Realizing Site Permutations

Stephane Durocher, Saeed Mehrabi,
Debajyoti Mondal, Matthew Skala

Site permutations

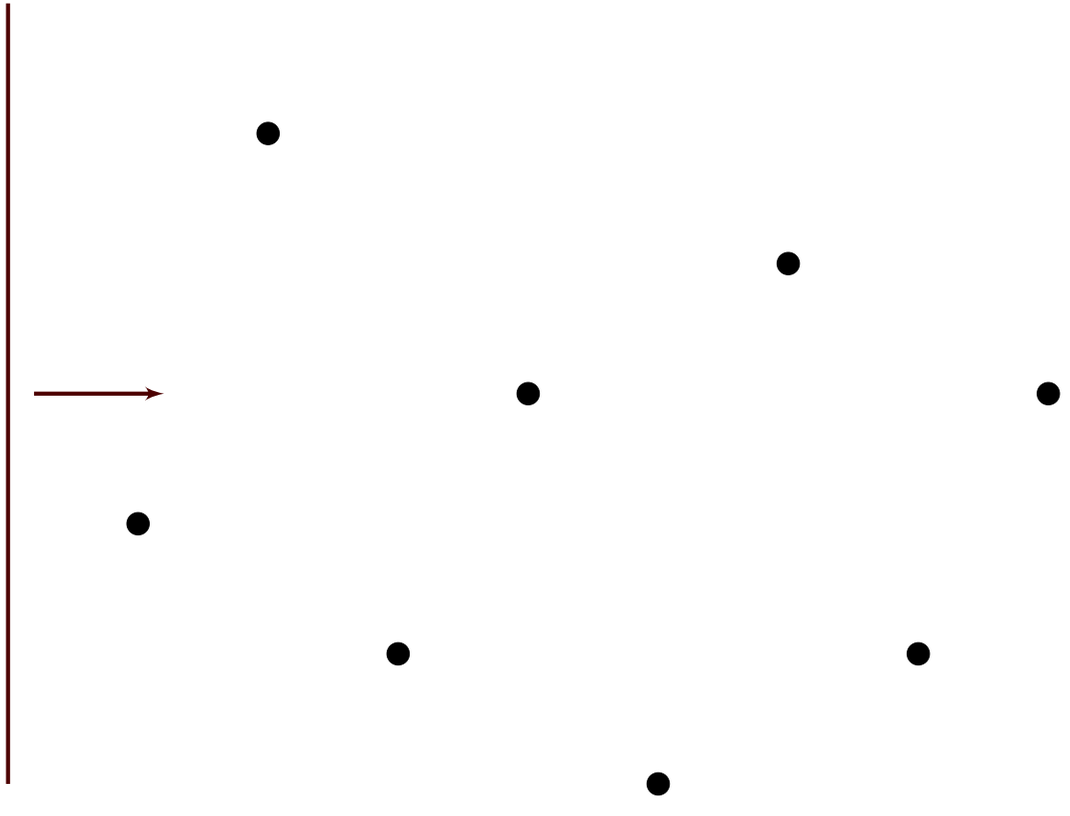
We are sitting somewhere on the Euclidean plane. We want to describe our location in relation to some fixed, known "sites."

One way to do it would be to use some numbers, such as distances or angles to the sites from our current vantage point.

What if instead, we have to express our location as a *permutation* of the sites?

