Seminar 7 – Mobile Robots: Square Formation and Meeting

Distributed Computing in Anonymous Dynamic Systems

Giovanni Viglietta

Rome – March 14, 2024

Syllabus

- Anonymous Networks
  - Introduction and basic algorithms for static networks
  - Dynamicity and history trees
  - Optimal computation in networks with and without leaders
  - Computation in dynamic congested networks
- Population Protocols
  - Introduction and basic algorithmic techniques
  - Leader election in Mediated Population Protocols
- Mobile Robots
  - Gathering and Pattern Formation in the plane
  - Meeting in a polygon by oblivious robots

Exam

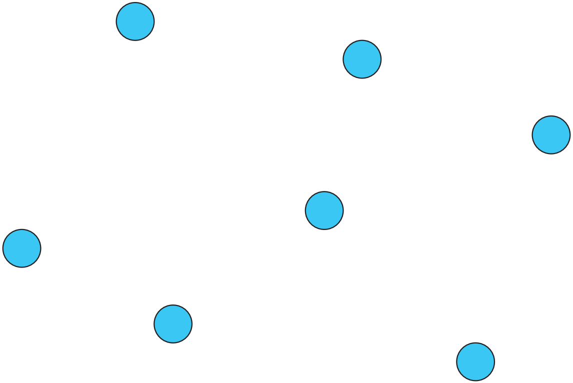
Pre-recorded 10-minute presentation video on one of the papers that will be suggested at the end of the course.

Today's seminar

- Mobile robots in the plane
- Square Formation problem
- Meeting problem in a polygon
  - With memory
  - With no memory

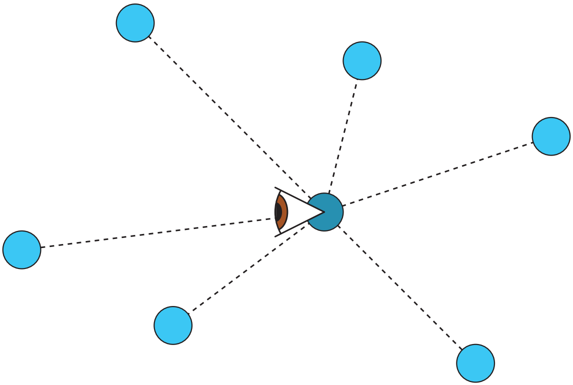
Square Formation

Anonymous robots sensing and moving in the plane



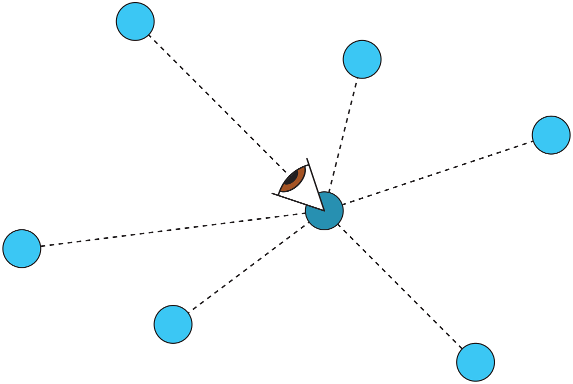
We consider a swarm of anonymous robots in the Euclidean plane

Anonymous robots sensing and moving in the plane



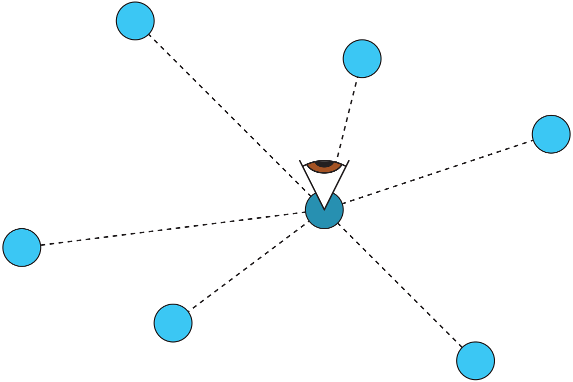
Each robot can sense the positions of all other robots...

Anonymous robots sensing and moving in the plane

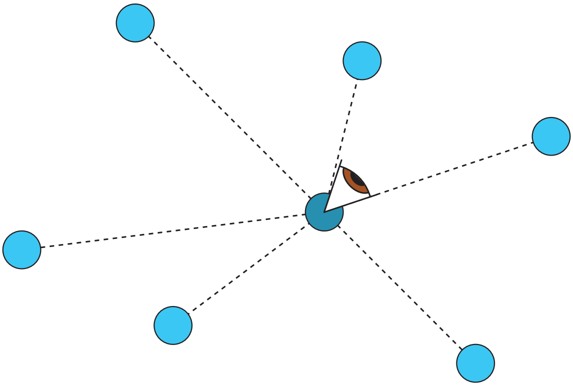


Each robot can sense the positions of all other robots...

Anonymous robots sensing and moving in the plane

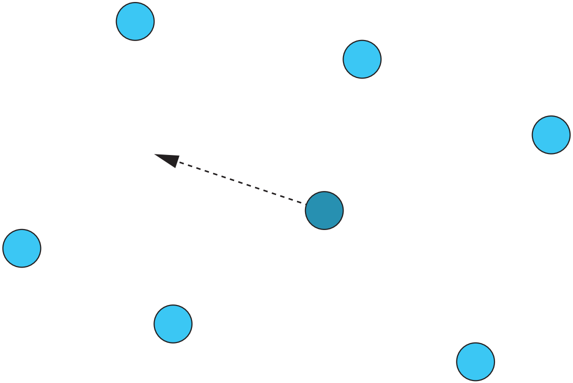


Each robot can sense the positions of all other robots...



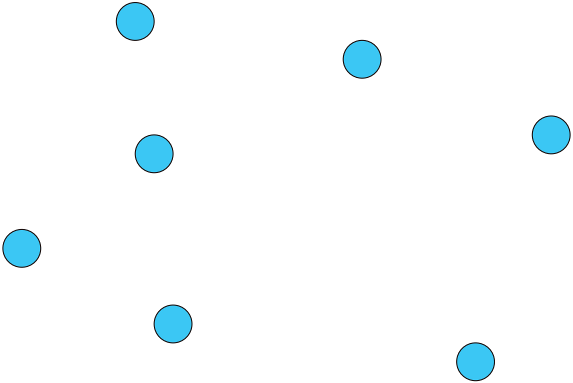
Each robot can sense the positions of all other robots...

Anonymous robots sensing and moving in the plane



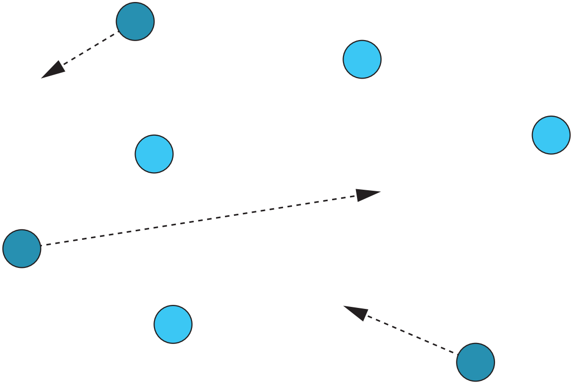
...And move according to a deterministic algorithm

Anonymous robots sensing and moving in the plane



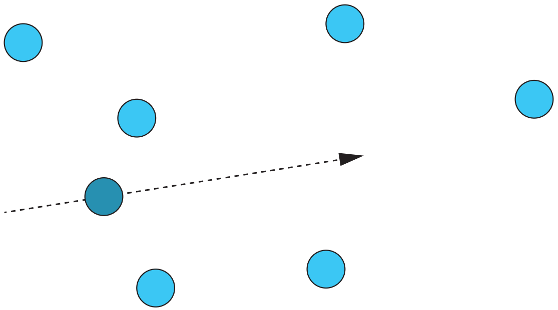
...And move according to a deterministic algorithm

Anonymous robots sensing and moving in the plane



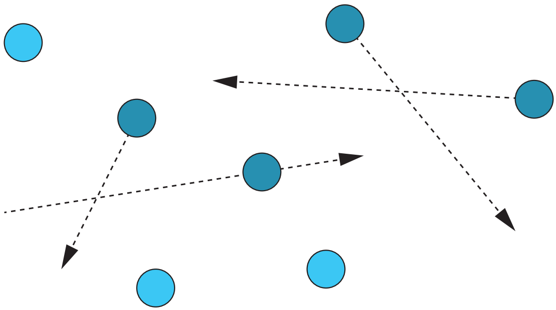
Different robots are activated asynchronously

Anonymous robots sensing and moving in the plane



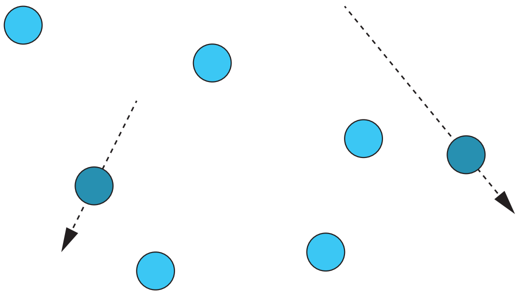
Different robots are activated asynchronously

Anonymous robots sensing and moving in the plane



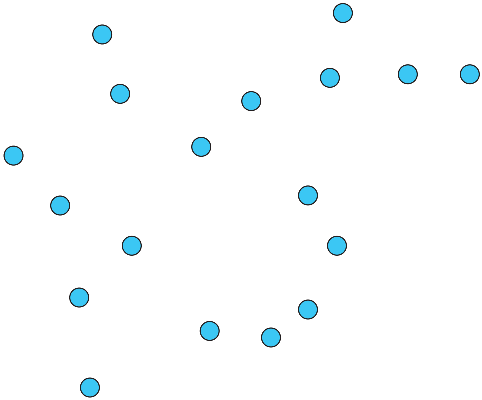
Different robots are activated asynchronously

Anonymous robots sensing and moving in the plane



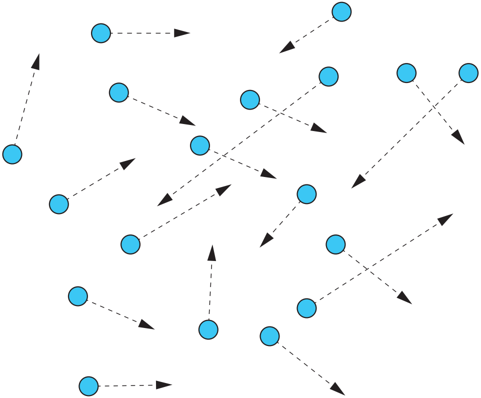
Different robots are activated asynchronously

Pattern Formation problem



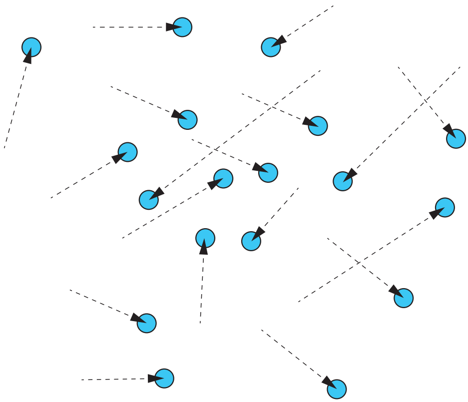
Problem: form a given pattern from any initial configuration

Pattern Formation problem



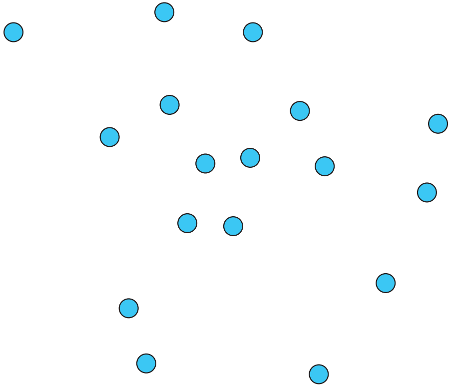
Problem: form a given pattern from any initial configuration

Pattern Formation problem



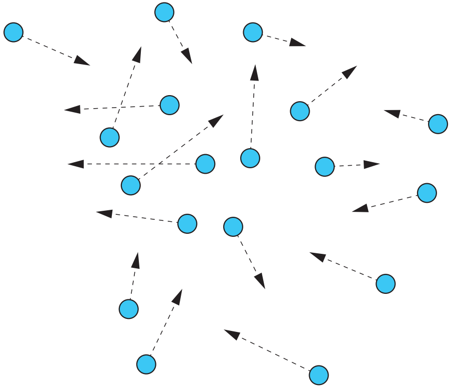
Problem: form a given pattern from any initial configuration

Pattern Formation problem



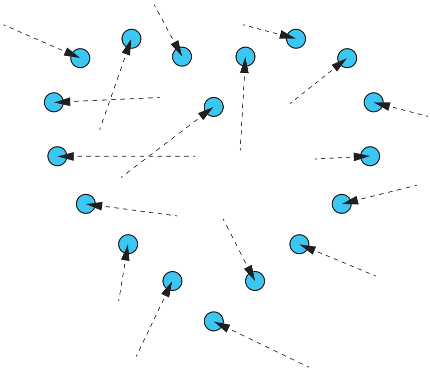
Problem: form a given pattern from any initial configuration

Pattern Formation problem



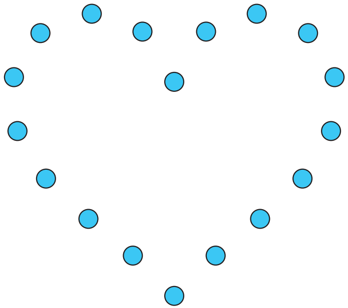
Problem: form a given pattern from any initial configuration

Pattern Formation problem



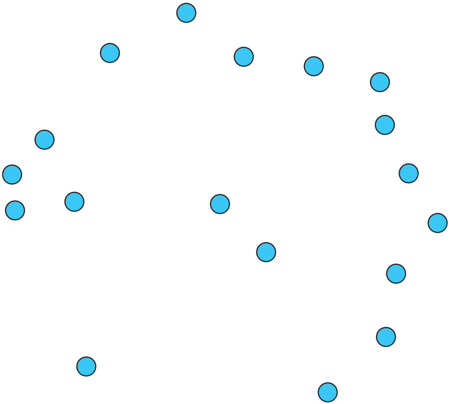
Problem: form a given pattern from any initial configuration

Pattern Formation problem



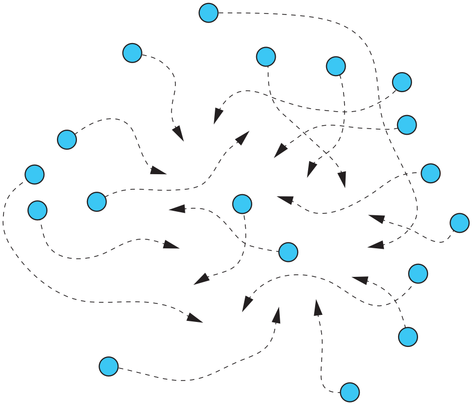
Problem: form a given pattern from any initial configuration

Pattern Formation problem



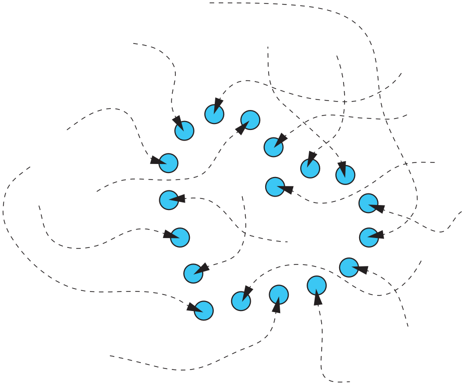
The pattern may be rotated, reflected, and scaled

Pattern Formation problem



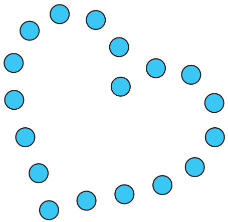
The pattern may be rotated, reflected, and scaled

Pattern Formation problem



The pattern may be rotated, reflected, and scaled

Pattern Formation problem



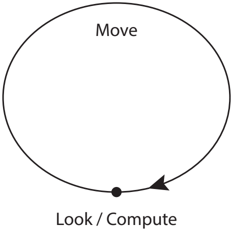
The pattern may be rotated, reflected, and scaled

Model definition

Robots are:

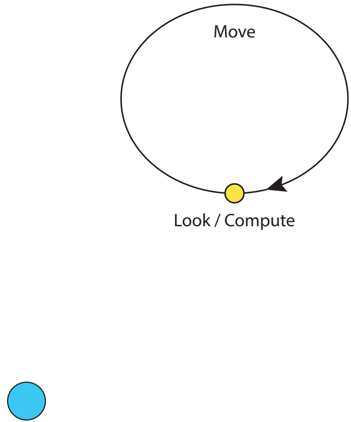
- **Dimensionless** (robots are modeled as geometric points)
- **Anonymous** (no unique identifiers)
- **Homogeneous** (the same algorithm is executed by all robots)
- **Autonomous** (no centralized control)
- **Oblivious** (no memory of past events)
- **Silent** (no explicit way of communicating)
- **Long-sighted** (complete visibility of all other robots)
- **Disoriented** (robots do not share a common reference frame, and a robot's reference frame may change from turn to turn)
  - No common unit distance
  - No common compass
  - No common notion of clockwise direction

Life cycles and asynchronicity



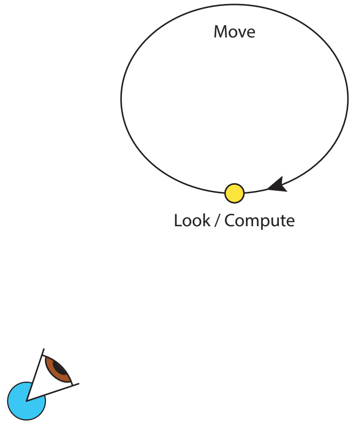
Each robot repeats a Look/Compute/Move cycle

Life cycles and asynchronicity



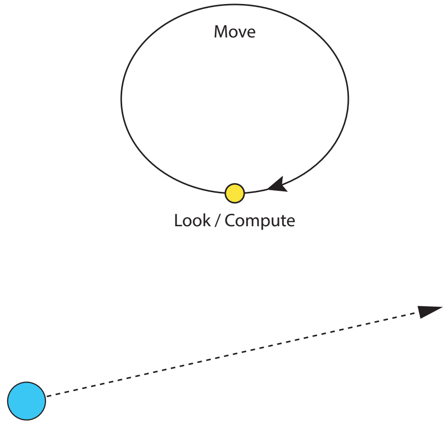
Each robot repeats a Look/Compute/Move cycle

Life cycles and asynchronicity



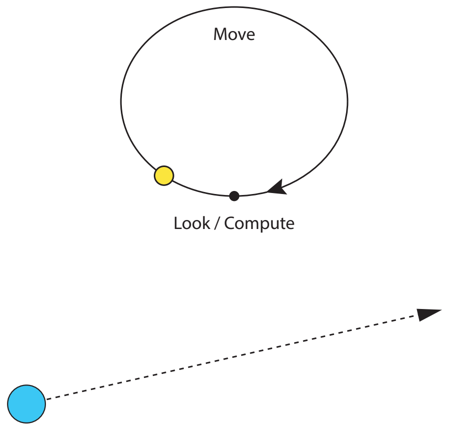
In a Look phase, an instantaneous snapshot is taken of all robots

Life cycles and asynchronicity



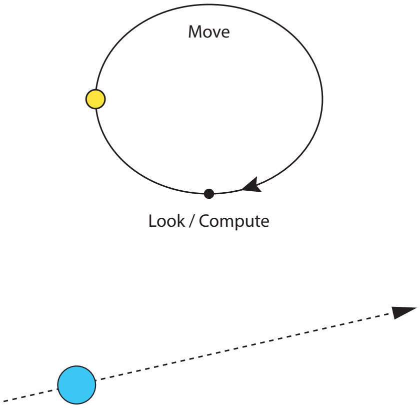
A destination point is computed as a function of the snapshot

Life cycles and asynchronicity



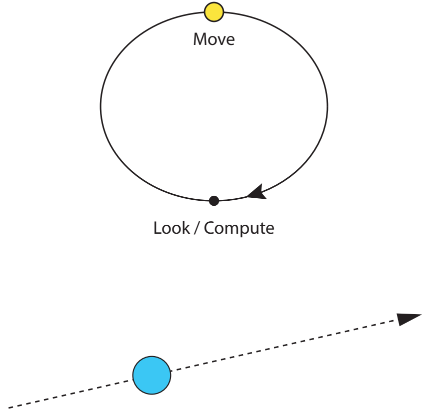
The destination point is approached with unpredictable speed

Life cycles and asynchronicity



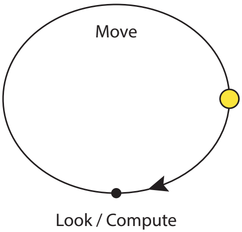
The destination point is approached with unpredictable speed

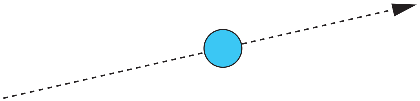
Life cycles and asynchronicity



The destination point is approached with unpredictable speed

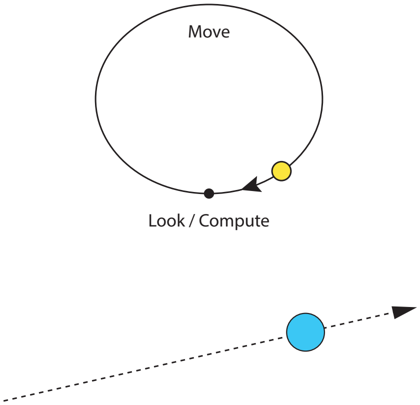
Life cycles and asynchronicity





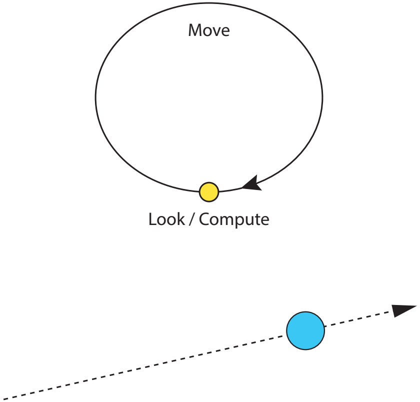
The destination point is approached with unpredictable speed

Life cycles and asynchronicity



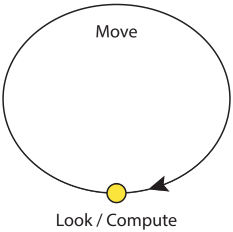
The destination point is approached with unpredictable speed

Life cycles and asynchronicity



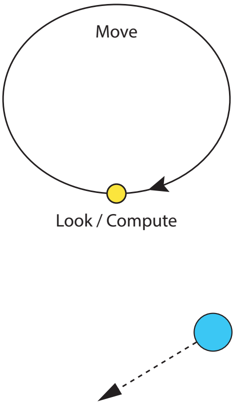
The robot may unpredictably stop before reaching the destination...

