)}$ where $\theta_i < 180°$, it must be that v, $s_{\pi(i)}$, and $s_{\pi(j)}$ form a triangle in counterclockwise order, like v, $s_{\pi(1)}$, and $s_{\pi(2)}$ in Figure 4. That is equivalent to the statement that v is on the left side of the directed line from $s_{\pi(i)}$ to $s_{\pi(j)}$, and we can express that statement as a half-plane constraint. We create such a constraint for each pair of successive sites.

One way v might realize π would be if it were in the kernel of a star-shaped polygon formed by the sites in the order described by π; then every $\theta_i < 180°$ and all the linear constraints would be satisfied. This situation is illustrated in Figure 5(a).

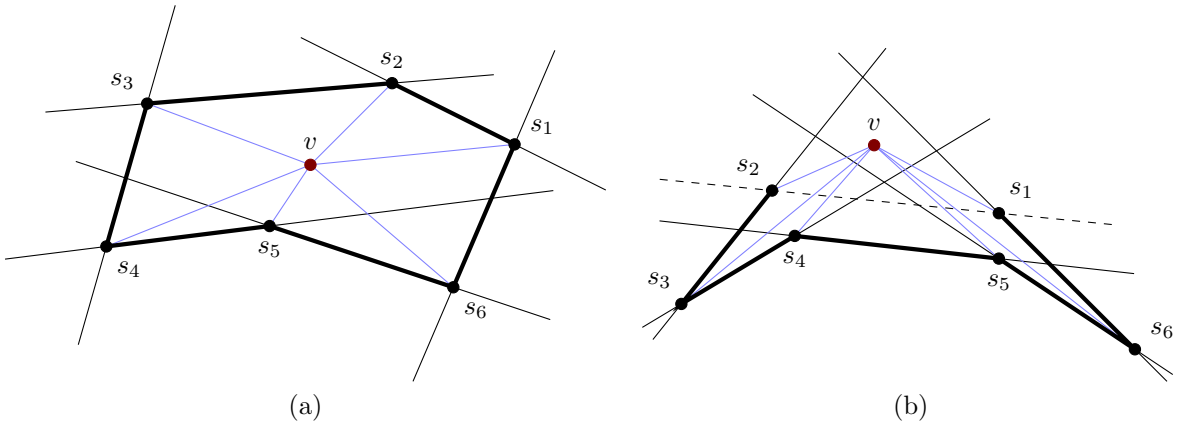

Figure 5: Realizing a permutation while violating (a) zero or (b) one of the linear constraints (π is the identity).

It is also possible for the vantage point v to lie outside the kernel of the star-shaped polygon, as shown in Figure 5(b). However, if v realizes π, then summing all the θ_i corresponds to making one full sweep around v; $\Theta = 360°$. Thus, at most one of the θ_i can be greater than 180°, corresponding to a violated constraint; all the others must be satisfied. Therefore, at most one of the constraints can be violated. □

To actually solve the realization problem we must not only perform linear programming but also determine which constraint to violate, if any. Here we exploit the special properties of Megiddo's linear programming algorithm [5], which either finds a solution to the realization problem immediately, or gives us a clue to where the permutation must start.

Theorem 4 *There exists a linear-time algorithm that given sites $s_1, s_2, \ldots, s_n$ and a permutation π on the indices, finds a vantage point v for which the radial permutation is π up to rotation, if such a v exists.*

Proof. We invoke the linear-time two-dimensional linear programming algorithm of Megiddo [5] to find a point satisfying all the constraints of Lemma 3, if possible. We can then distinguish two cases: (1) such a point exists; or (2) no such point exists.

Case 1. It is possible that a point v could satisfy all the constraints but not realize the permutation π, if the sequence of sites described by π winds more than once around v. An example demonstrating this situation is shown in Figure 6. When the linear program returns a solution v, it is easy to test in linear time whether v realizes the permutation π. If it does, the algorithm returns it immediately.

Suppose v does not realize π. The cumulative angle Θ must be an integer multiple of 360°; and when v is a solution to the linear program but does not realize the desired permutation, it must be at least 720°. Removing one constraint (changing the polygon to a path, which might still be self-intersecting) reduces the sum for the remaining pairs of sites by strictly less than 360°, leaving it strictly greater than 360°. Now suppose we start our ray sweep with a ray pointing from v', a solution to the linear program with one constraint relaxed, to the site at the start of the path. If we sweep to each successive site on the path in turn, we will complete a full angle (360°) and see the start of the path again, before we complete the sweep at the end of the path. That means we must already have seen the end of the path, before its proper place at the end of the sweep. Therefore v' cannot realize the permutation π. In intuitive terms, if the polygon wraps more than once around some solution, it must wrap at least twice, and then the path formed by deleting one edge from the polygon (which subtracts less than 360°) must still wrap more than once around every solution.