Sweep-line permutations

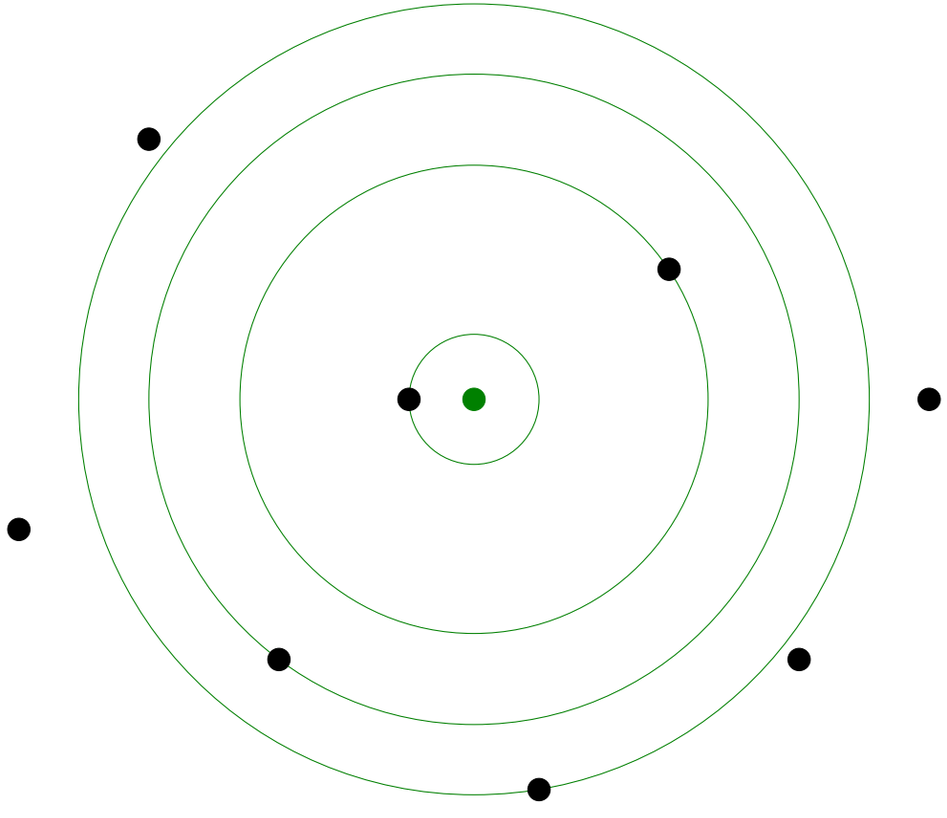
Okay, we're not actually at a location; we are at infinity and looking in a direction. When we sweep a line across the arrangement of sites, in what order does it encounter the sites?



Alternatively, and equivalently, if we project all the sites onto a ray, in what order do they occur along the ray?

Distance permutations

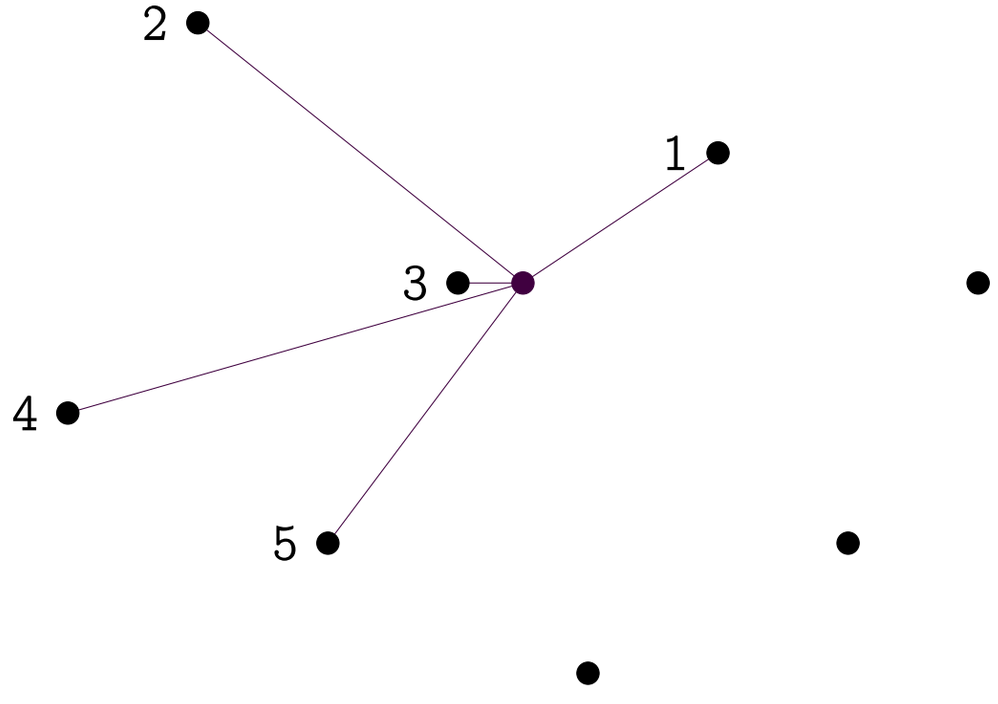
Starting from our vantage point, what is the closest site? What is the next closest? And so on...



Imagine this as sending out a non-directional sonar ping and recording the order in which the echoes come back without their exact timing.