Life cycles and asynchronicity



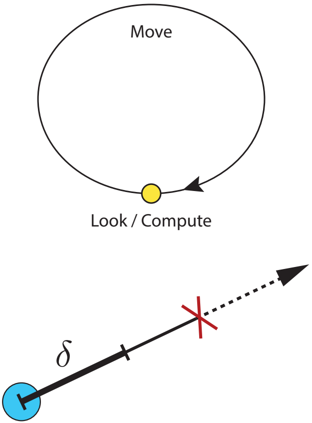
...and execute a new Look/Compute phase

Life cycles and asynchronicity



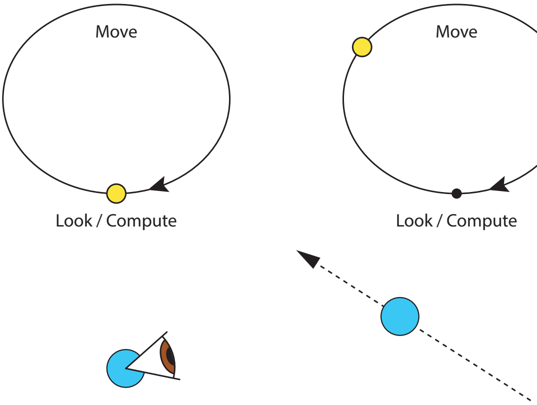
...and execute a new Look/Compute phase

Life cycles and asynchronicity



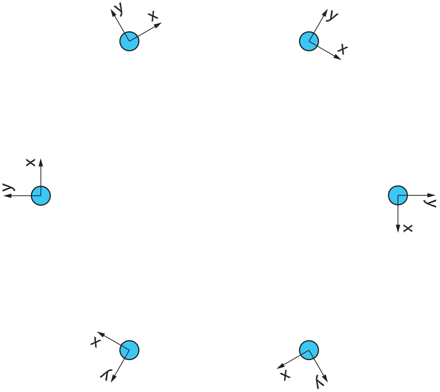
At each cycle, a robot is guaranteed to move by at least δ

Life cycles and asynchronicity



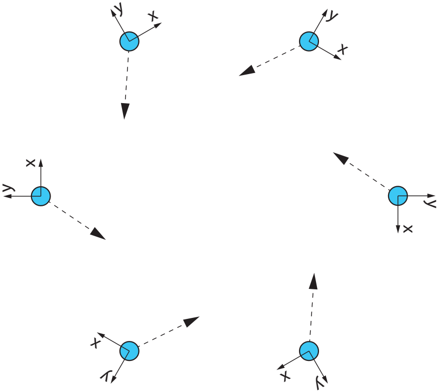
Different robots execute independent cycles, asynchronously

Pattern Formation problem: counterexample



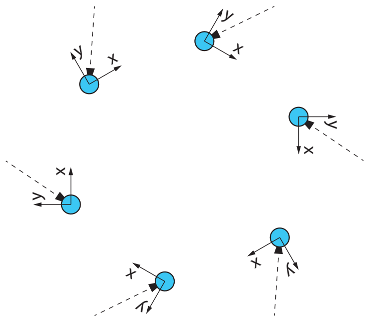
Let the initial configuration be rotationally symmetric

Pattern Formation problem: counterexample



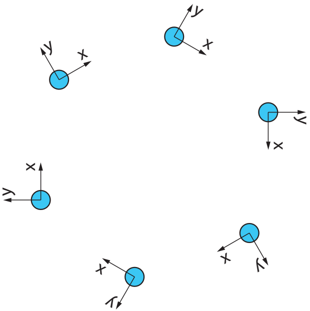
All robots have the same view and compute symmetric destinations

Pattern Formation problem: counterexample



If they are all activated synchronously, they remain symmetric

Pattern Formation problem: counterexample

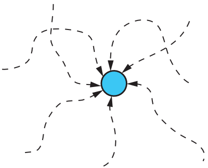


Hence Pattern Formation is *unsolvable* if the pattern is *asymmetric*

Pattern Formation problem: state of the art

No pattern is formable from every possible initial configuration, except:

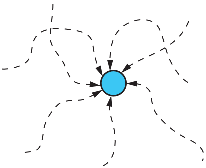
- **Single point** (aka Gathering problem)
  $\Longrightarrow$ Solved [Cieliebak-Flocchini-Prencipe-Santoro, 2012]

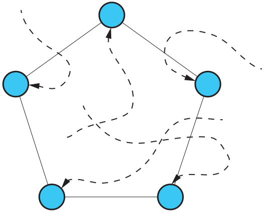


Pattern Formation problem: state of the art

No pattern is formable from every possible initial configuration, except:

- **Single point** (aka Gathering problem)
  $\Longrightarrow$ Solved [Cieliebak-Flocchini-Prencipe-Santoro, 2012]



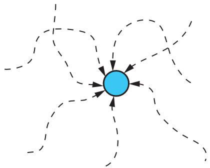


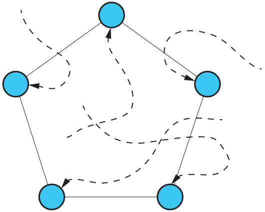
- **Regular polygon**
  $\Longrightarrow$ Solved... [Flocchini-Prencipe-Santoro-Viglietta, 2014–15]

Pattern Formation problem: state of the art

No pattern is formable from every possible initial configuration, except:

- **Single point** (aka Gathering problem)
  $\Longrightarrow$ Solved [Cieliebak-Flocchini-Prencipe-Santoro, 2012]

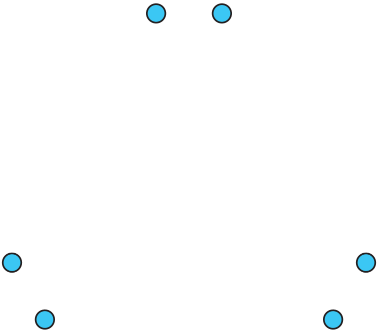




- **Regular polygon**
  $\Longrightarrow$ Solved... [Flocchini-Prencipe-Santoro-Viglietta, 2014–15]

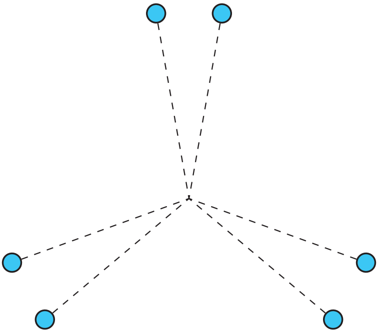
...*except for 4 robots!* (aka Square Formation problem)

General approach to forming a regular polygon



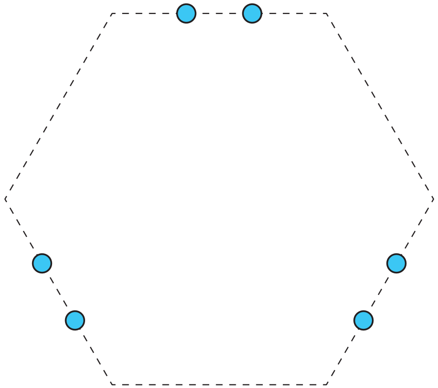
An important configuration is the *biangular* one

General approach to forming a regular polygon



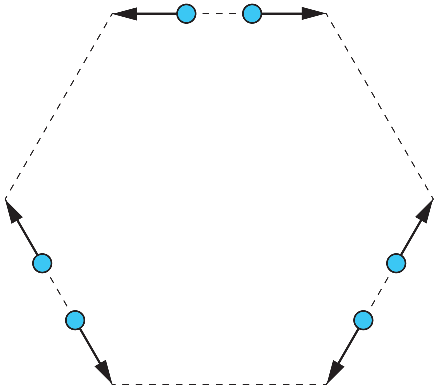
An important configuration is the *biangular* one

General approach to forming a regular polygon



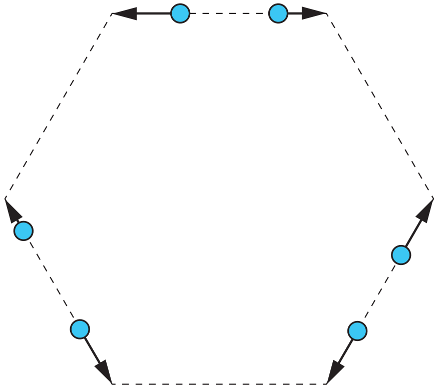
The general algorithm identifies a *supporting polygon*...

General approach to forming a regular polygon



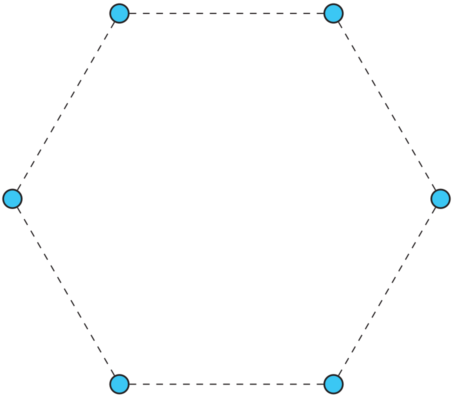
...And makes each robot move to the closest vertex

General approach to forming a regular polygon



As robots move, the supporting polygon is preserved

General approach to forming a regular polygon

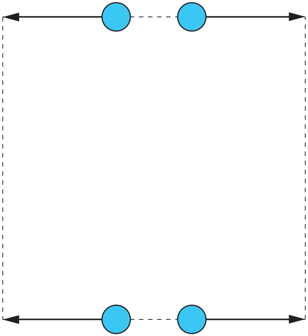


As robots move, the supporting polygon is preserved

Why the general approach fails with 4 robots

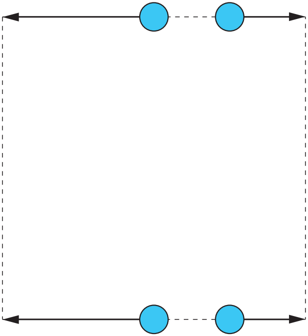
With 4 robots, biangular configurations are rectangles

Why the general approach fails with 4 robots



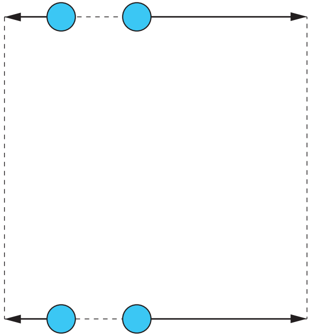
We can still identify a supporting square...

Why the general approach fails with 4 robots



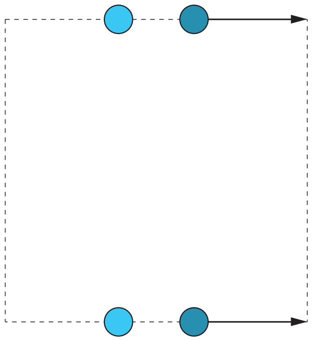
...But it is not unique!

Why the general approach fails with 4 robots



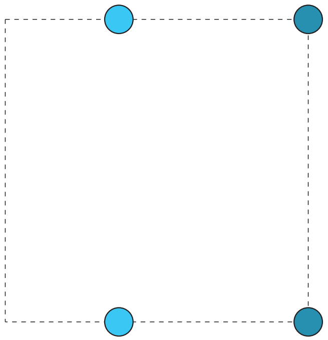
...But it is not unique!

Why the general approach fails with 4 robots



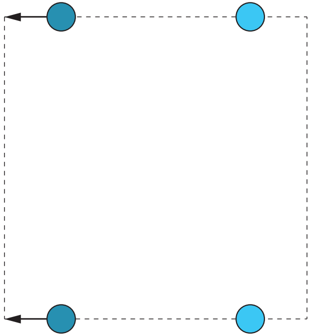
The “central” supporting polygon may be chosen...

Why the general approach fails with 4 robots



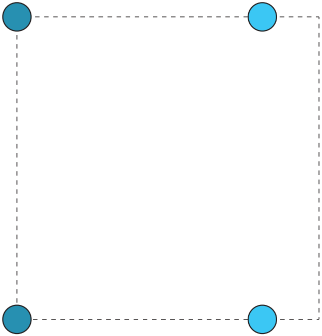
...But asynchronous robots may never manage to form a square

Why the general approach fails with 4 robots



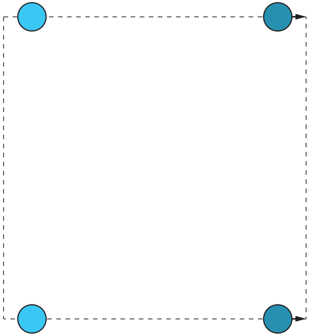
...But asynchronous robots may never manage to form a square

Why the general approach fails with 4 robots



...But asynchronous robots may never manage to form a square

Why the general approach fails with 4 robots

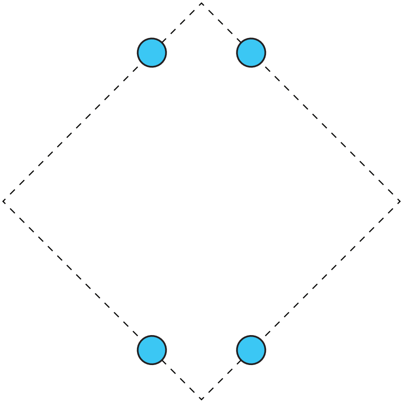


...But asynchronous robots may never manage to form a square

How to solve the rectangle

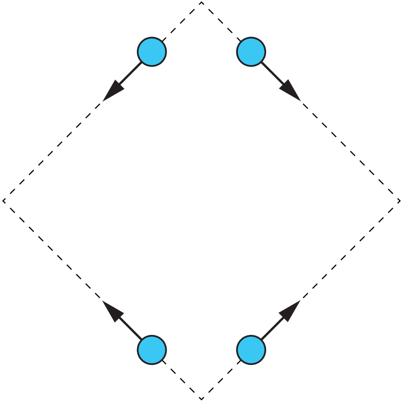
How do we solve the rectangular case?

How to solve the rectangle



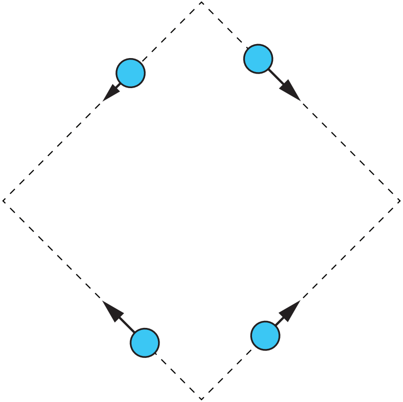
Choose a supporting square that is tilted by 45°...

How to solve the rectangle



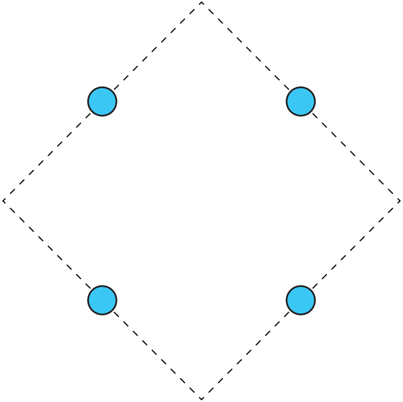
...And make the robots move to the midpoints of its edges

How to solve the rectangle



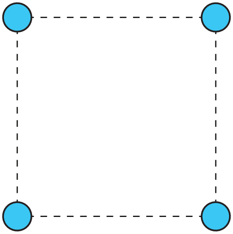
Again, the supporting square is preserved as the robots move

How to solve the rectangle



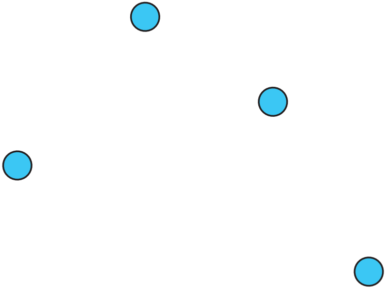
Again, the supporting square is preserved as the robots move

How to solve the rectangle



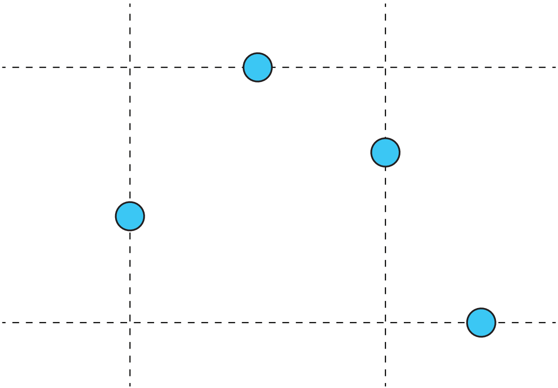
When they reach the midpoints, they form a square

Identifying the supporting square



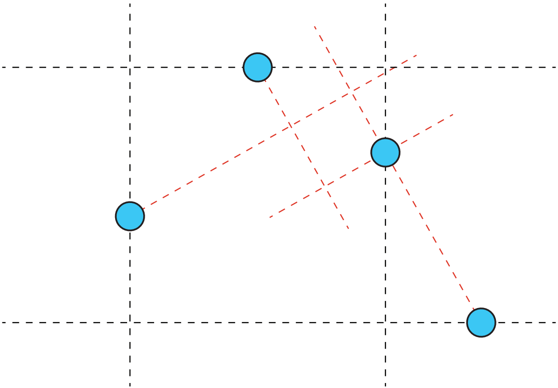
In general, we can also identify a supporting square...

Identifying the supporting square



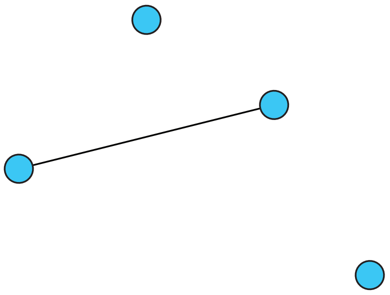
...Having a robot on each (extended) edge

Identifying the supporting square



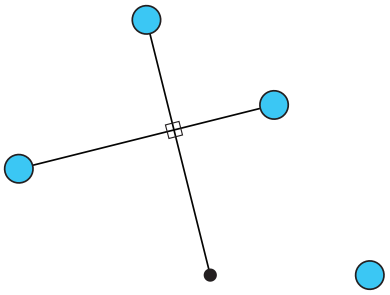
But once again, the supporting square is not unique!