We have that if there exists a vantage point v that is a solution to the linear program but does not realize the permutation π, then no point v' which is a solution to any linear program formed by relaxing one of the original constraints, can realize the permutation π. Since every point realizing π must be a solution to our original linear program with at most one constraint relaxed, then there can be no point realizing π at all. Thus, in Case 1, where the linear program is feasible, it suffices to test whether the solution v realizes π, return it if it does realize π, and return failure if it does not.

Case 2. If the first linear program is infeasible, then any v realizing π must be a solution to the linear program with exactly one constraint relaxed. Megiddo's algorithm [5] works by examining constant-sized subsets of the input constraints and, at each one, attempting to prove that at least one of the constraints is unnecessary for the optimal solution. His analysis shows that in each of the subsets at least one constraint can always be removed if the input is feasible, allowing the algorithm to stop after a linear number of steps with either a solution or a proof of infeasibility. That approach has the impor-

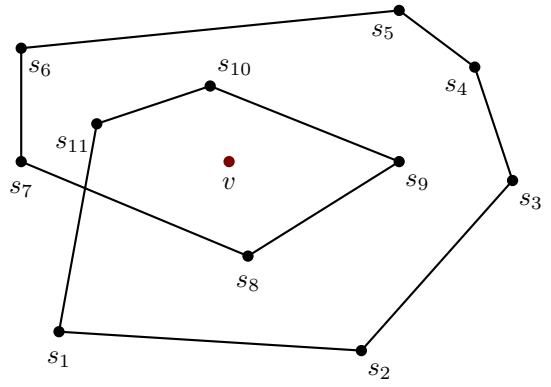

Figure 6: The sites wind more than once around v (π is the identity).

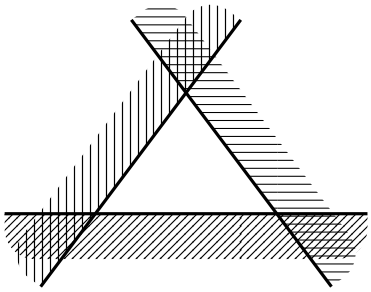

Figure 7: An infeasibility certificate.

tant consequence that in case of an infeasible input, the algorithm actually finds a constant-sized certificate of infeasibility, namely the last subset of input constraints it examined before halting. The algorithm can easily be modified to produce the certificate as output, in the form of at most three constraints that cannot all be satisfied. In general those constraints will be arranged as shown in Figure 7; with input not in general position a certificate consisting of two non-intersecting parallel half-planes would also be possible.

In order to be a vantage point realizing the desired radial permutation, v would have to satisfy all except at most one of the constraints in the original linear programming problem. If v can satisfy all except one, but not all of the constraints, then every infeasible subset of the constraints must include that one constraint, so it must be among the at most three returned when Megiddo's algorithm failed. By invoking Megiddo's algorithm at most three more times, with each constraint from the certificate removed in turn, we can find a value for v that realizes the permutation π, if any exists. □

7 Conclusion

In this paper, we considered the problem of realizing a permutation π on a set of n sites in the plane. We gave three linear-time algorithms for this kind of problem, corresponding to three schemes of determining permutations: sweeping a line in direction u, measuring distance from a vantage point v, and sweeping a ray counterclockwise around v. One obvious direction for future work is to consider other ways of determining a permutation; for example, rotating a line through v instead of a ray starting at v. We might also consider more general kinds of constraint satisfaction involving site permutations; for instance, finding a point that realizes any permutation containing a given contiguous subsequence.

Acknowledgments

We wish to thank Pak Ching Li and Jason Morrison for insightful discussions on these and related problems.

References

[1] H. Bieri and P.-M. Schmidt. On the permutations generated by rotational sweeps of planar point sets. In *Proceedings of the 8th Canadian Conference on Computational Geometry, Ottawa, Canada (CCCG 1996)*, pages 179–184, August 12–15 1996.

[2] A. Björner, M. L. Vergnas, B. Sturmfels, N. White, and G. M. Ziegler. *Oriented Matroids*. Cambridge University Press, 2nd edition, 1999.

[3] E. Chávez, K. Figueroa, and G. Navarro. Proximity searching in high dimensional spaces with a proximity preserving order. In *Proceedings of the 4th Mexican International Conference on Artificial Intelligence (MICAI 2005), Monterrey, Mexico*, volume 3789 of *Lecture Notes in Computer Science*, pages 405–414. Springer, November 14–18 2005.

[4] J. E. Goodman. On the combinatorial classification of nondegenerate configurations in the plane. *Journal of Combinatorial Theory, Series A*, 29(2):220–235, September 1980.

[5] N. Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM Journal on Computing*, 12(4):759–776, November 1983.

[6] M. Skala. Counting distance permutations. *Journal of Discrete Algorithms*, 7(1):49–61, March 2009.

[7] I. Streinu. Clusters of stars. In *Proceedings of the 13th Annual Symposium on Computational Geometry (SCG 1997), Nice, France*, pages 439–441, June 4–6 1997.

[8] B. Tovar, L. Freda, and S. M. LaValle. Learning combinatorial map information from permutations of landmarks. *International Journal of Robotics Research*, October 2010.