Cyclic permutations

Sweep a ray, originating at our vantage point, counterclockwise through a circle. Record the *cyclic* permutation—so 1234 is considered the same as 2341.



Imagine this as rotating a radar beam in a circle and writing down the order in which we get the returns, without their exact azimuths.

The realization problem

For one of these schemes, given the locations of the sites and a permutation π, find a point (or the set of points, or a direction) whose permutation under the selected scheme is π.

We can do this, with appropriate variations, in linear time for all three schemes. We assume a real RAM and general position.

Typical application: if we have a map, and a sensor that returns one of these permutations, we want to find our location (or at least *a* location) consistent with the sensor readings.

Related previous work

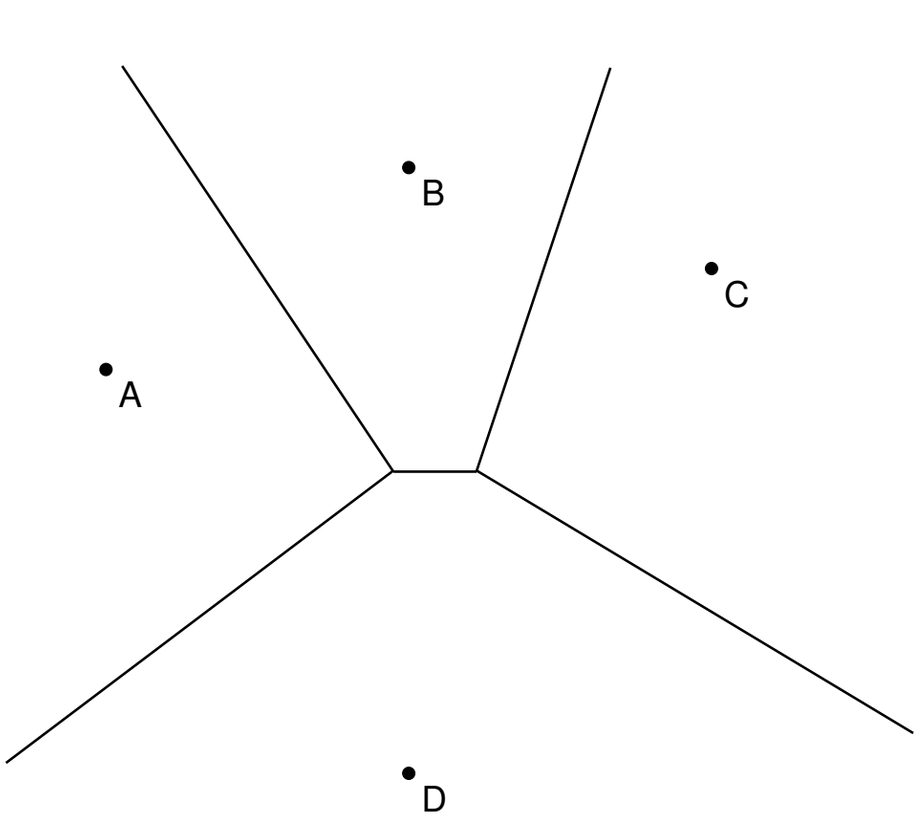
Sweep-line permutations: as we move the sweep direction through a circle, the sequence of sweep-line permutations thus produced is called an *allowable sequence* and is the subject of much combinatorial work (Goodman and Pollack 1980, etc.). It characterizes the *order type* of the arrangement of sites.