Identifying the supporting square



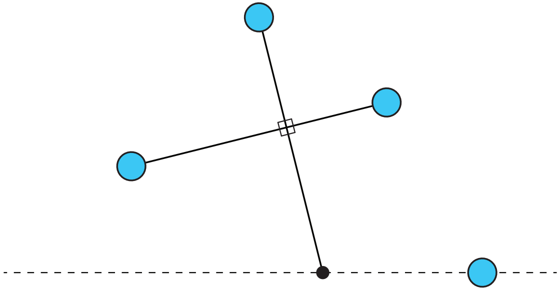
However, there is a geometric construction that identifies one

Identifying the supporting square



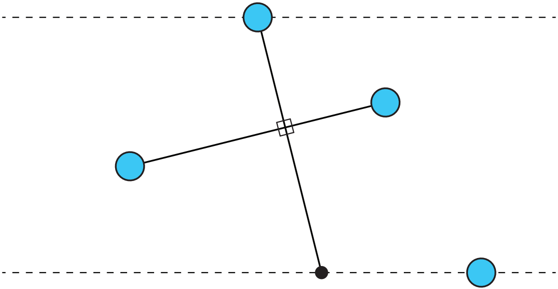
However, there is a geometric construction that identifies one

Identifying the supporting square



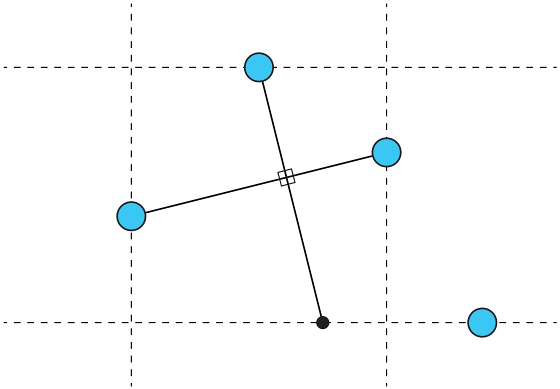
However, there is a geometric construction that identifies one

Identifying the supporting square



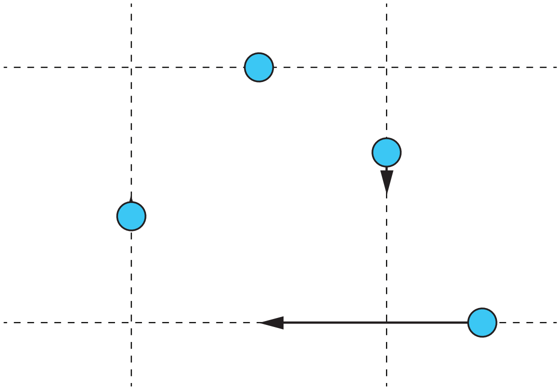
However, there is a geometric construction that identifies one

Identifying the supporting square



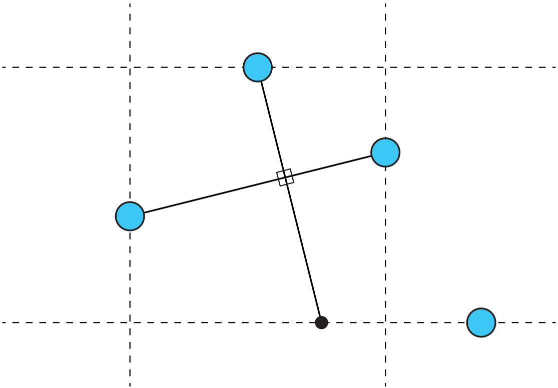
However, there is a geometric construction that identifies one

Identifying the supporting square



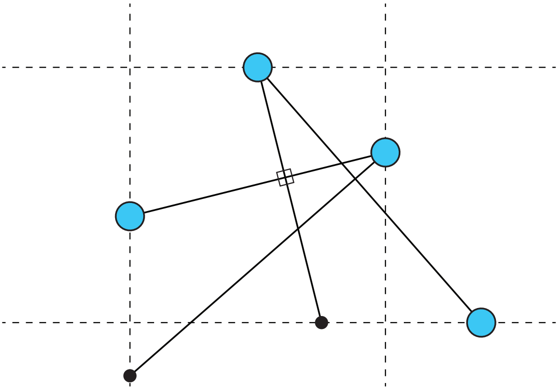
However, there is a geometric construction that identifies one

Identifying the supporting square



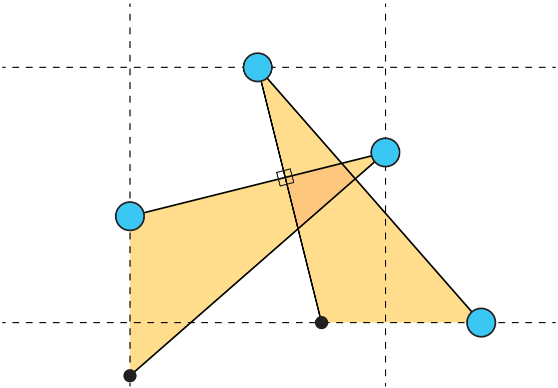
All robots automatically agree on the same supporting square!

Identifying the supporting square



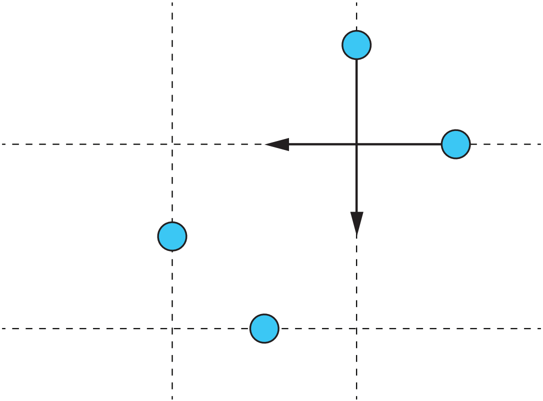
All robots automatically agree on the same supporting square!

Identifying the supporting square



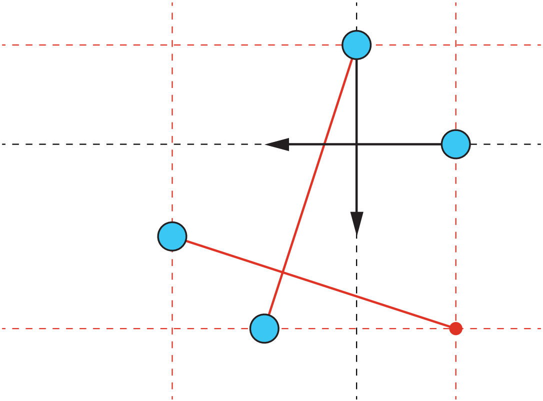
All robots automatically agree on the same supporting square!

Identifying the supporting square



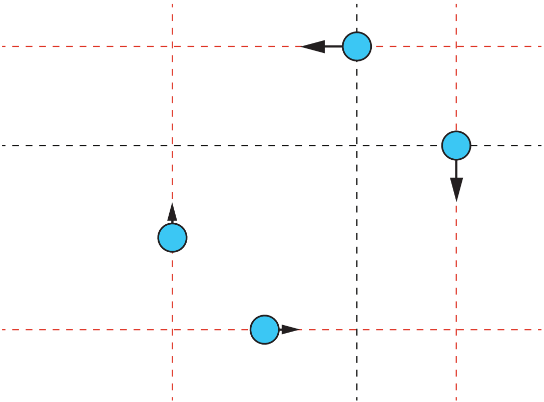
No two robots have intersecting pathways!

Identifying the supporting square



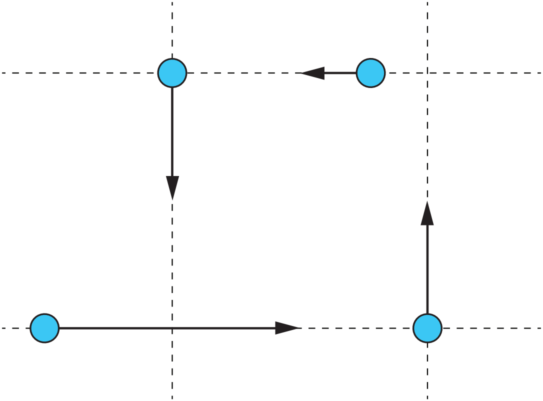
No two robots have intersecting pathways!

Identifying the supporting square



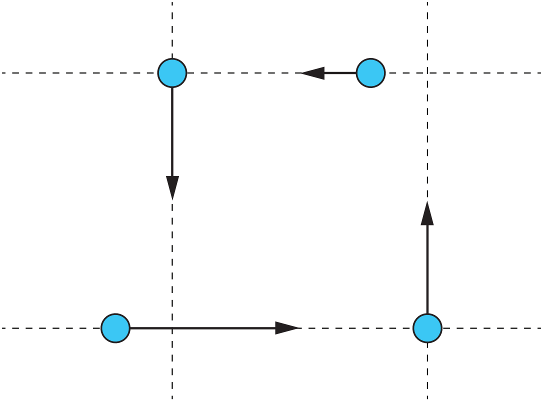
No two robots have intersecting pathways!

Problem: orthogonal diagonals



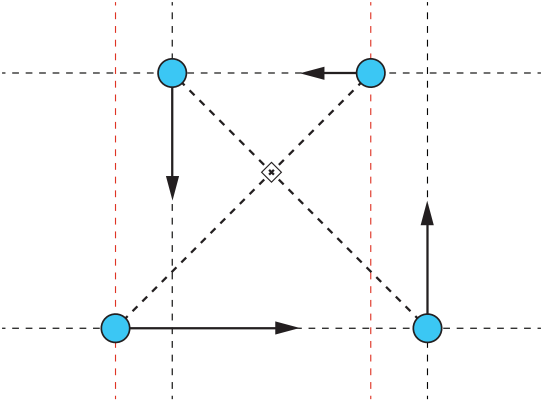
Suppose the two diagonals “accidentally” become orthogonal

Problem: orthogonal diagonals



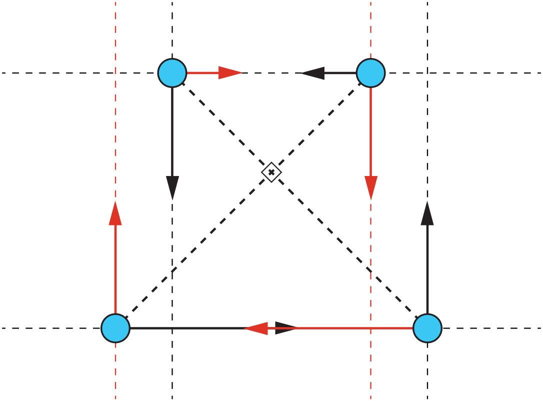
Suppose the two diagonals “accidentally” become orthogonal

Problem: orthogonal diagonals



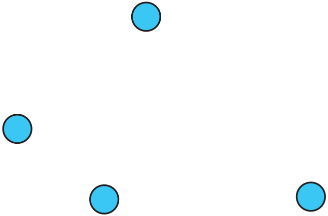
Then our construction does not work

Problem: orthogonal diagonals



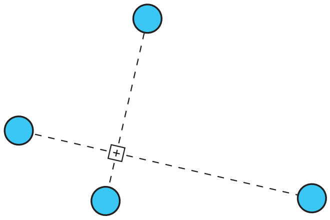
The robots may not agree on a supporting square

Special strategy for orthogonal diagonals



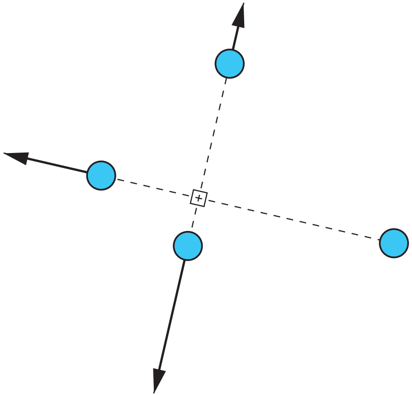
If the diagonals are orthogonal, we use a different approach

Special strategy for orthogonal diagonals



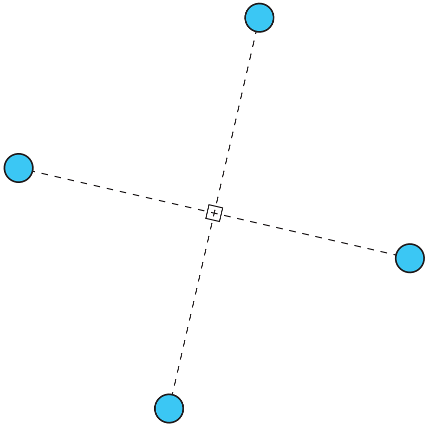
If the diagonals are orthogonal, we use a different approach

Special strategy for orthogonal diagonals



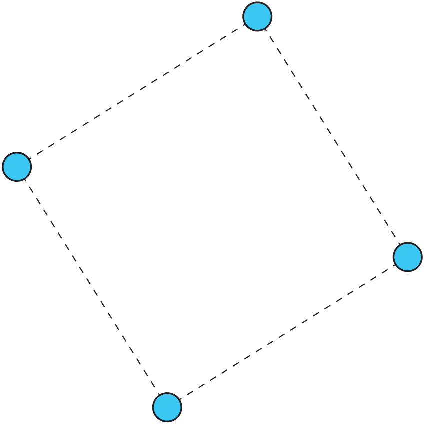
The robots that are closest to the center move away from it

Special strategy for orthogonal diagonals



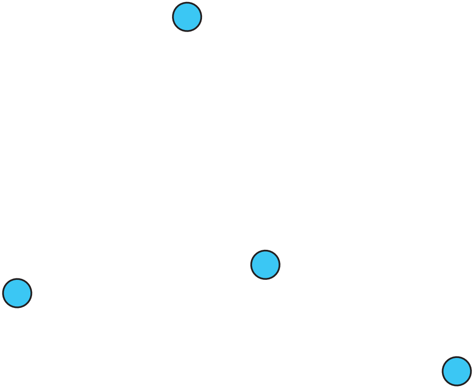
The robots that are closest to the center move away from it

Special strategy for orthogonal diagonals



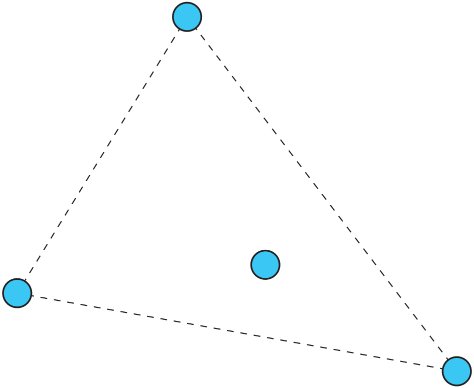
The robots that are closest to the center move away from it

Special strategy for non-convex configurations



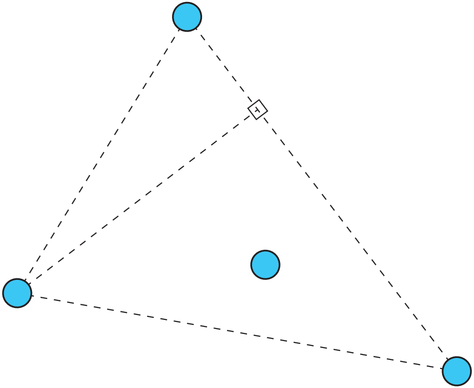
For non-convex configurations, our construction does not work...

Special strategy for non-convex configurations



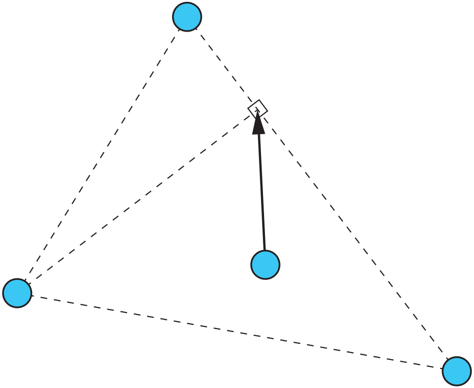
...Because the diagonals are not well defined

Special strategy for non-convex configurations



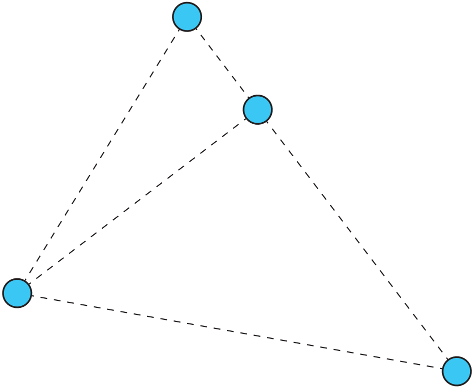
In this case, the internal robot moves...

Special strategy for non-convex configurations



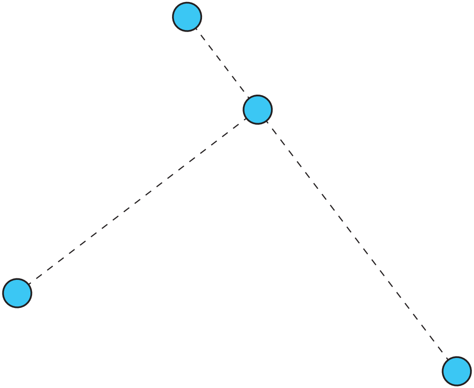
In this case, the internal robot moves...

Special strategy for non-convex configurations



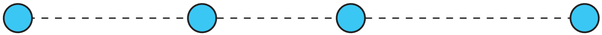
...So to make the diagonals orthogonal...

Special strategy for non-convex configurations



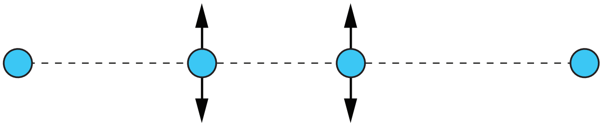
...And reduce the problem to the previous case

Special strategy for collinear configurations



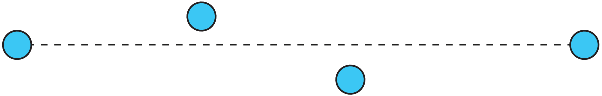
If the robots are collinear, the previous approach does not work

Special strategy for collinear configurations



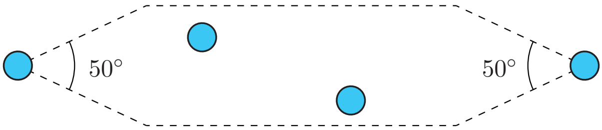
In this case, the internal robots move to either side of the line

Special strategy for collinear configurations



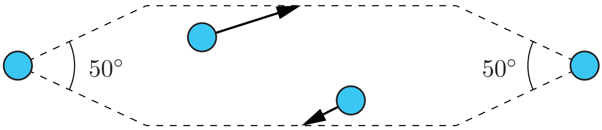
As they asynchronously move, their supporting square may change

Special strategy for collinear configurations



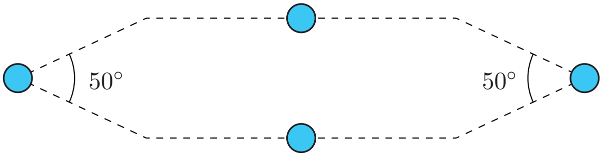
So we must identify a “safe region”, e.g., a thin hexagon

Special strategy for collinear configurations



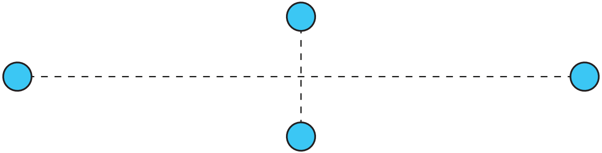
If the robots are in a thin hexagon, they follow a special algorithm

Special strategy for collinear configurations



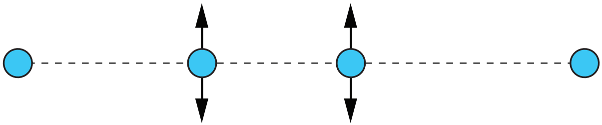
If they end up on opposite sides of the long diagonal...

Special strategy for collinear configurations



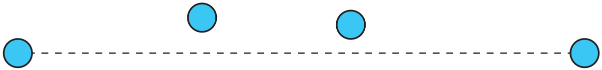
…We make them form a configuration with orthogonal diagonals

Special strategy for collinear configurations



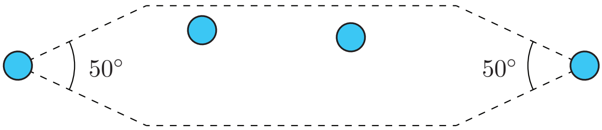
Otherwise, they move on two vertices and wait for each other

Special strategy for collinear configurations



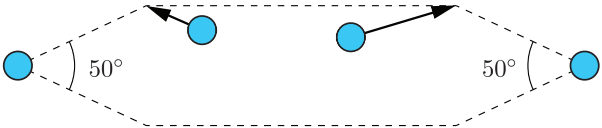
Otherwise, they move on two vertices and wait for each other

Special strategy for collinear configurations



Otherwise, they move on two vertices and wait for each other

Special strategy for collinear configurations

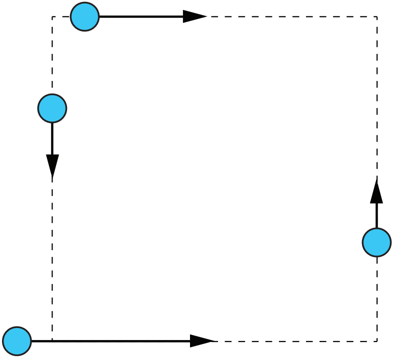


Otherwise, they move on two vertices and wait for each other

Special strategy for collinear configurations

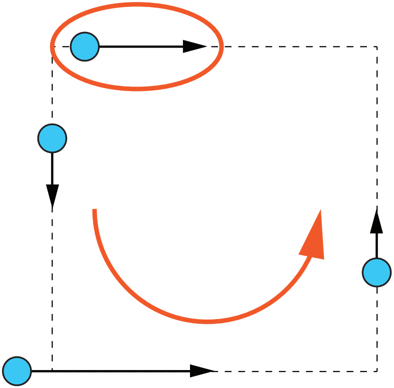
Now that they are not moving, they agree on a supporting square

General algorithm: one discordant robot



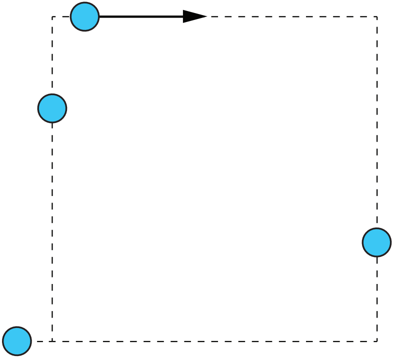
Suppose one robot is “discordant” with all the others

General algorithm: one discordant robot



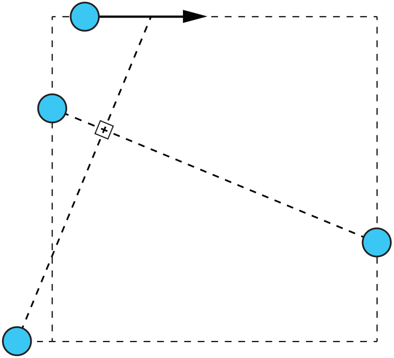
Suppose one robot is "discordant" with all the others

General algorithm: one discordant robot



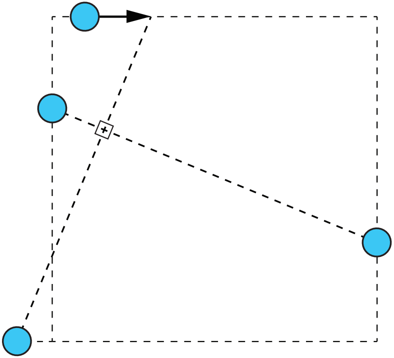
We let only the discordant robot move toward its final destination

General algorithm: one discordant robot



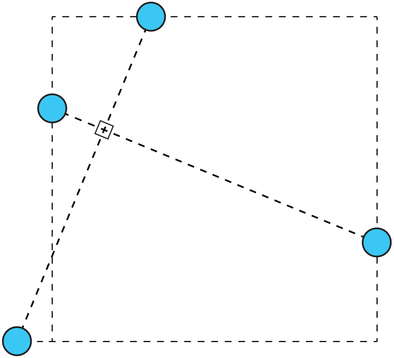
As it moves, it may cause the diagonals to become orthogonal!