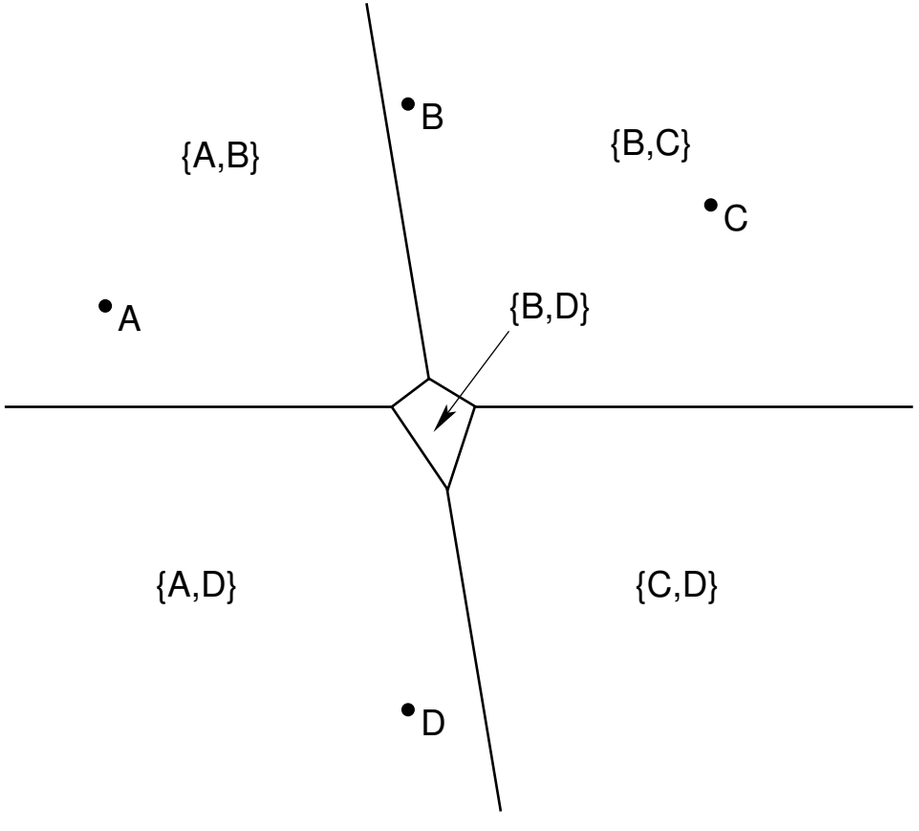
Distance permutations: introduced for distance-based indexing data structure (put points in bins according to permutation, near points should have near permutations; Chávez, Figueroa, and Navarro, 2005). Known number realized depending on number of sites and dimensions, $\Theta(k^{2d})$ (Skala, 2009)

Radial permutations: Bieri and Schmidt (1996) find them all in $\Theta(n^4)$ time, interesting because size of output is $\Theta(n^5)$. Tovar, Freda, and LaValle (2010) discuss them for robot navigation.

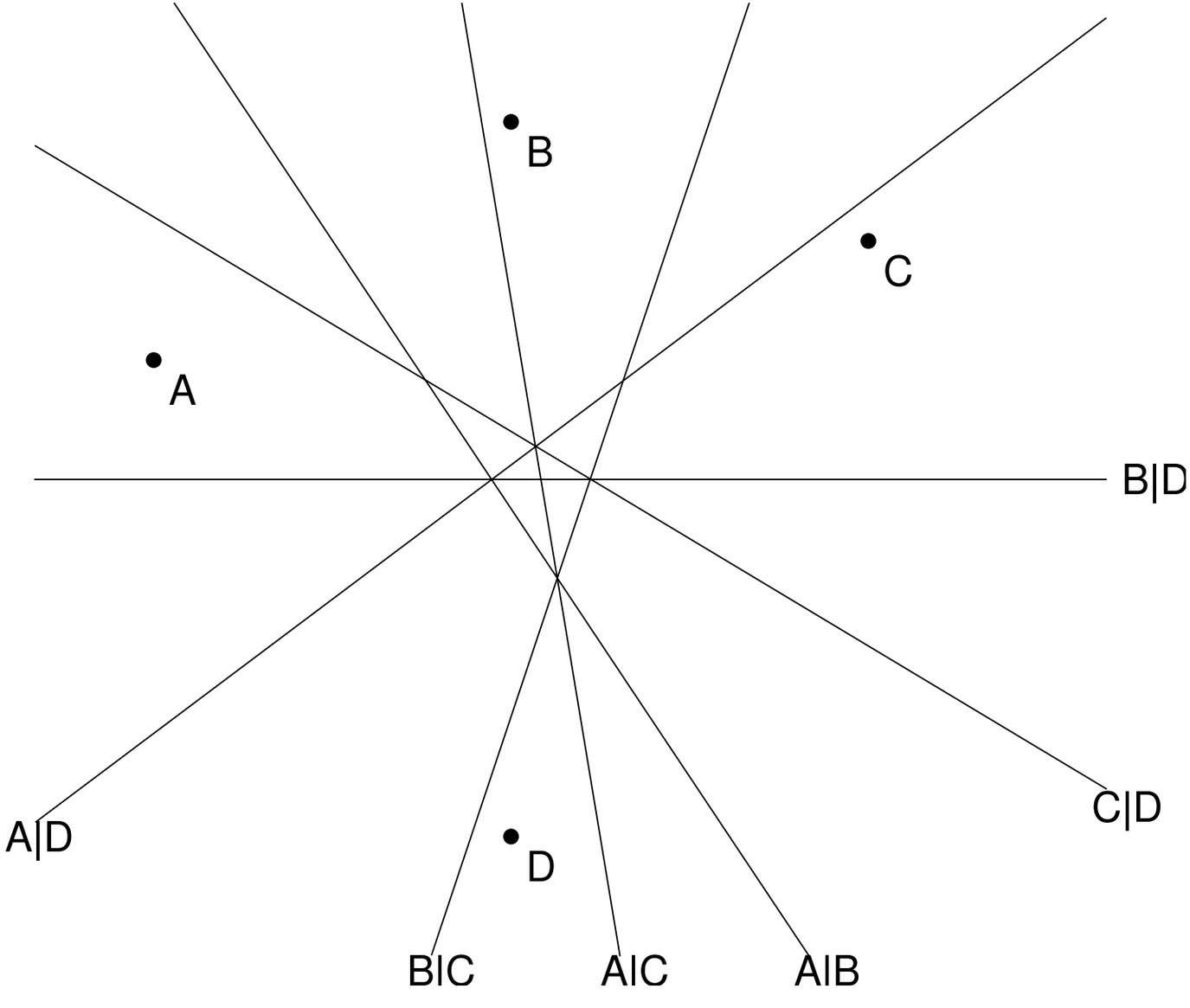
"Clusters of stars" (Streinu, 1997): like radial permutations but we rotate a line instead of a ray, with or without directions. Reconstructing site arrangement from each site's permutation of the others.