General algorithm: one discordant robot



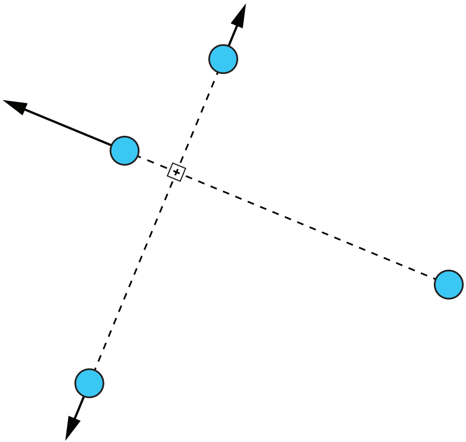
In this case, it has to stop at the point of orthogonality...

General algorithm: one discordant robot



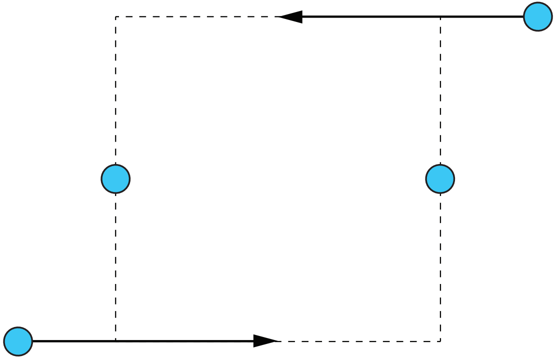
In this case, it has to stop at the point of orthogonality...

General algorithm: one discordant robot



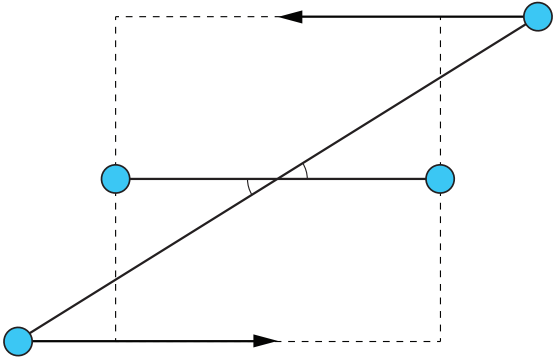
...So all robots will behave coherently, despite asynchronicity

General algorithm: two opposite concordant, two finished



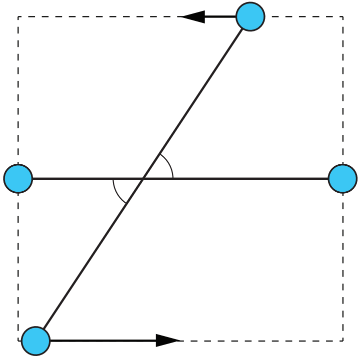
We let the two opposite robots move

General algorithm: two opposite concordant, two finished



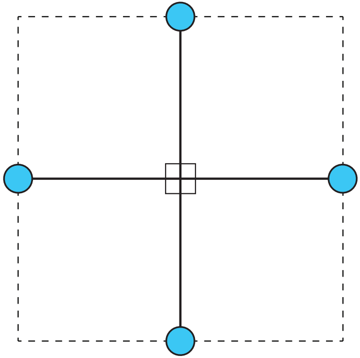
The diagonals can never become orthogonal by accident

General algorithm: two opposite concordant, two finished



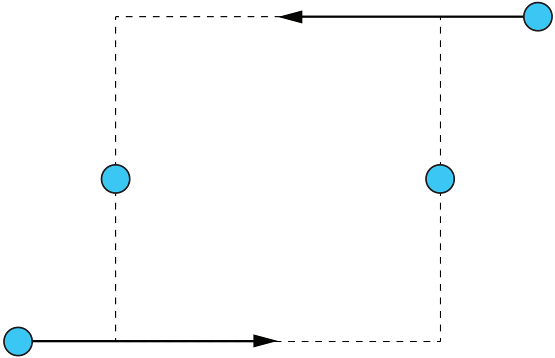
The diagonals can never become orthogonal by accident

General algorithm: two opposite concordant, two finished



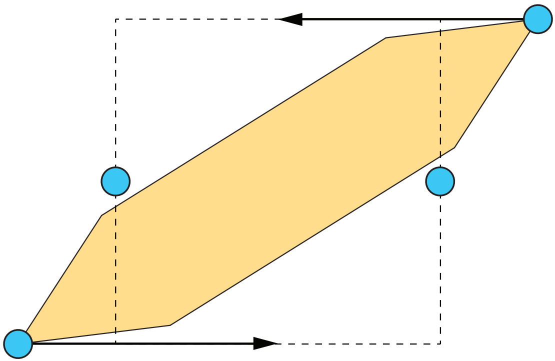
The diagonals can never become orthogonal by accident

General algorithm: two opposite concordant, two finished



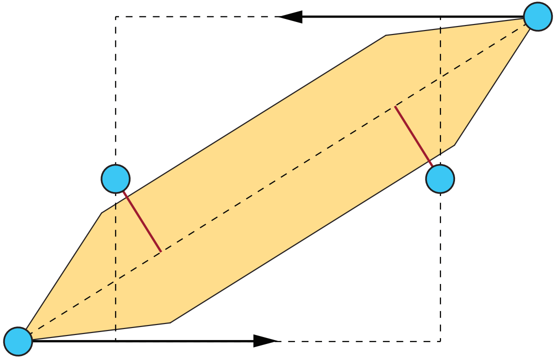
No thin hexagon can be formed by accident...

General algorithm: two opposite concordant, two finished



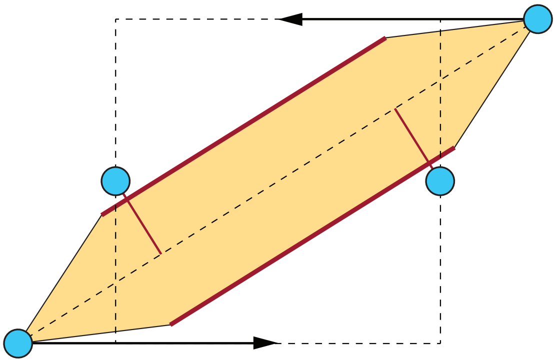
No thin hexagon can be formed by accident...

General algorithm: two opposite concordant, two finished



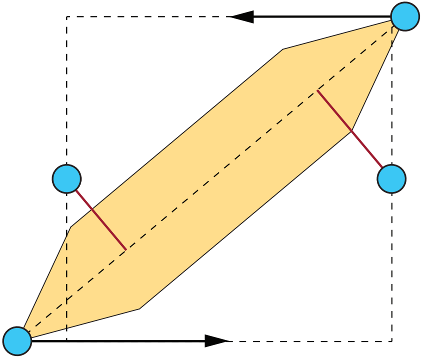
...Because the sum of distances from the long diagonal is too large

General algorithm: two opposite concordant, two finished



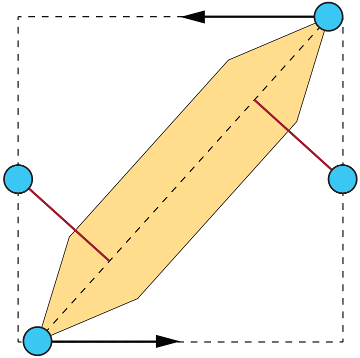
...Because the sum of distances from the long diagonal is too large

General algorithm: two opposite concordant, two finished



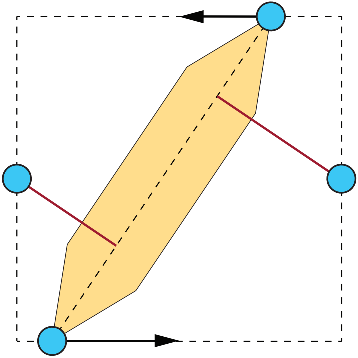
...Because the sum of distances from the long diagonal is too large

General algorithm: two opposite concordant, two finished



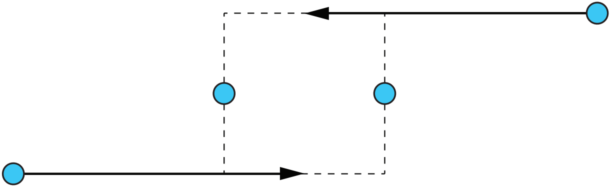
...Because the sum of distances from the long diagonal is too large

General algorithm: two opposite concordant, two finished

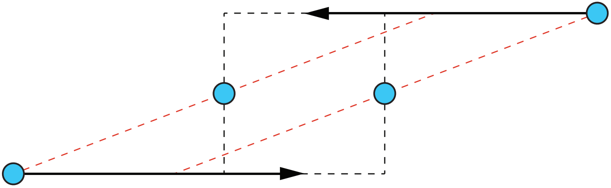


...Because the sum of distances from the long diagonal is too large

General algorithm: two opposite concordant, two finished

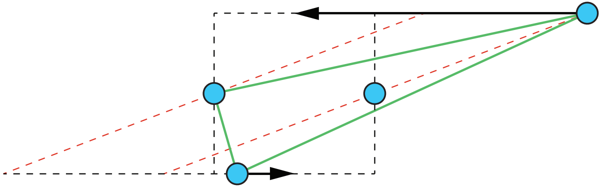


But the configuration may become non-convex by accident!



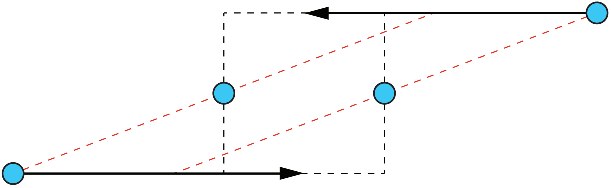
But the configuration may become non-convex by accident!

General algorithm: two opposite concordant, two finished



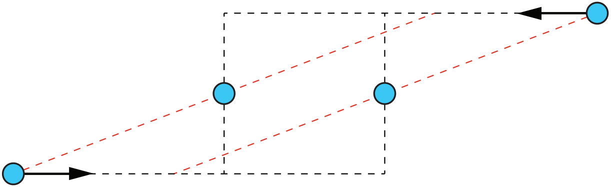
But the configuration may become non-convex by accident!

General algorithm: two opposite concordant, two finished

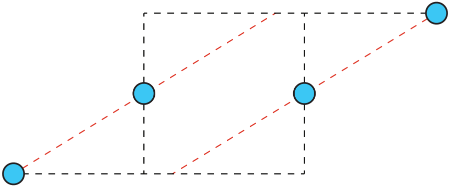


This can be prevented by making several shorter moves

General algorithm: two opposite concordant, two finished

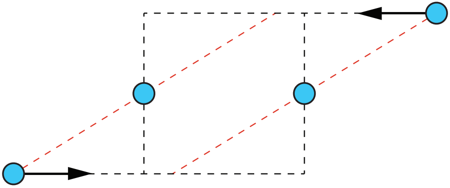


This can be prevented by making several shorter moves



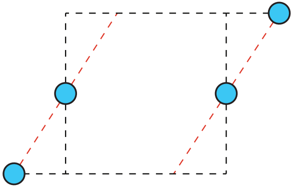
This can be prevented by making several shorter moves

General algorithm: two opposite concordant, two finished



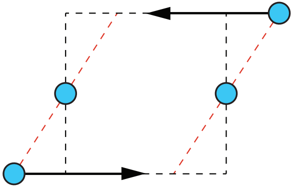
This can be prevented by making several shorter moves

General algorithm: two opposite concordant, two finished



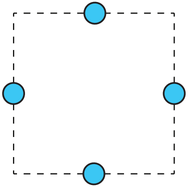
This can be prevented by making several shorter moves

General algorithm: two opposite concordant, two finished



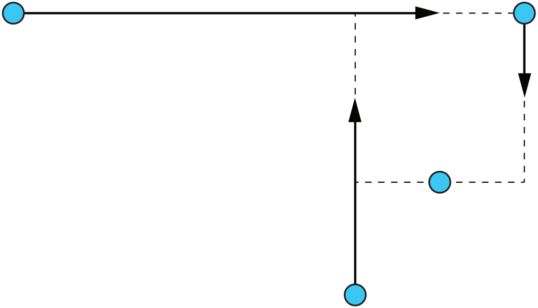
This can be prevented by making several shorter moves

General algorithm: two opposite concordant, two finished



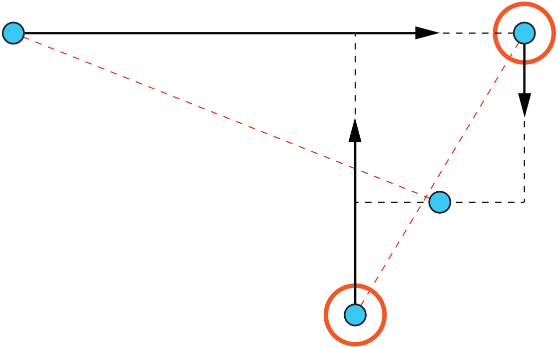
This can be prevented by making several shorter moves

General algorithm: all concordant



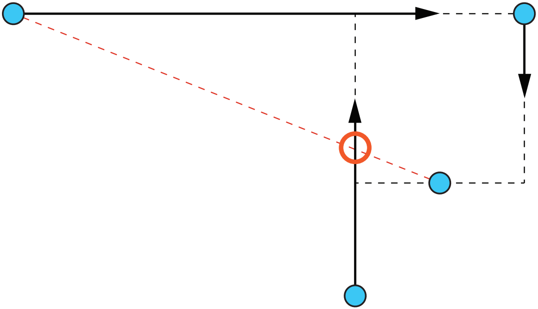
We let only the robots on the shortest diagonal move...

General algorithm: all concordant



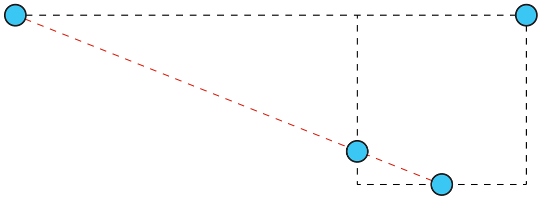
...Because it will remain the shortest as they move

General algorithm: all concordant



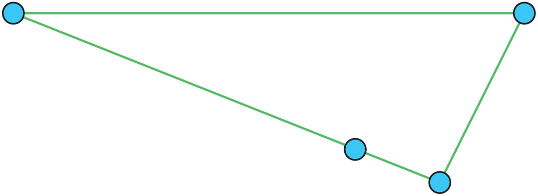
But one robot (not both!) may be “blocked” by the other diagonal

General algorithm: all concordant



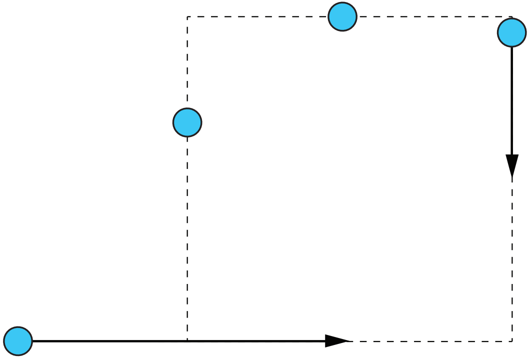
If so, only the blocked robot moves, and stops on the diagonal

General algorithm: all concordant



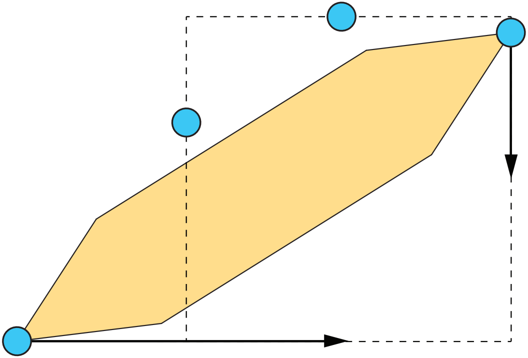
Then all robots behave coherently as in the non-convex case

General algorithm: two convergent robots



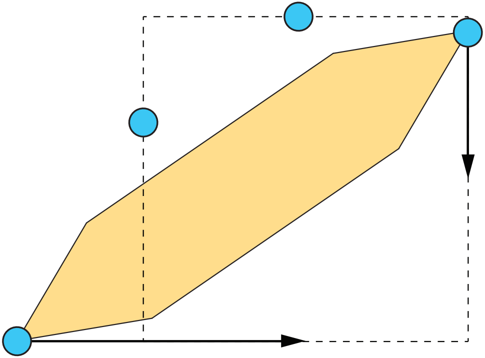
The convergent robots move, while the others wait

General algorithm: two convergent robots



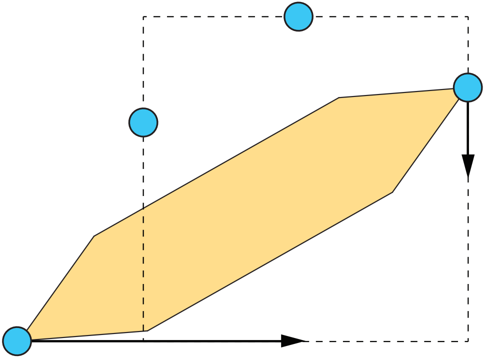
No thin hexagon can be formed by accident

General algorithm: two convergent robots



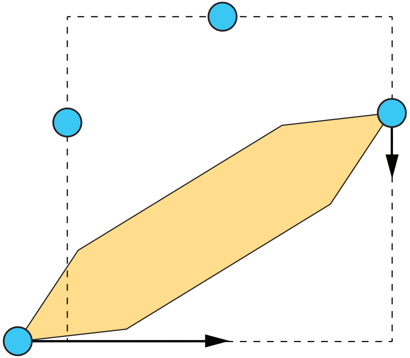
No thin hexagon can be formed by accident

General algorithm: two convergent robots



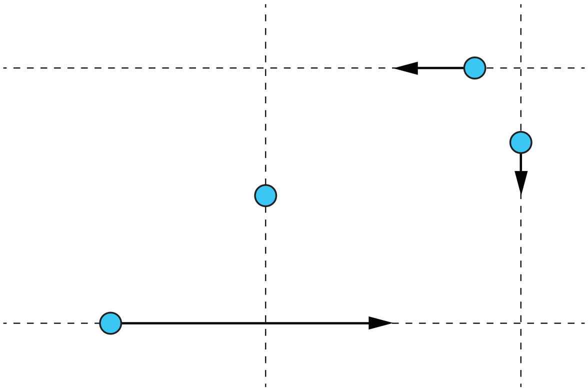
No thin hexagon can be formed by accident

General algorithm: two convergent robots



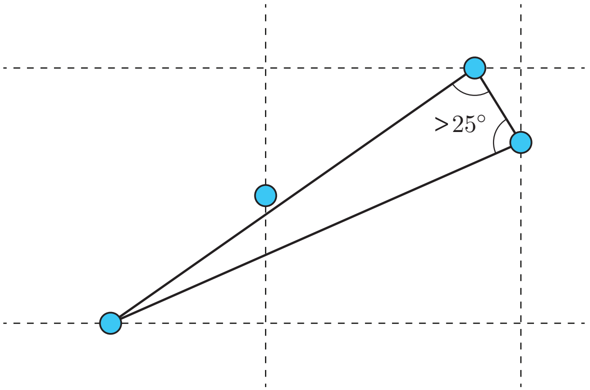
No thin hexagon can be formed by accident

General algorithm: last case



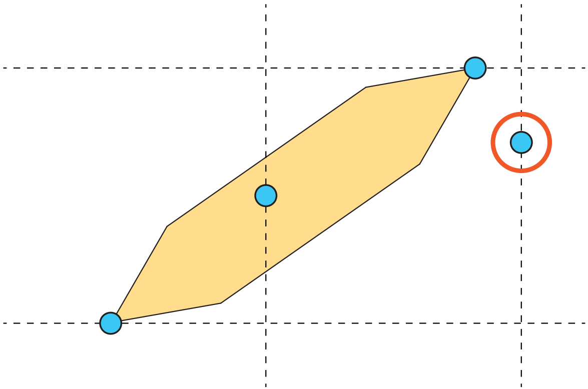
If only one robot is external...

General algorithm: last case



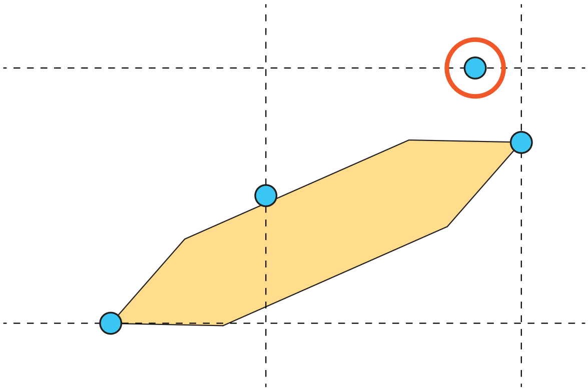
...The angles it forms with the two far robots are $> 25°$

General algorithm: last case



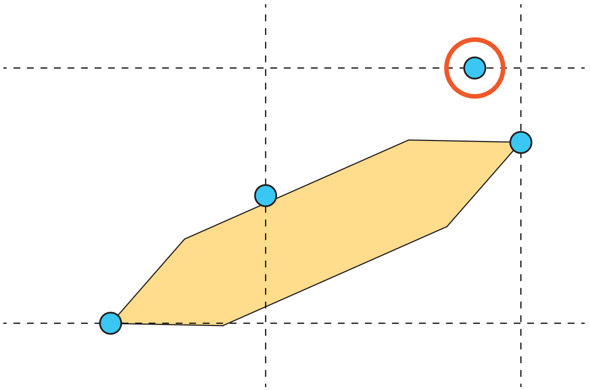
So a thin hexagon cannot be formed, because its angles are 50°

General algorithm: last case



So a thin hexagon cannot be formed, because its angles are 50°

General algorithm: last case



This yields a simple coordination protocol for the robots in all cases

Algorithm summary

The configuration is checked against each possible class,
<u>in the correct order!</u>

1. Orthogonal diagonals
2. Thin hexagon
3. Non-convex
4. All concordant
5. Two convergent, two divergent
6. Two divergent, two divergent
7. One discordant

Algorithm summary

The configuration is checked against each possible class,
<u>in the correct order!</u>

1. Orthogonal diagonals
2. Thin hexagon
3. Non-convex
4. All concordant
5. Two convergent, two divergent
6. Two divergent, two divergent
7. One discordant

Ensure that, when a class transition occurs,

- No robot is moving (to prevent inconsistent behaviors!)
- The resulting class has lower index (in the list above)

Algorithm summary

The configuration is checked against each possible class, <u>in the correct order!</u>

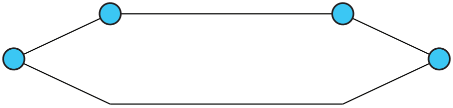
1. Orthogonal diagonals
2. Thin hexagon
3. Non-convex
4. All concordant
5. Two convergent, two divergent
6. Two divergent, two divergent
7. One discordant

Ensure that, when a class transition occurs,

- No robot is moving (to prevent inconsistent behaviors!)
- The resulting class has lower index (in the list above)

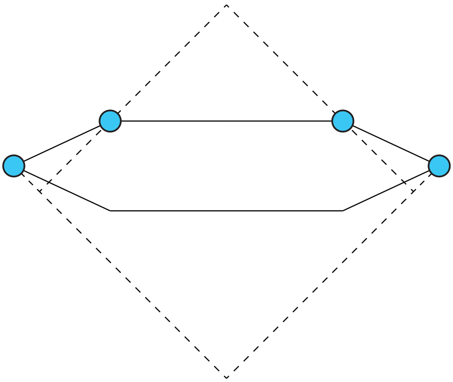
<u>The last rule is broken in only one case!</u>

Resolving the anomaly



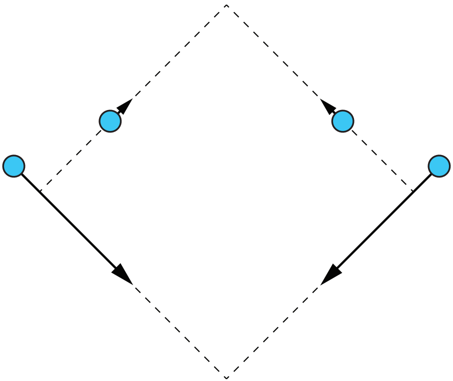
When all robots are on the same side of a thin hexagon...

Resolving the anomaly



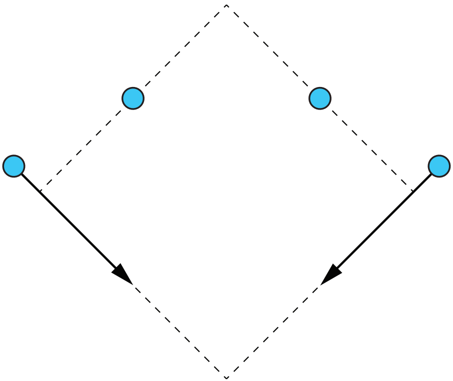
…They move to the vertices, and then apply the general algorithm

Resolving the anomaly



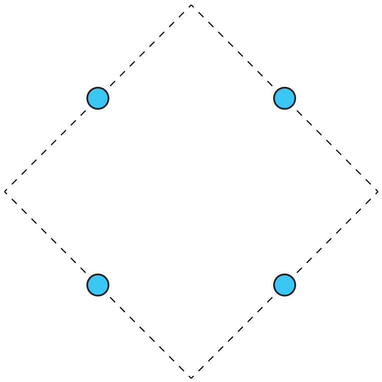
As a consequence, the internal robots move first

Resolving the anomaly



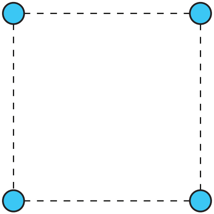
As a consequence, the internal robots move first

Resolving the anomaly



And finally the external robots move...

Resolving the anomaly



…Thus forming a square

Concluding remarks

The only solvable Pattern Formation problems for n robots are:

- **Single point** (except the case $n = 2$, which is unsolvable)
- **Regular n-gon** (now also for $n = 4$)

Concluding remarks

The only solvable Pattern Formation problems for n robots are:

- **Single point** (except the case $n = 2$, which is unsolvable)
- **Regular n-gon** (now also for $n = 4$)

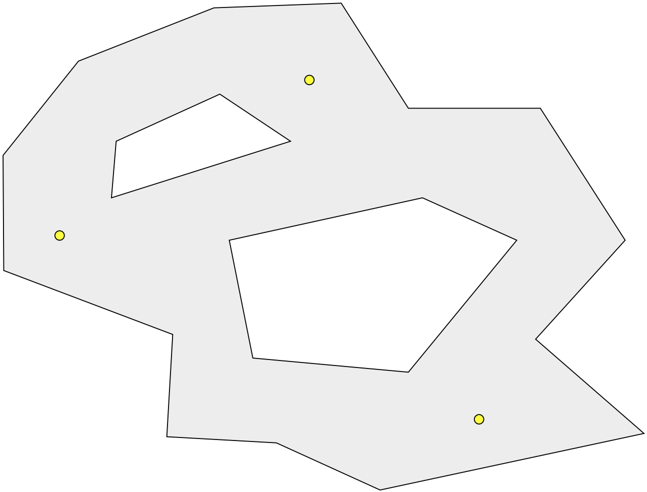
For $n > 2$, this is true even if

- Robots are fully synchronous
- Robots have a common notion of “clockwise” (chirality)
- Robots always reach their destination (rigidity)

$\implies$ For Pattern Formation problems, these features are computationally irrelevant!

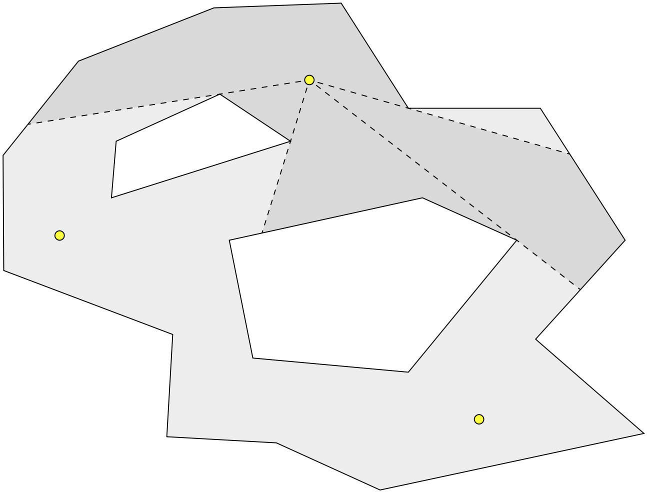
Meeting in a Polygon

Meeting Problem



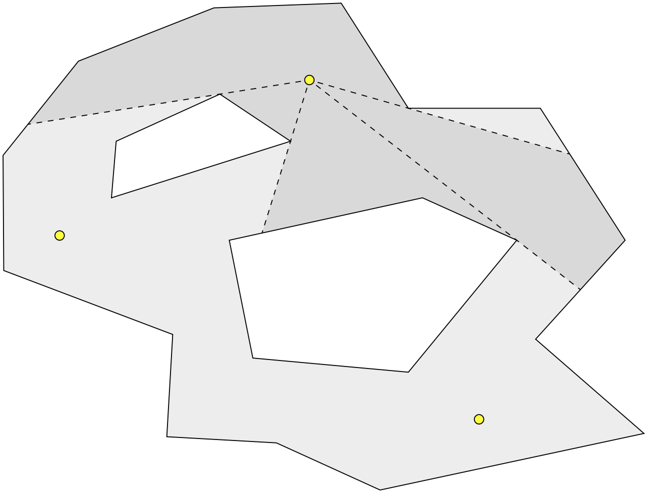
Setting: a polygon with some *searchers* in it.

Meeting Problem



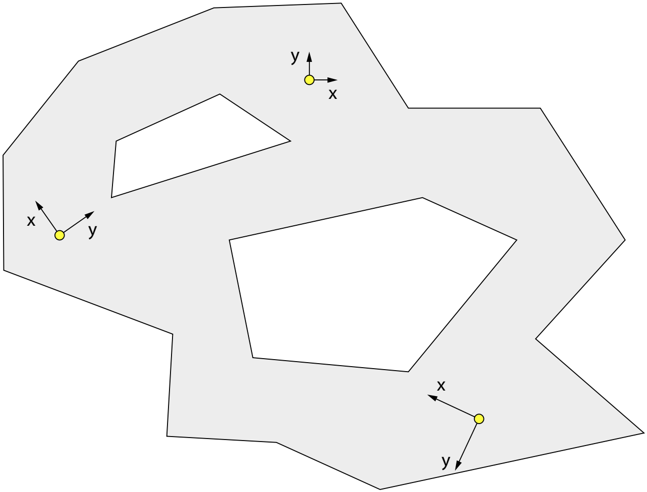
The polygon's edges obstruct visibility.

Meeting Problem



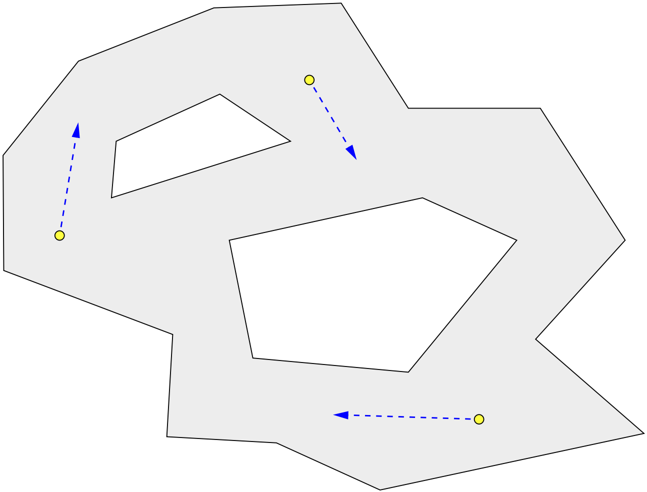
The invisible parts of the polygon are unknown to the searchers.

Meeting Problem



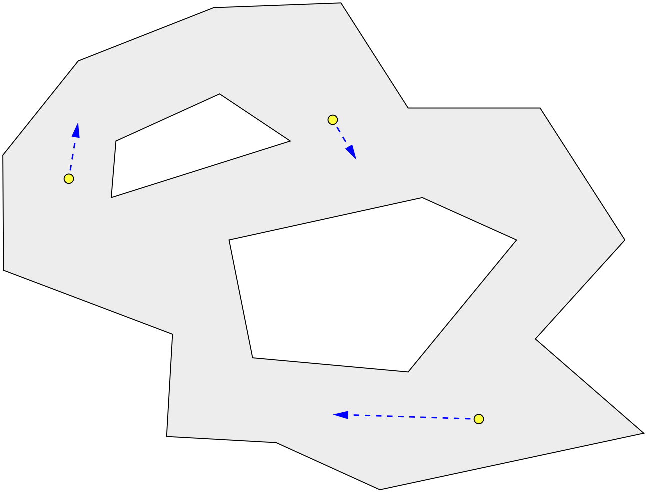
Each searcher has its own coordinate system.

Meeting Problem



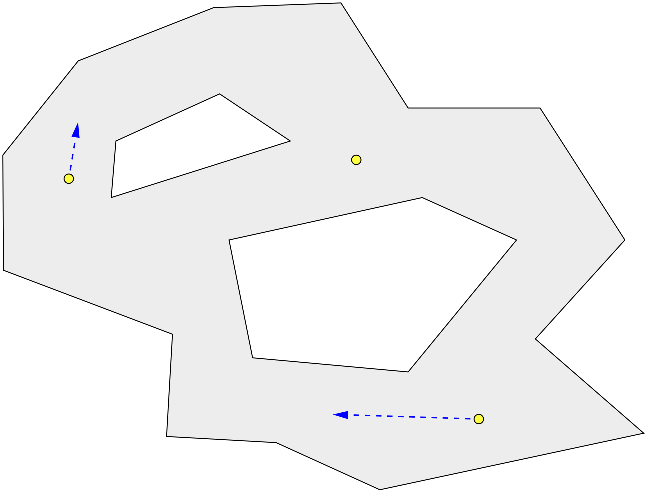
Searchers can move within the polygon.

Meeting Problem



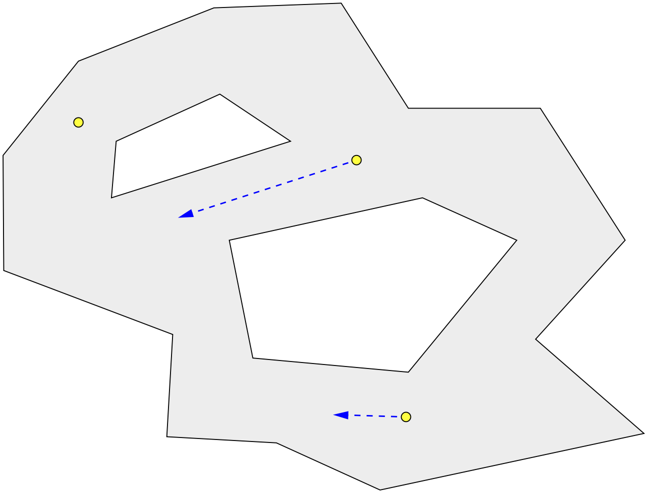
Movements are *asynchronous*.

Meeting Problem



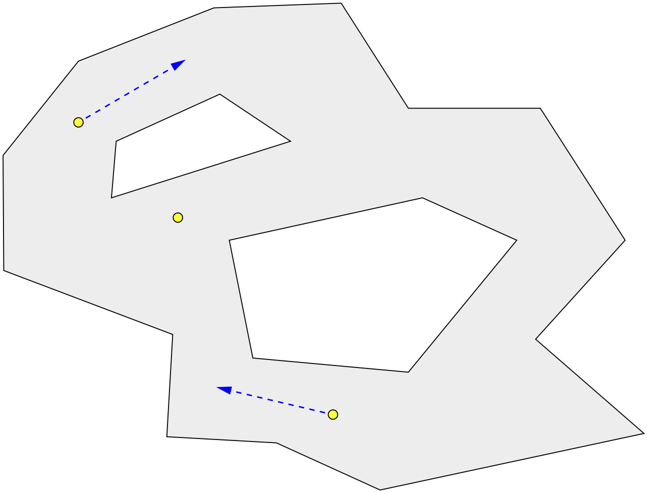
Movements are *asynchronous*.

Meeting Problem



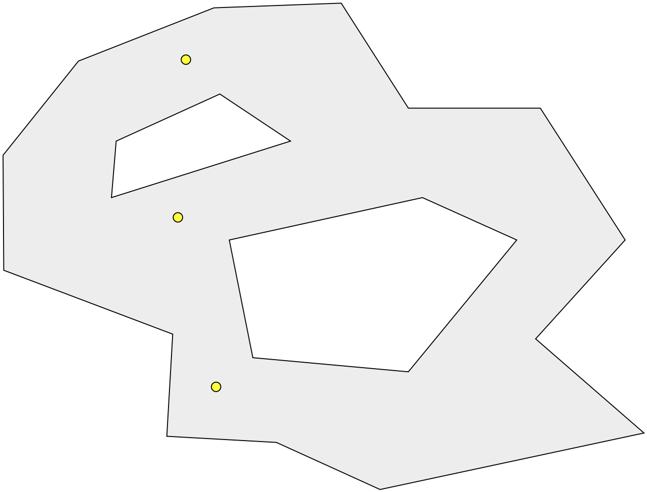
Movements are *asynchronous*.

Meeting Problem



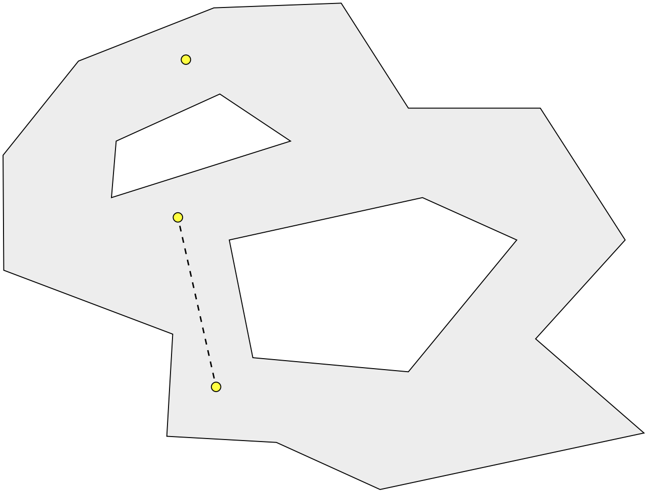
Searchers are *anonymous*: they all execute the same algorithm.

Meeting Problem



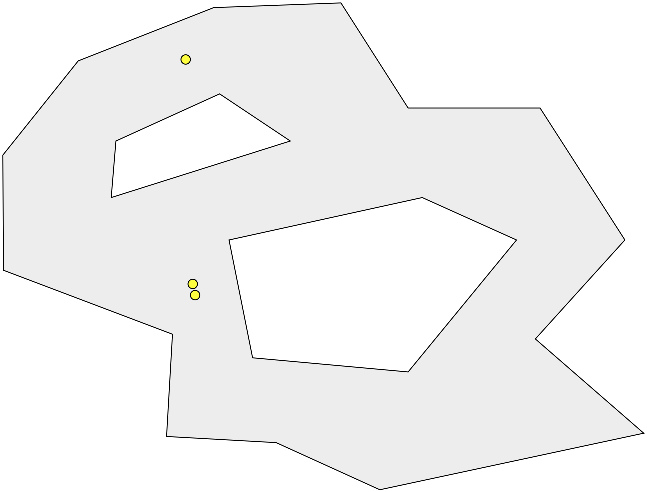
The goal is for any two searchers to see each other.

Meeting Problem



The goal is for any two searchers to see each other.

Meeting Problem



After, they can rendezvous and carry out more complex tasks.

Related Literature

"Static version" of the Meeting problem:

T. Shermer
Hiding people in polygons
Computing, 42(2):109–131, 1989

Meeting with unique ids or unlimited reliable memory:

J. Czyzowicz, D. Ilcinkas, A. Labourel, and A. Pelc
Asynchronous deterministic rendezvous in bounded terrains
Theoretical Computer Science, 412(50):6926–6937, 2011

J. Czyzowicz, A. Labourel, and A. Pelc
How to meet asynchronously (almost) everywhere
ACM Transactions on Algorithms, 8(4):37:1–37:14, 2012

J. Czyzowicz, A. Kosowski, and A. Pelc
Deterministic rendezvous of asynchronous bounded-memory agents in polygonal terrains
Theory of Computing Systems, 52(2):179–199, 2013

Y. Dieudonné, A. Pelc, and V. Villain
How to meet asynchronously at polynomial cost
SIAM Journal on Computing, 44(3):844–867, 2015

Summary

Results:

- If the polygon's *symmetricity* is σ, then $\sigma + 1$ searchers are always sufficient and sometimes necessary.

  (the symmetricity is the order of the rotation group of the polygon)

- If the polygon's center is not in a hole, 2 searchers are enough.

  (this includes all polygons with no holes)

Summary

Results:

- If the polygon's *symmetricity* is σ, then $\sigma + 1$ searchers are always sufficient and sometimes necessary.

  (the symmetricity is the order of the rotation group of the polygon)

- If the polygon's center is not in a hole, 2 searchers are enough.

  (this includes all polygons with no holes)

We establish these results for searchers with infinite faulty memory, and then we extend them to memoryless searchers.

Summary

Results:

- If the polygon's *symmetricity* is σ, then $\sigma + 1$ searchers are always sufficient and sometimes necessary.

  (the symmetricity is the order of the rotation group of the polygon)

- If the polygon's center is not in a hole, 2 searchers are enough.

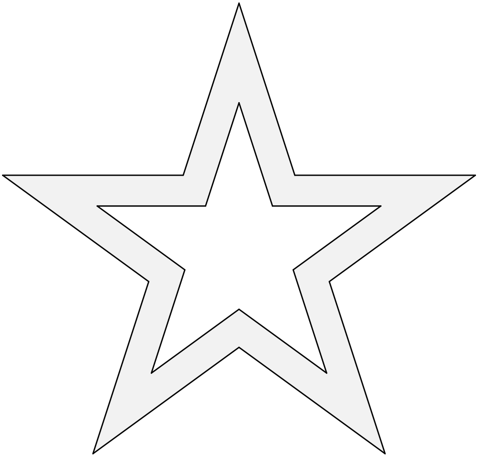
  (this includes all polygons with no holes)

We establish these results for searchers with <u>infinite faulty memory</u>, and then we extend them to <u>memoryless</u> searchers.