Generalized Voronoi diagrams

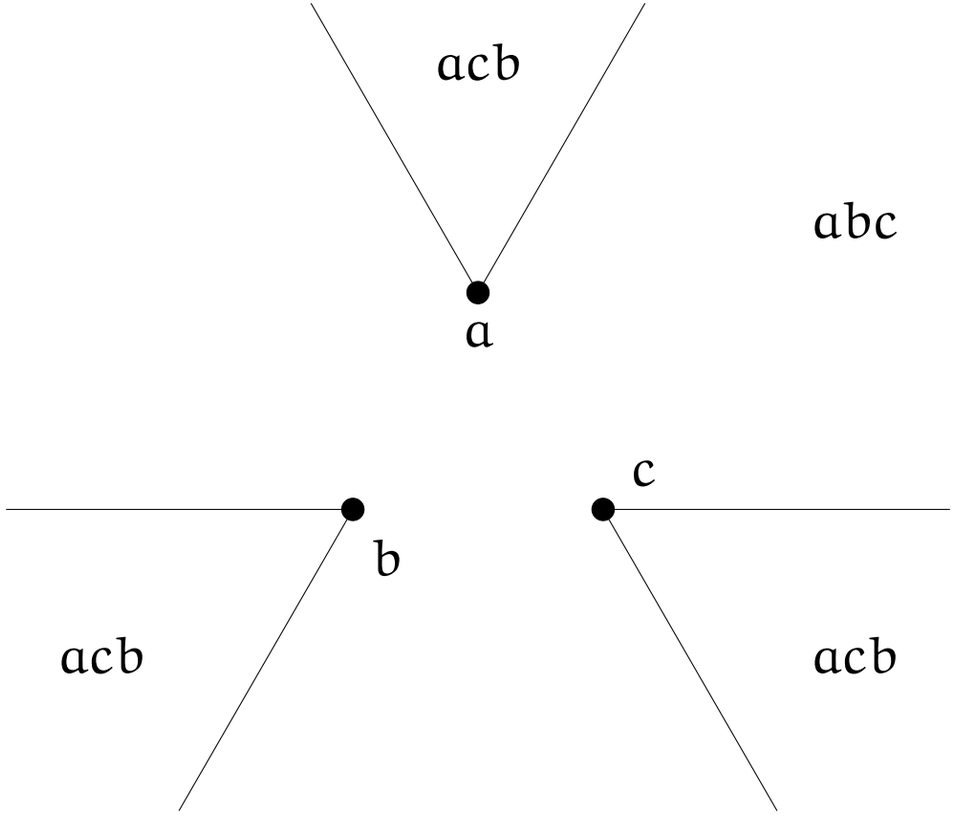




Voronoi diagram for distance permutations



Voronoi diagram for cyclic permutations



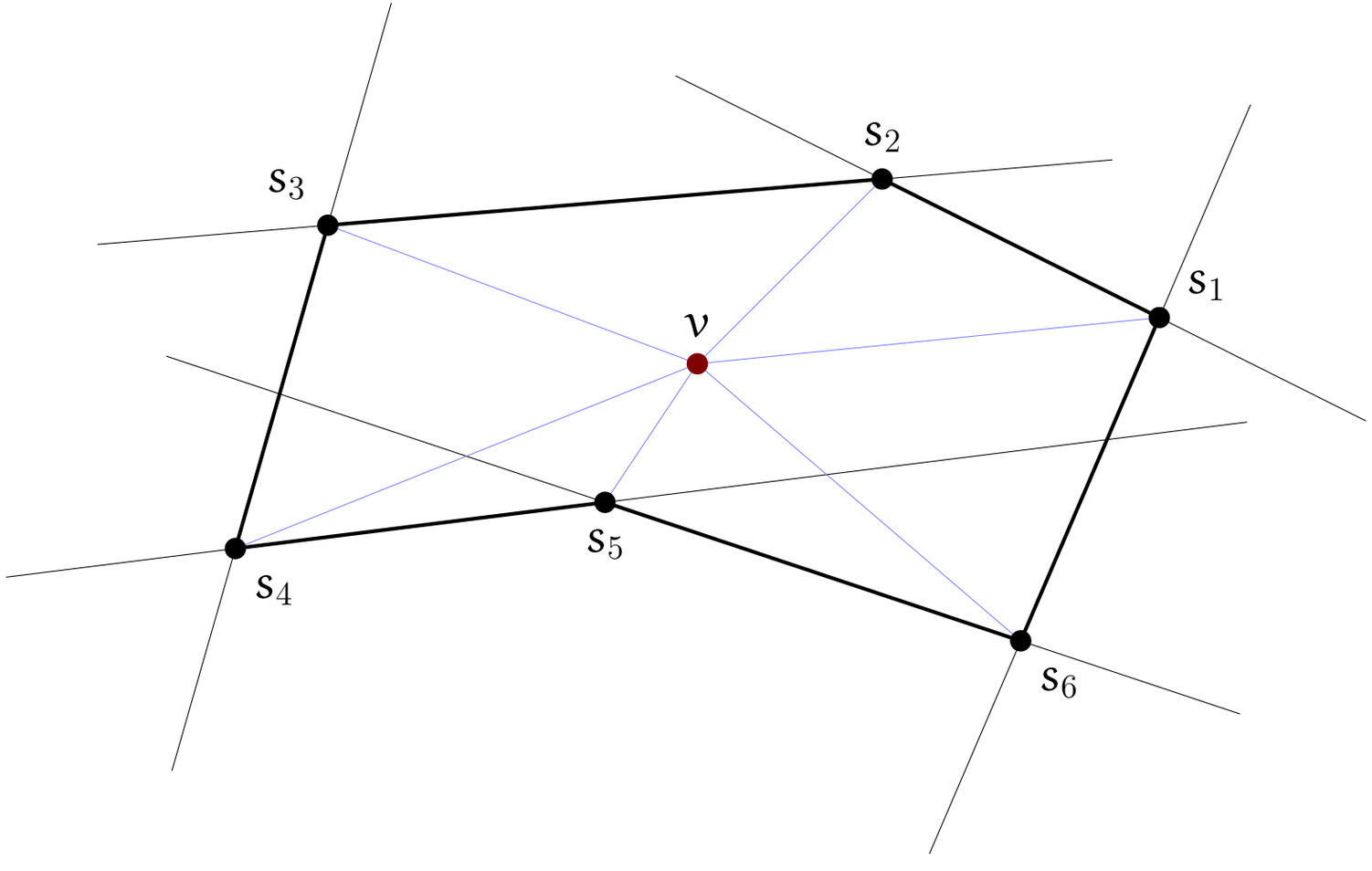
Realizing sweep-line permutations

- For each pair of sites x, y, where x comes before y in the permutation, that means the sweep line must hit x before it hits y. That corresponds to a 180° interval of sweep line directions.
- Because of transitivity, we only really care about enforcing the constraints between sites that are succesive in the permutation; therefore, a linear number of constraints.
- Take the intersection of all those intervals. The general position assumption guarantees it will be a single interval.
- Linear time to find the interval of all directions that will realize the permutation.