Techniques:

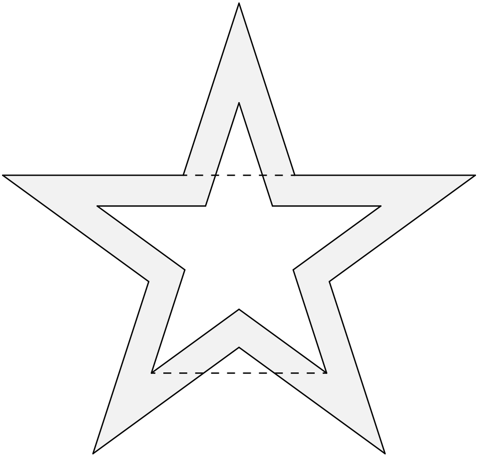
- Self-stabilizing map-construction algorithm
- Positional encoding of algebraic numbers

Negative Examples



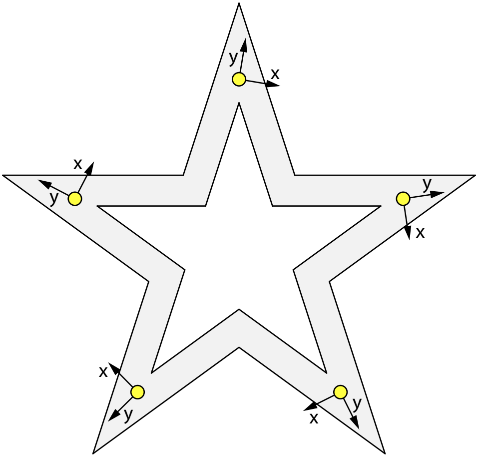
Consider a polygon of symmetricty σ with a large central hole.

Negative Examples



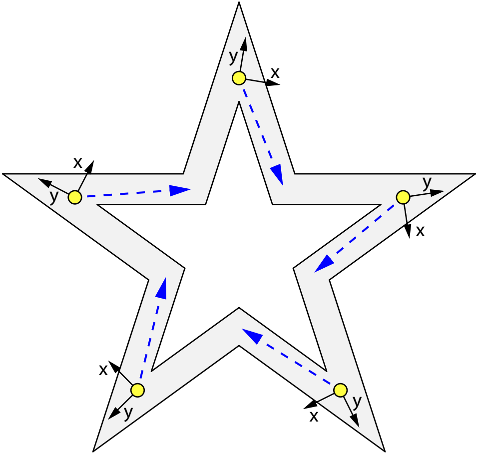
No two symmetric points can see each other.

Negative Examples



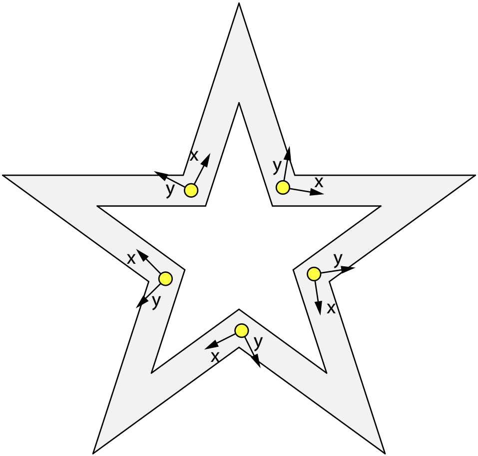
Place σ searchers in symmetric locations, oriented symmetrically.

Negative Examples



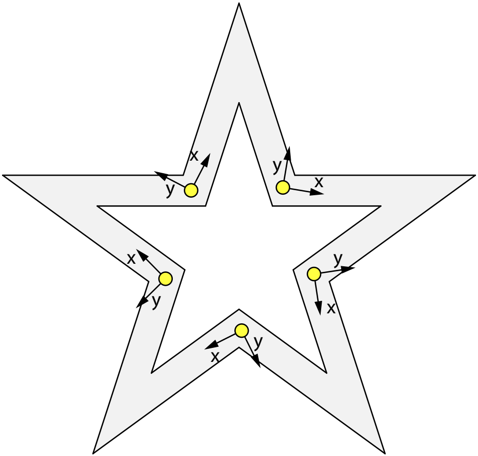
Their views are equal, so they compute symmetric destinations.

Negative Examples



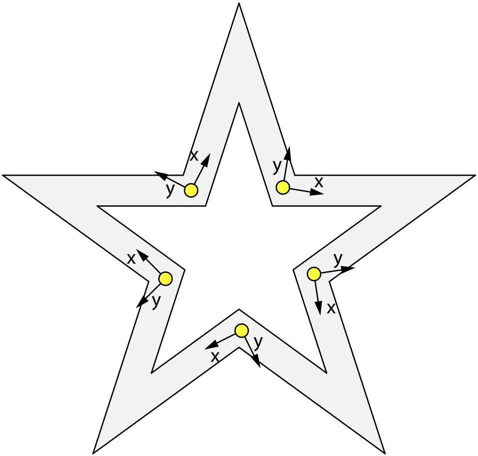
If they keep moving synchronously, they never see each other.

Negative Examples



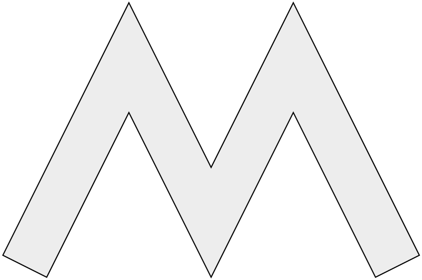
Theorem: in general, σ searchers are insufficient.

Negative Examples



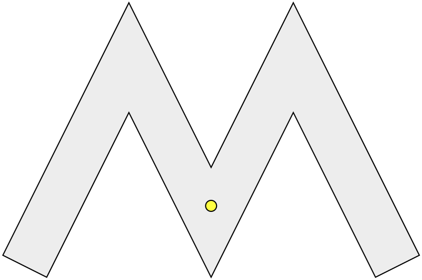
Claim: $\sigma + 1$ searchers are always sufficient.

Meeting with Faulty Memory



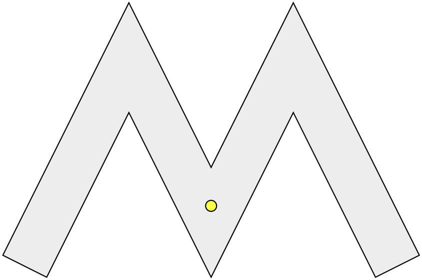
Traditionally, Meeting has been solved by identifying a *landmark*

Meeting with Faulty Memory



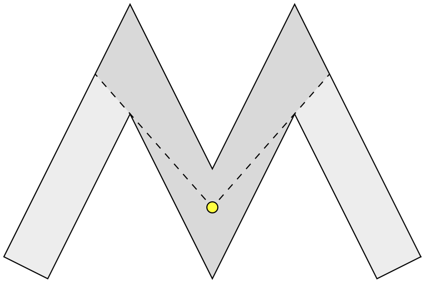
and making all searchers go there and wait for each other.

Meeting with Faulty Memory



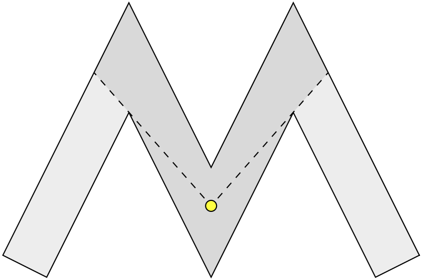
However, this does not work if searchers have faulty memory!

Meeting with Faulty Memory



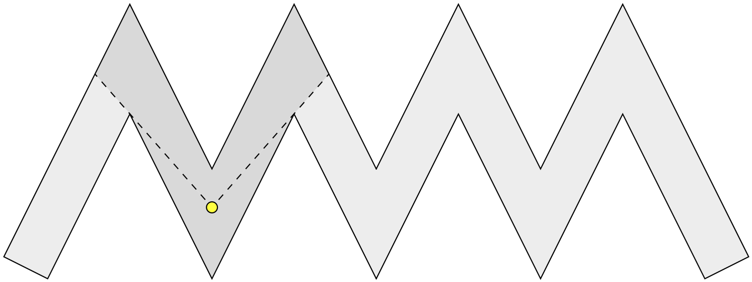
A searcher may believe to be in the polygon's landmark,

Meeting with Faulty Memory



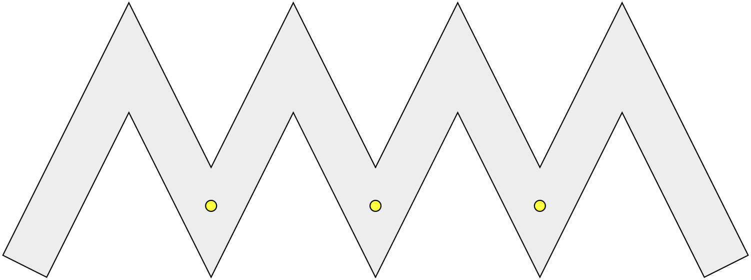
and its local view may support this belief.

Meeting with Faulty Memory



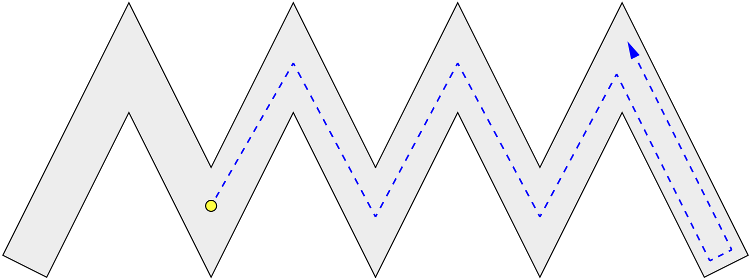
But the polygon may actually be different,

Meeting with Faulty Memory



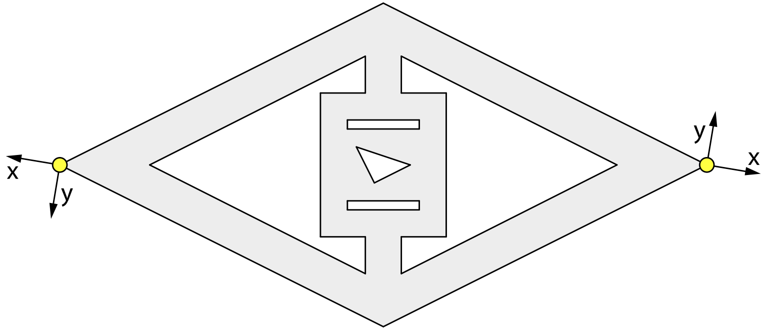
and different searchers may wait in different landmarks.

Meeting with Faulty Memory



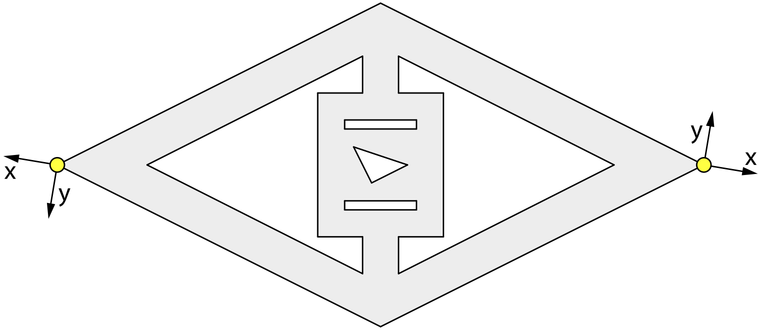
Observation: searchers must keep exploring the polygon.

Meeting with Faulty Memory



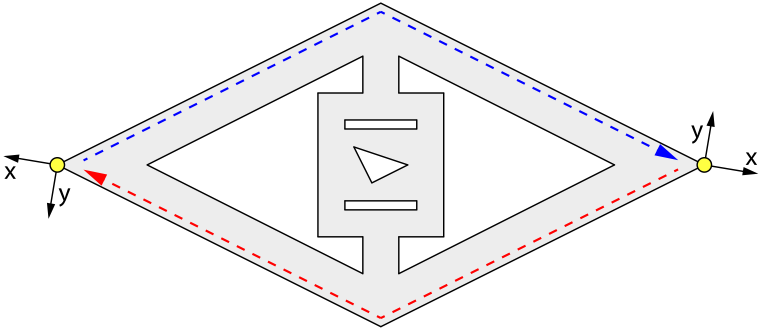
In this polygon, the only asymmetric element is the central hole.

Meeting with Faulty Memory



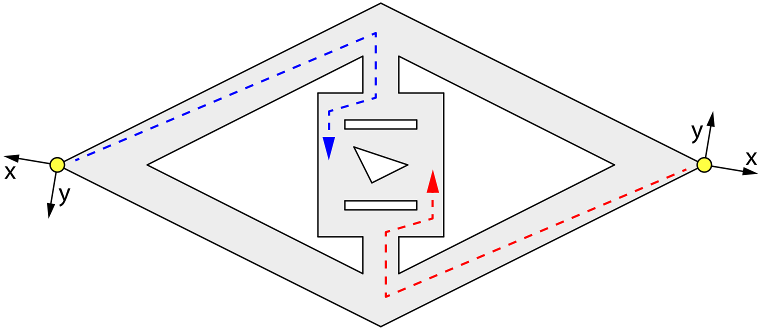
Its symmetricity is 1, but it “looks” 2 from the outer perimeter.

Meeting with Faulty Memory



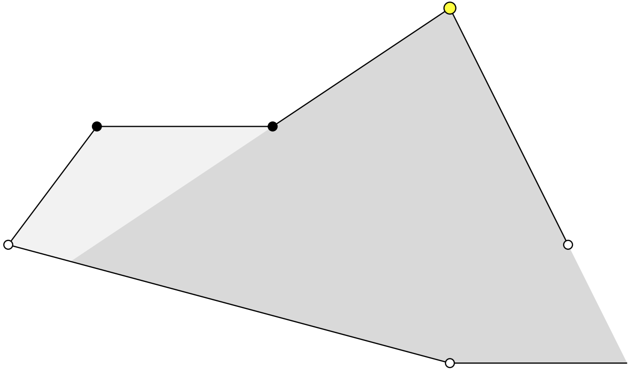
If the searchers do not explore the center, they cannot meet.

Meeting with Faulty Memory



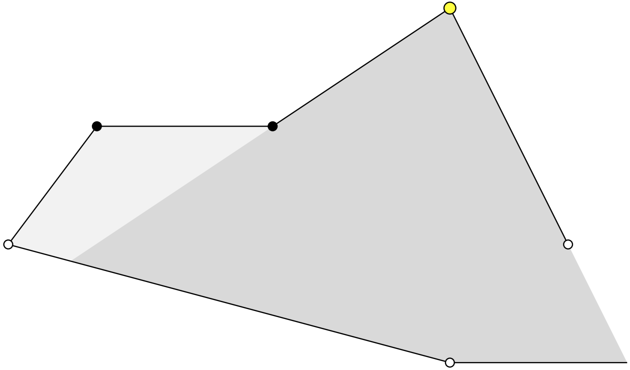
Observation: searchers must explore every hole of the polygon.

Basic Algorithm: EXPLORE Phase



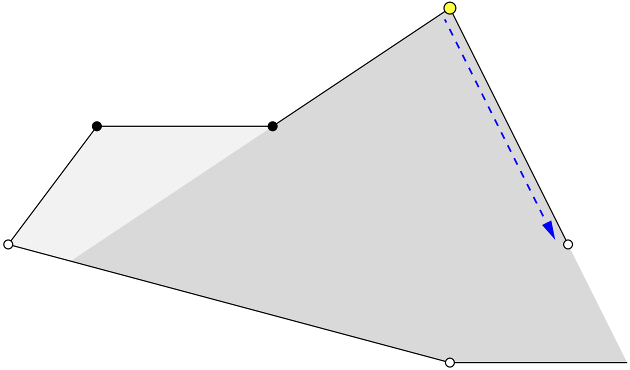
To begin with, assume searchers have infinite memory.

Basic Algorithm: EXPLORE Phase



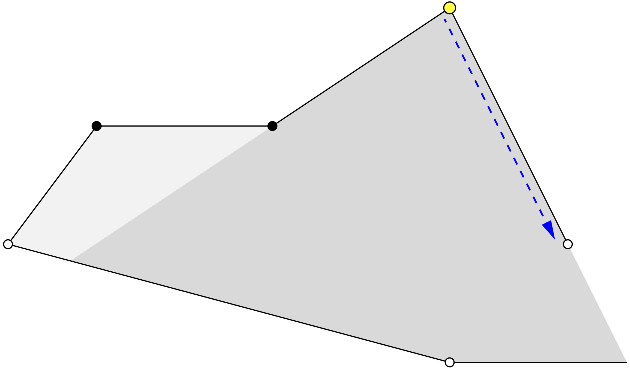
So they can build a partial map of the polygon as they explore it.

Basic Algorithm: EXPLORE Phase



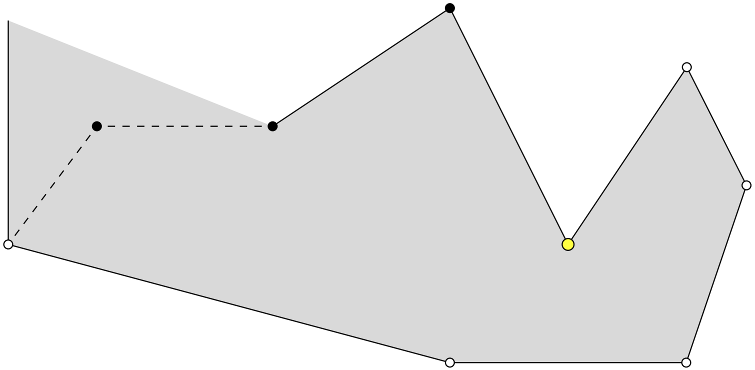
A searcher keeps a list of the vertices it has seen but not visited.

Basic Algorithm: EXPLORE Phase



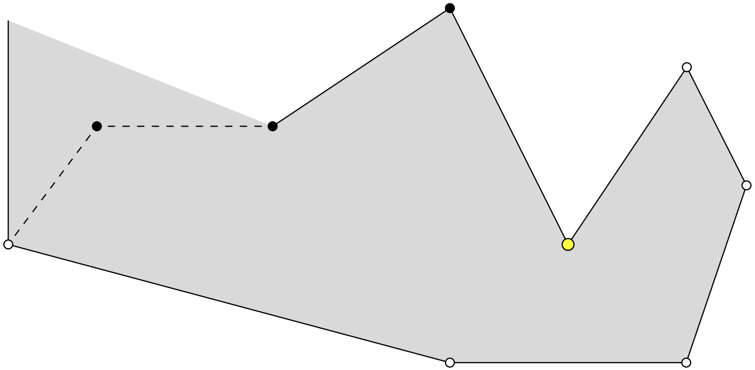
So it keeps moving toward the next unvisited vertex.

Basic Algorithm: EXPLORE Phase



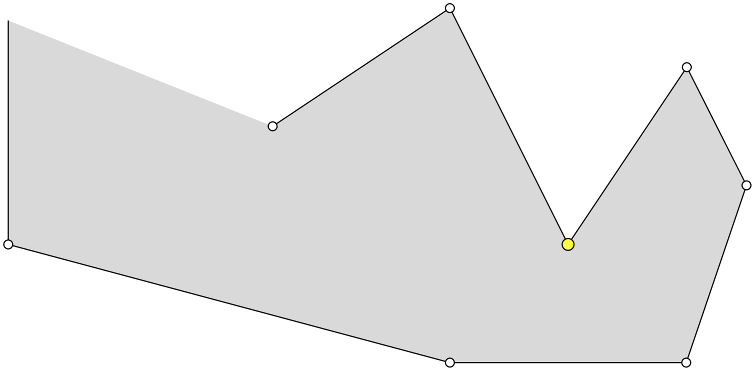
However, the initial contents of its memory are arbitrary!

Basic Algorithm: EXPLORE Phase



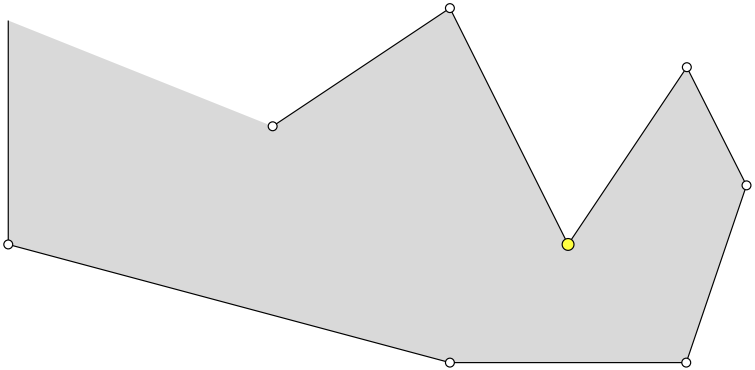
In particular, a searcher may have a false map of the polygon.

Basic Algorithm: EXPLORE Phase



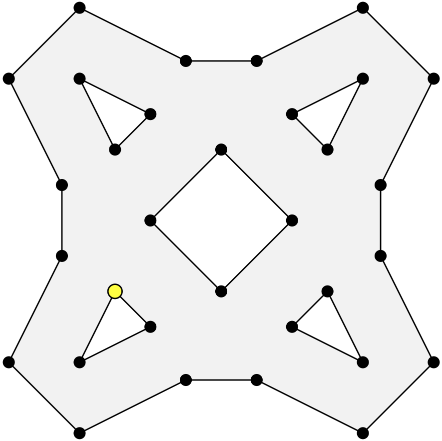
Hence, when it notices any discrepancy, it resets its own memory

Basic Algorithm: EXPLORE Phase



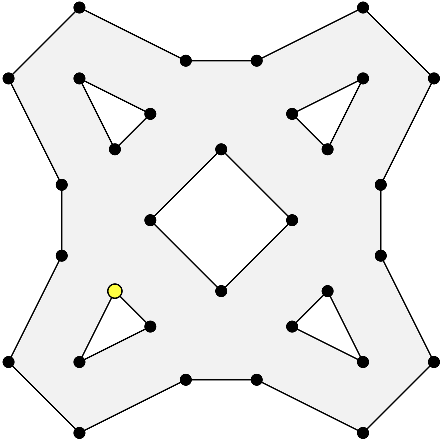
and starts rebuilding a new map from scratch.