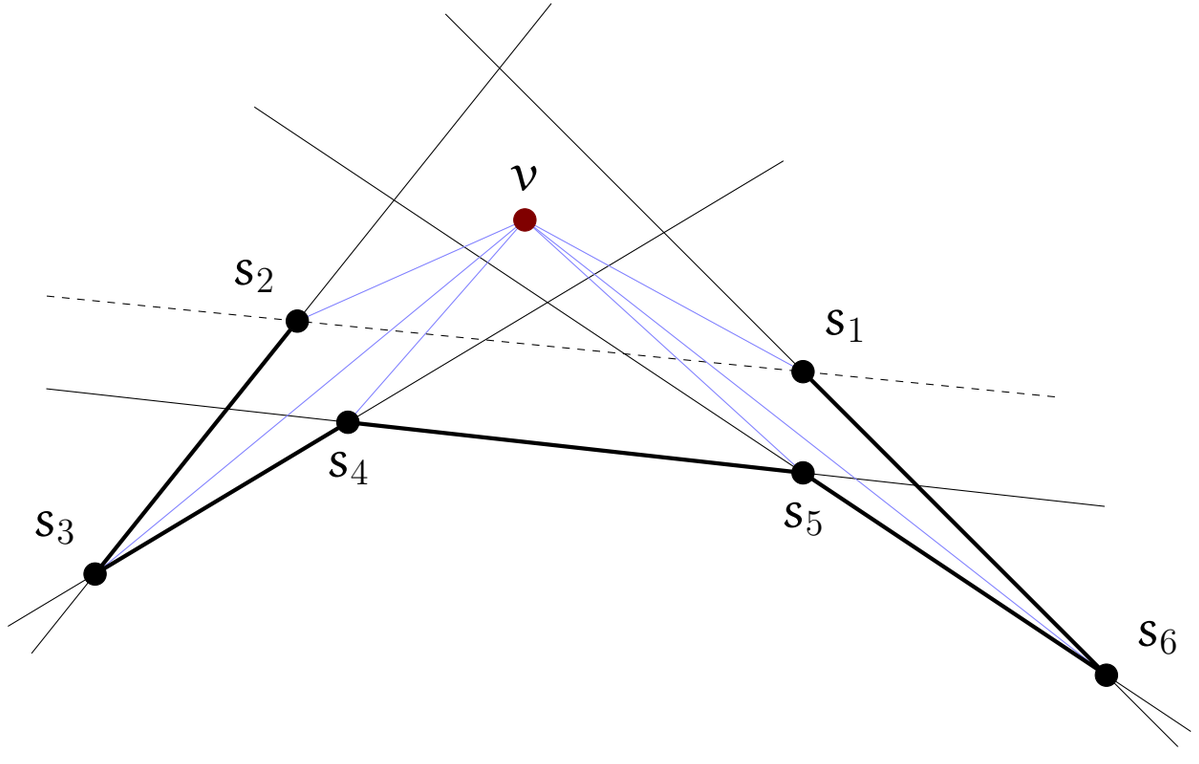
Realizing distance permutations

- Transitivity, as before, reduces the number of constraints we must enforce to linear, namely those corresponding to successive sites.
- Each constraint says "must be on this side of the line that bisects the two sites"; that is a linear constraint.
- Find a point satisfying linear constraints: that is linear programming.
- Solvable in linear time by Megiddo's linear-time fixed-dimension LP. (In arbitrary constant dimension, if we use the more complicated one.)

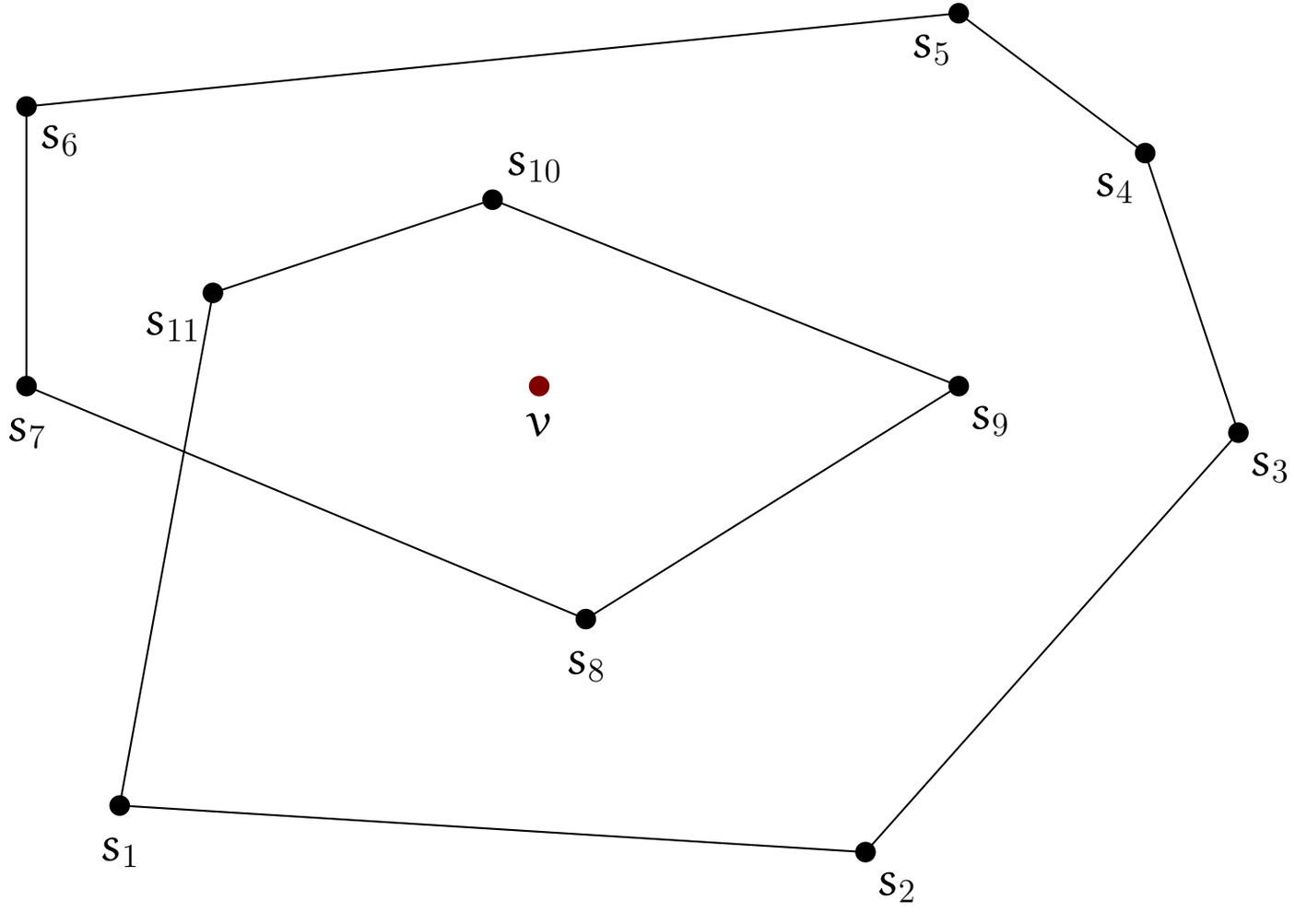
Satisfying all constraints



Satisfying all but one constraint



Satisfying the constraints is not enough



Realizing radial permutations

- If it is possible to satisfy all constraints, and there is a solution, then any point that satisfies all constraints must be a solution.
- So try to satisfy all constraints. (Megiddo, 1983, linear time)
- If we can solve the LP problem: test whether the result has the correct permutation, and we're done.
- If we can't solve the LP problem: we get a certificate that happens to be three incompatible constraints. If any point satisfies all but one constraint, the violated constraint must be one *of those*.
- Attempt three more LP problems, leaving out each constraint from the certificate, and we're done.

Conclusions and future work

- Three ways to determine a permutation: sweep a line in a specified direction, take sites in order of distance, or sweep a ray and get a cyclic permutation.
- Realization problem: find an input (unit vector or point as appropriate) to give a desired permutation.
- For sweep-line permutations, it's easy, just intersect the intervals. (Finds all answers.)
- For distance permutations, it's linear programming.
- For radial permutations, it's linear programming with possibly one constraint relaxed and a check that the answer really is satisfactory; and we can do all that in linear time.
- Future work: other schemes for determining site permutations; other kinds of constraint satisfaction involving site permutations; other spaces.