Basic Algorithm: EXPLORE Phase



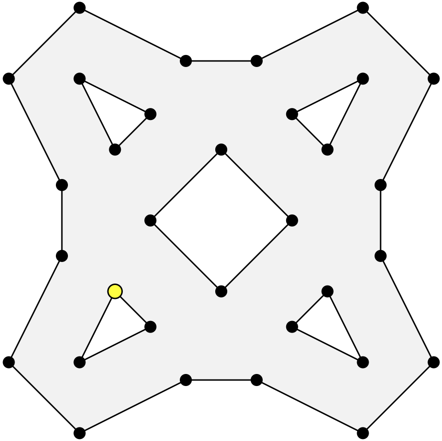
Eventually, the list of unvisited vertices becomes empty.

Basic Algorithm: EXPLORE Phase



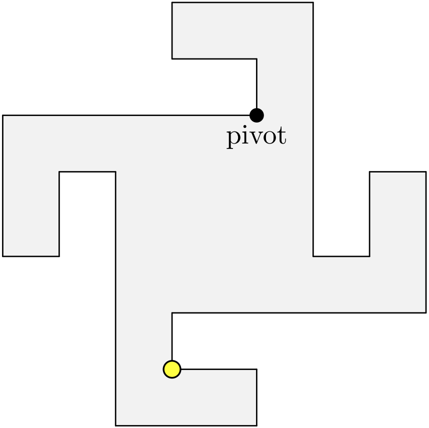
At this point, the searcher's map may or may not be correct.

Basic Algorithm: EXPLORE Phase



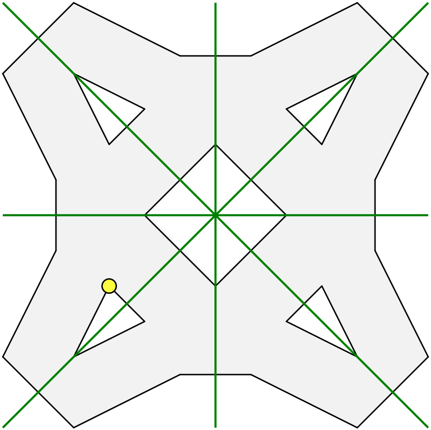
However, it assumes it is, and moves on to the next phase.

Basic Algorithm: PATROL Phase



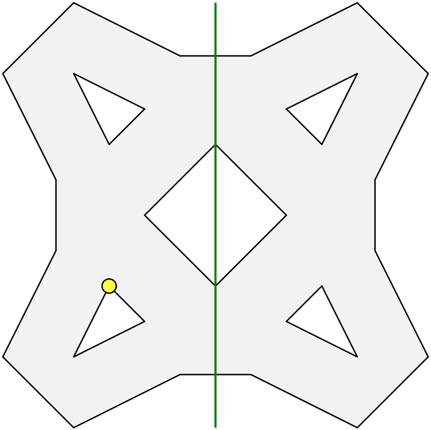
The searcher selects a *pivot point* in a similarity-invariant way.

Basic Algorithm: PATROL Phase



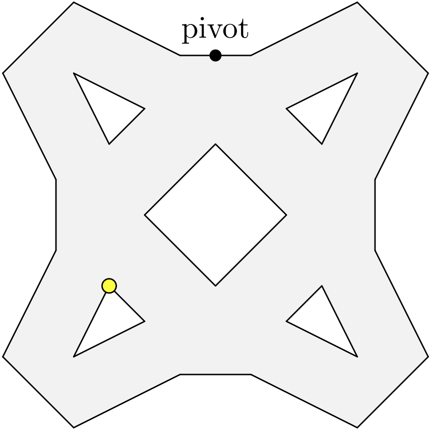
Unless the polygon has some axes of symmetry.

Basic Algorithm: PATROL Phase



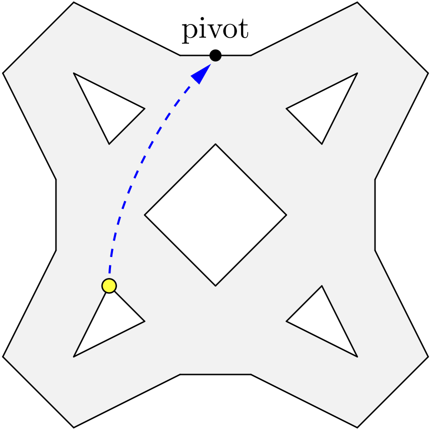
In this case, the searcher picks one axis of symmetry

Basic Algorithm: PATROL Phase



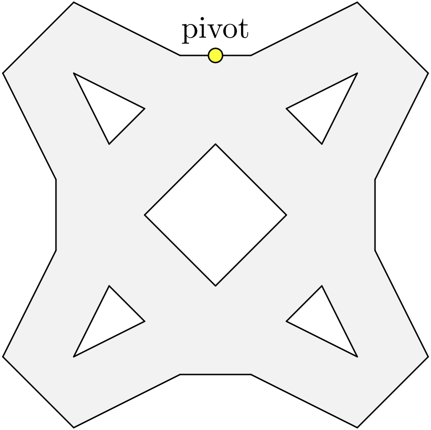
and selects a point on it in a similarity-invariant way.

Basic Algorithm: PATROL Phase



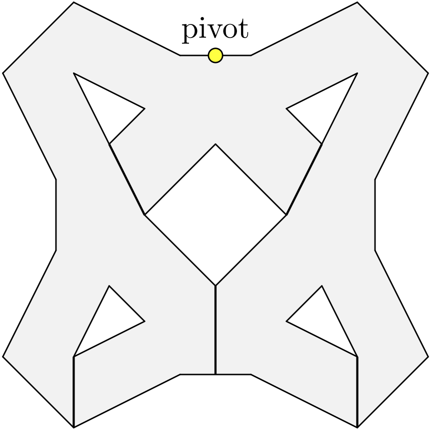
First, the searcher goes to the pivot point.

Basic Algorithm: PATROL Phase



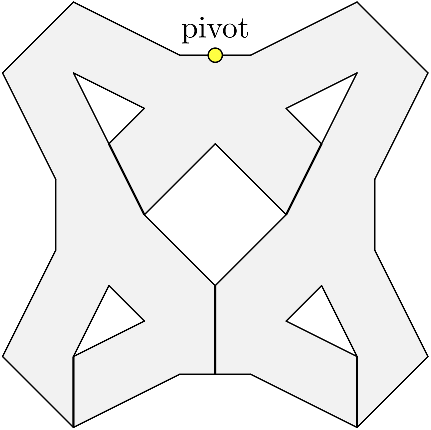
First, the searcher goes to the pivot point.

Basic Algorithm: PATROL Phase



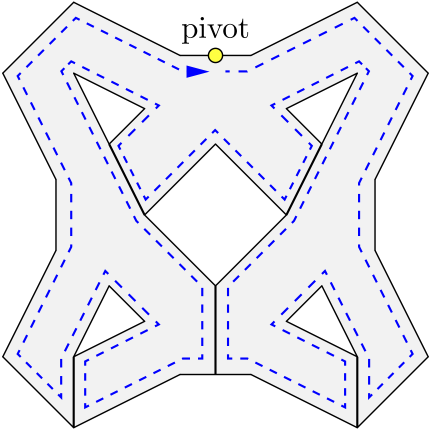
Then it *augments* the polygon in a similarity-invariant way

Basic Algorithm: PATROL Phase



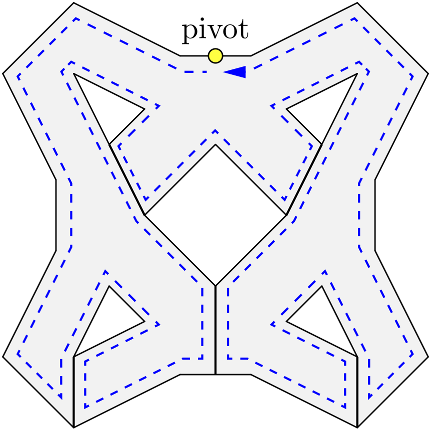
so to make its boundary connected, i.e., eliminate all holes.

Basic Algorithm: PATROL Phase



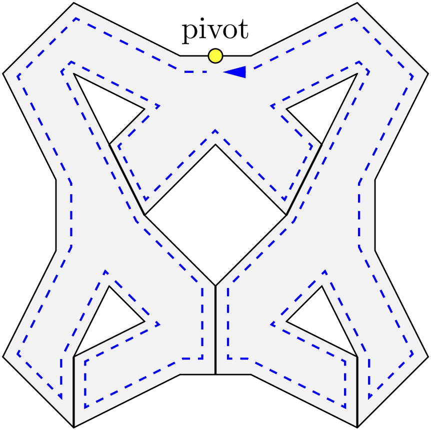
Then it keeps patrolling the augmented boundary.

Basic Algorithm: PATROL Phase



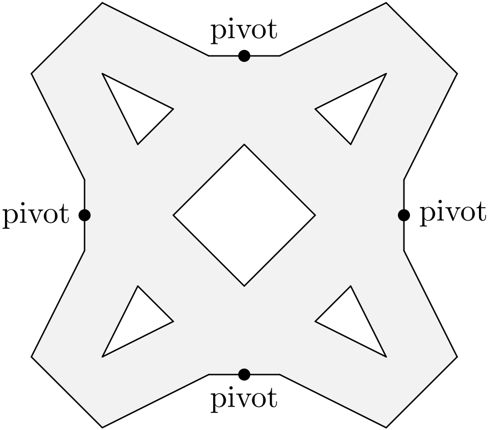
Whenever it reaches the pivot point again, it inverts direction.

Basic Algorithm: PATROL Phase



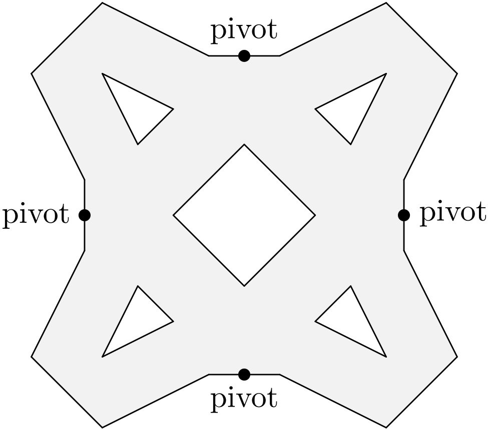
If at any time it realizes its map is wrong, it resets its memory.

Basic Algorithm: Correctness



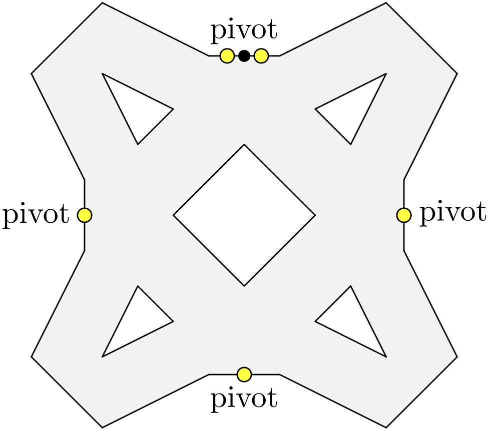
Eventually, all searchers have a correct map of the polygon.

Basic Algorithm: Correctness



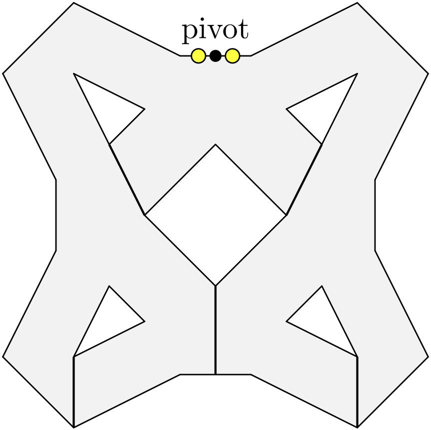
If the symmetricity is σ, there are σ possible pivot points.

Basic Algorithm: Correctness



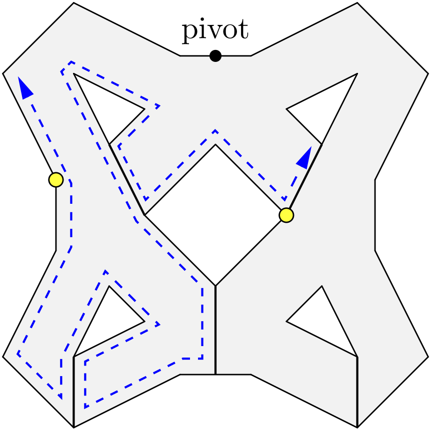
If there are $\sigma + 1$ searchers, two of them choose the same pivot

Basic Algorithm: Correctness



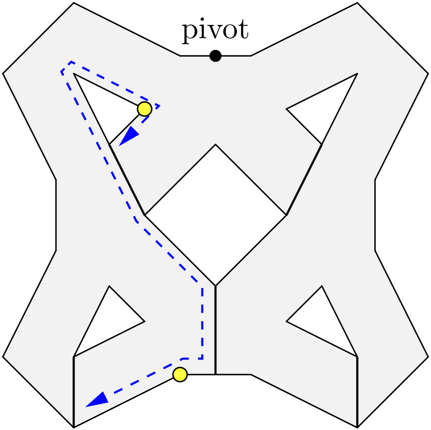
and they augment the polygon in the same way.

Basic Algorithm: Correctness



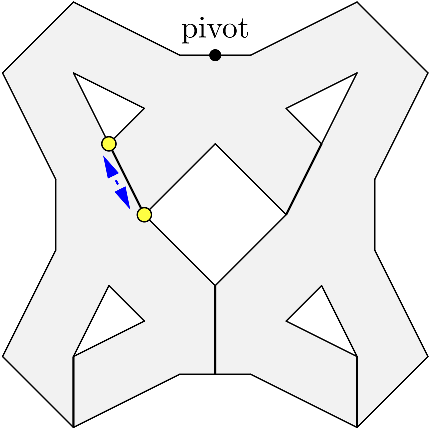
Hence they keep following the same path in both directions.

Basic Algorithm: Correctness



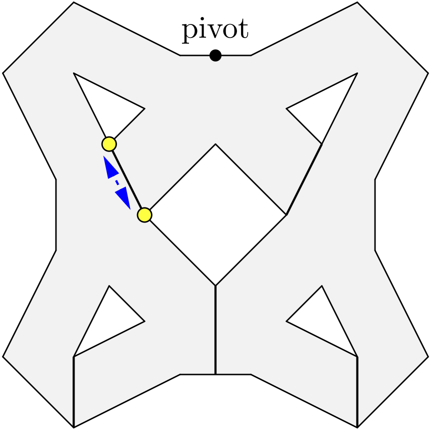
Hence they keep following the same path in both directions.

Basic Algorithm: Correctness



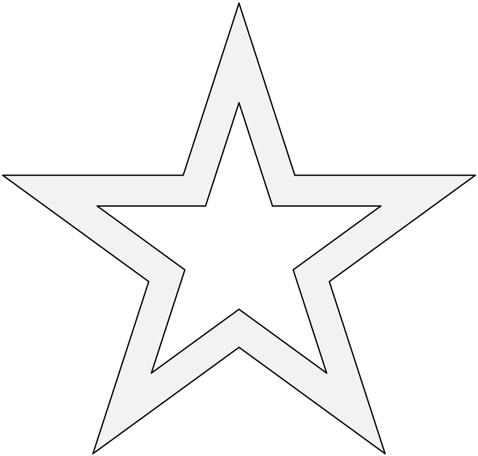
Eventually, they must meet on an edge of this path.

Basic Algorithm: Correctness



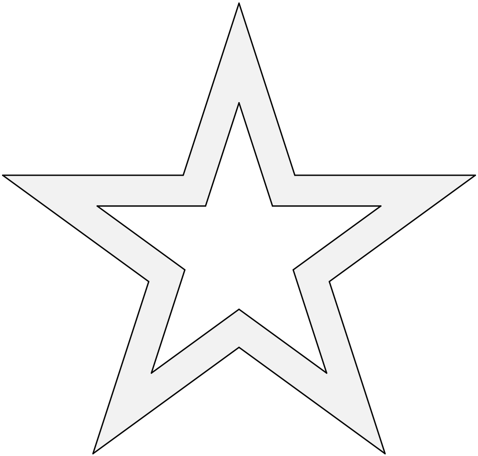
Theorem: among $\sigma + 1$ searchers, at least two will meet.

Improving the Basic Algorithm



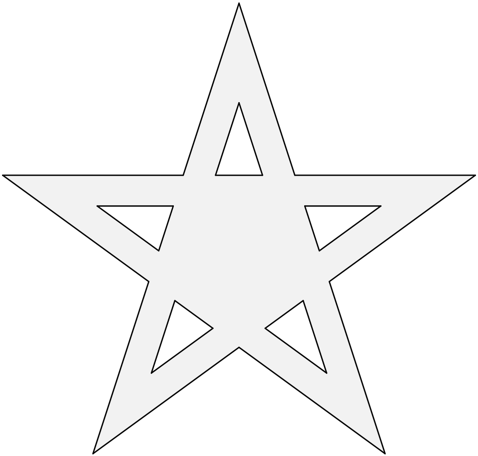
Recall that our negative examples had a hole around the center.

Improving the Basic Algorithm



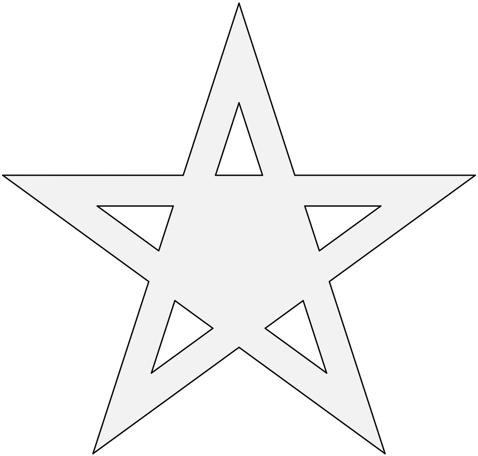
Can we do better if we exclude these polygons?

Improving the Basic Algorithm



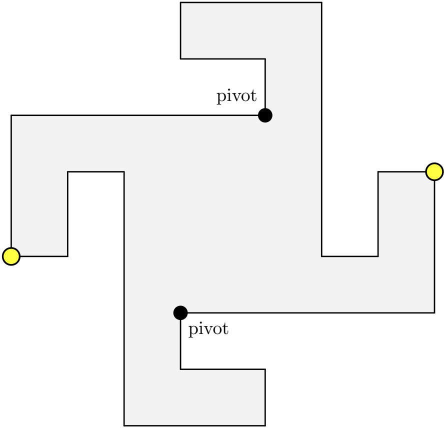
Suppose the center of the polygon is not in a hole.

Improving the Basic Algorithm



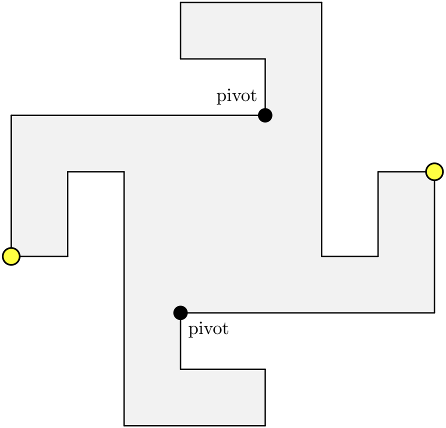
Claim: in this case 2 searchers are sufficient.

Improving the Basic Algorithm



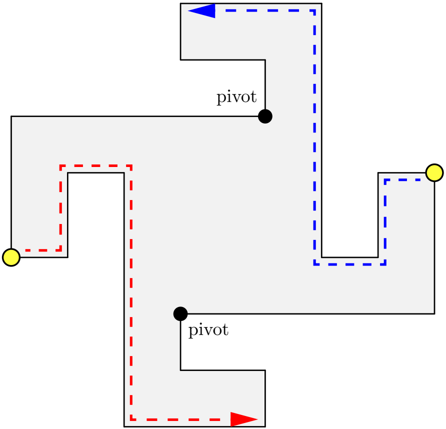
The basic algorithm may not work in this case!

Improving the Basic Algorithm



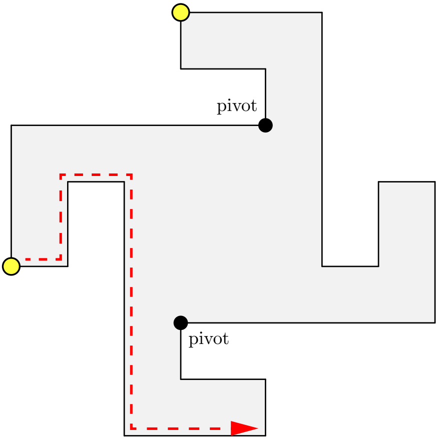
Let the two searchers choose opposite pivot points.

Improving the Basic Algorithm



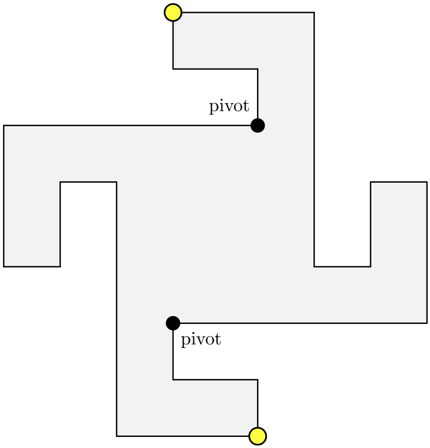
We can schedule their movements so that the never meet.

Improving the Basic Algorithm



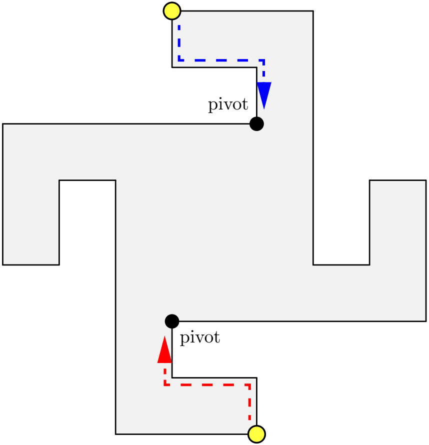
We can schedule their movements so that the never meet.

Improving the Basic Algorithm



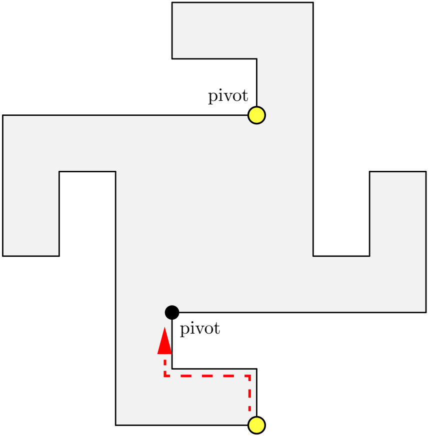
by keeping one hidden while the other visits the central area.

Improving the Basic Algorithm



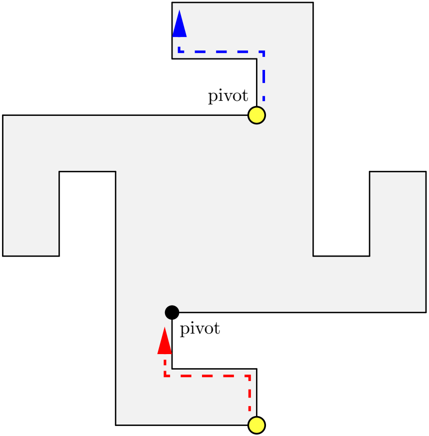
by keeping one hidden while the other visits the central area.

Improving the Basic Algorithm



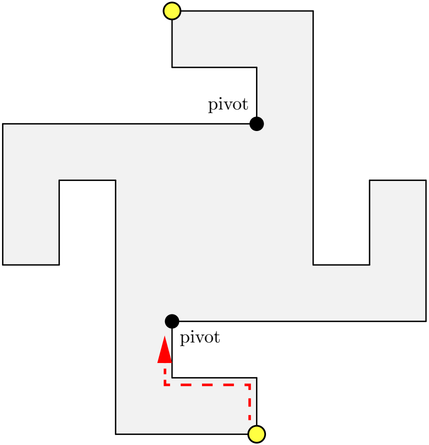
by keeping one hidden while the other visits the central area.

Improving the Basic Algorithm



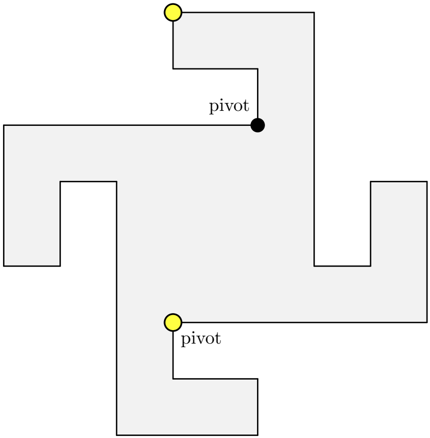
by keeping one hidden while the other visits the central area.

Improving the Basic Algorithm



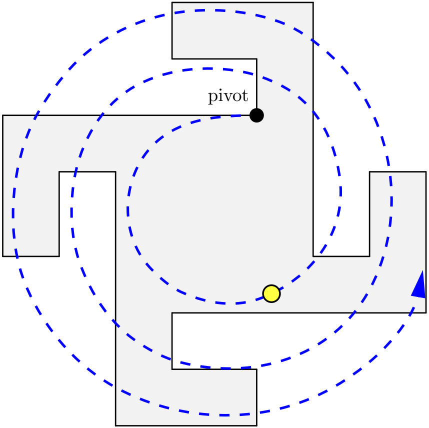
by keeping one hidden while the other visits the central area.

Improving the Basic Algorithm



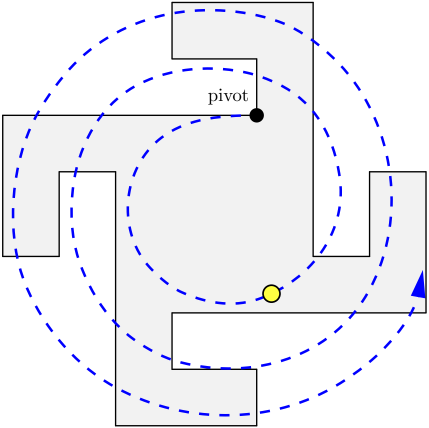
by keeping one hidden while the other visits the central area.

Improving the Basic Algorithm



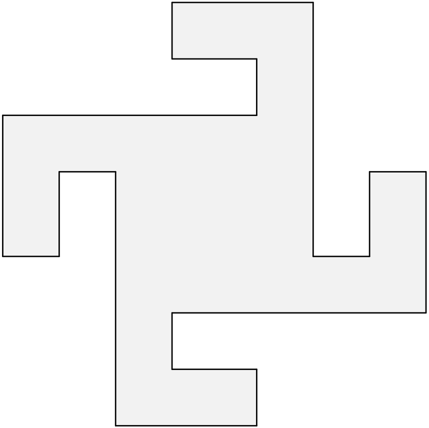
Observation: searchers should “spiral” around the center,

Improving the Basic Algorithm



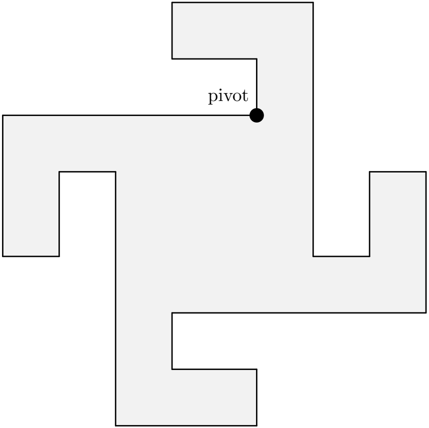
modifying their distance gradually.

Improved Algorithm: PATROL Phase



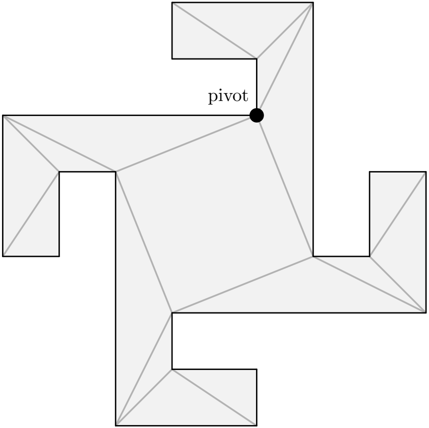
The EXPLORE phase is the same as in the basic algorithm.

Improved Algorithm: PATROL Phase



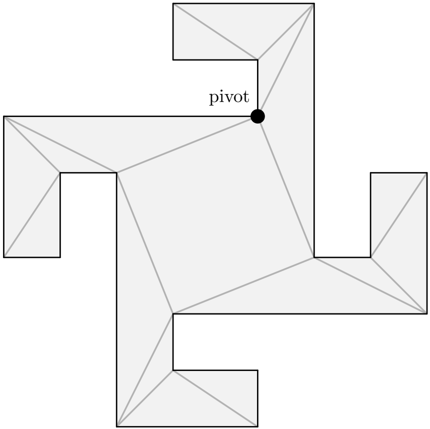
At the end, a pivot point is chosen as before.

Improved Algorithm: PATROL Phase



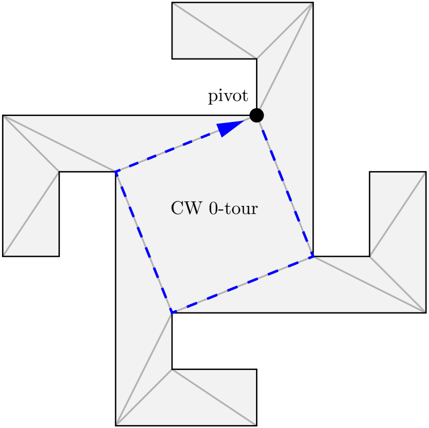
Then the non-central areas are triangulated.

Improved Algorithm: PATROL Phase



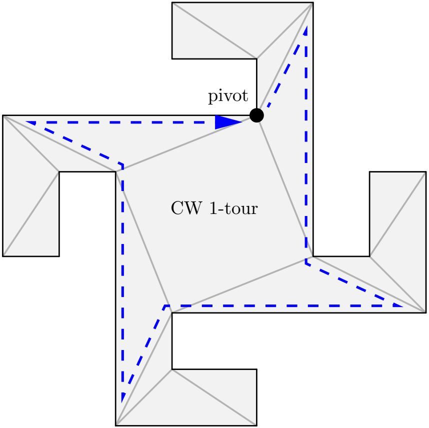
Symmetric branches are triangulated in a symmetric way.

Improved Algorithm: PATROL Phase



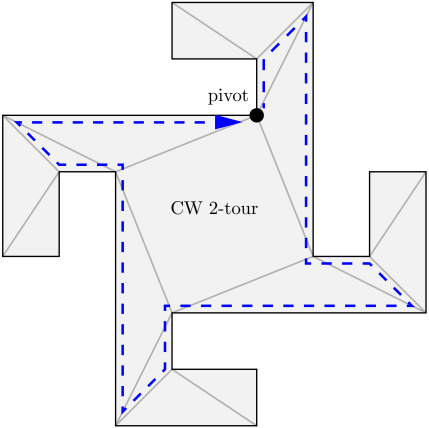
The patrol starts with a *clockwise* tour of the central area.

Improved Algorithm: PATROL Phase



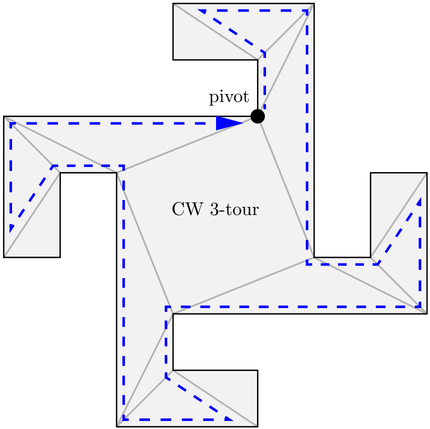
Followed by a *clockwise* tour of the triangles at depth 1.

Improved Algorithm: PATROL Phase



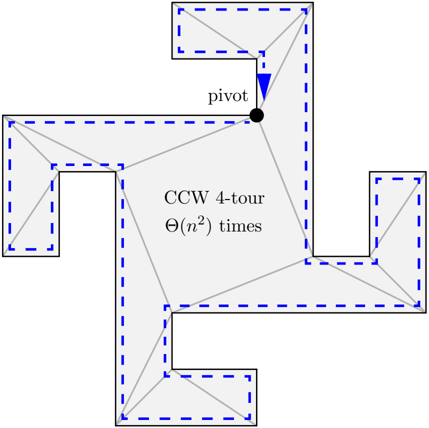
Then a *clockwise* tour of the triangles at depth at most 2.

Improved Algorithm: PATROL Phase



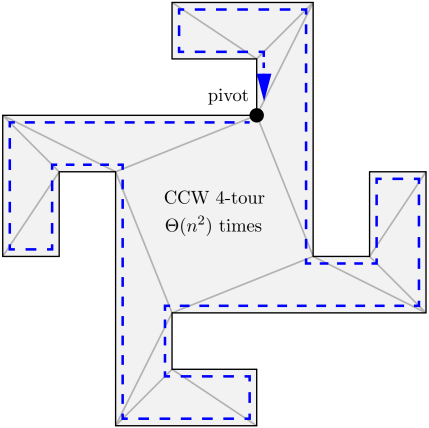
Then a *clockwise* tour of the triangles at depth at most 3, etc.

Improved Algorithm: PATROL Phase



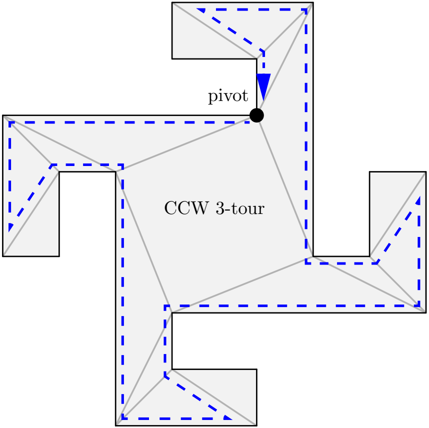
Then several *counterclockwise* tours of the perimeter.

Improved Algorithm: PATROL Phase



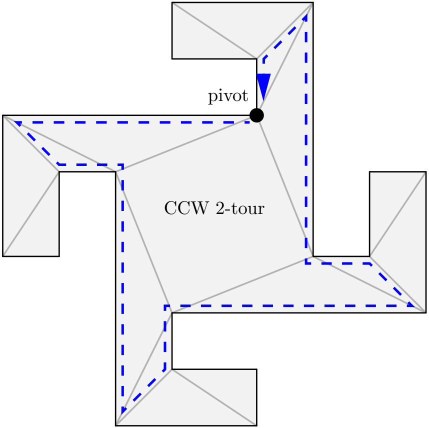
(A quadratic number of tours suffices.)

Improved Algorithm: PATROL Phase



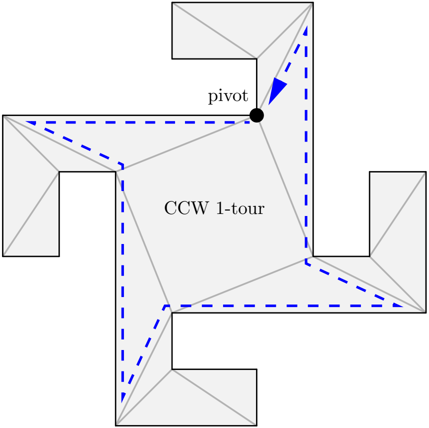
Then the smaller tours are repeated the reverse order,

Improved Algorithm: PATROL Phase



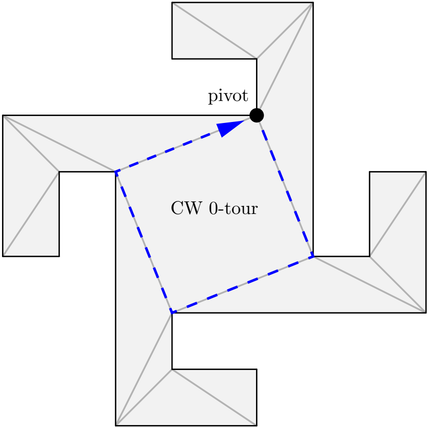
this time *counterclockwise*.

Improved Algorithm: PATROL Phase



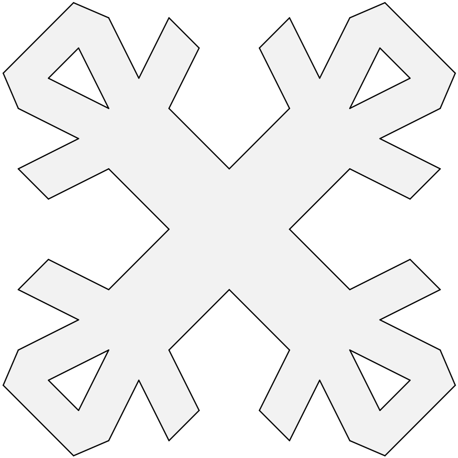
this time *counterclockwise*.

Improved Algorithm: PATROL Phase



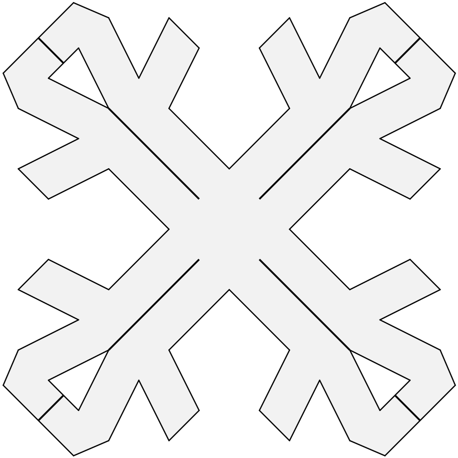
The patrol restarts with a *clockwise* tour of the central area, etc.

Improved Algorithm: PATROL Phase



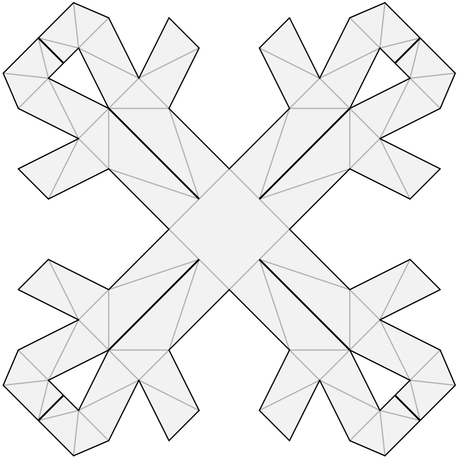
If the polygon has axes of symmetry, it is augmented first.

Improved Algorithm: PATROL Phase



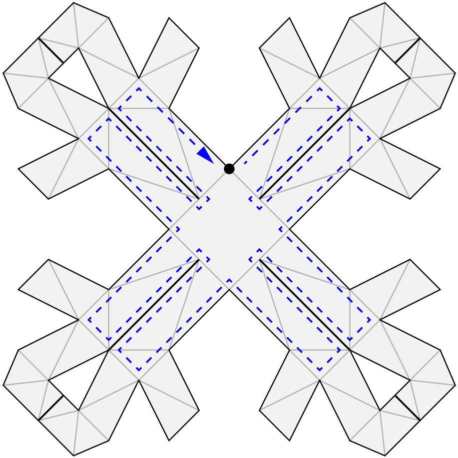
If the polygon has axes of symmetry, it is augmented first.

Improved Algorithm: PATROL Phase



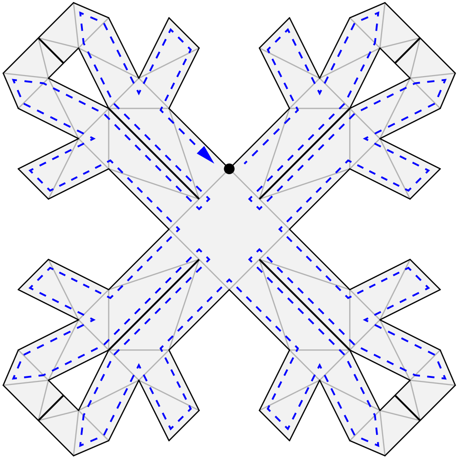
So that its branches can be triangulated in a symmetric way.

Improved Algorithm: PATROL Phase



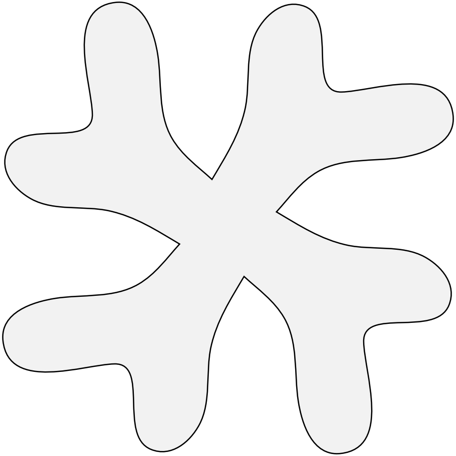
Hence the searchers implicitly agree on the same triangulation,

Improved Algorithm: PATROL Phase



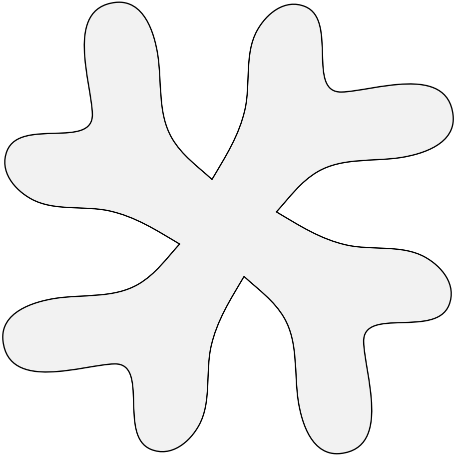
even if their coordinate systems are oriented specularly.

Improved Algorithm: Correctness



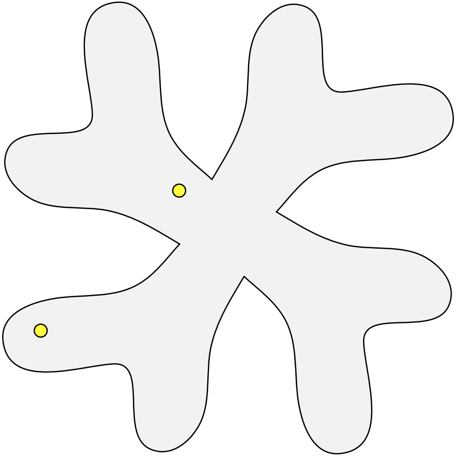
If $\sigma = 1$, the basic algorithm already works for 2 searchers.

Improved Algorithm: Correctness



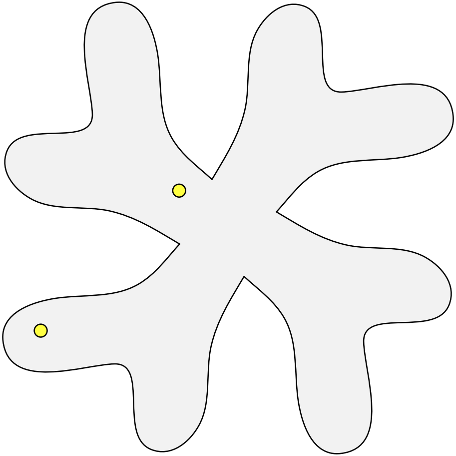
Let $\sigma > 1$ and let two searchers execute the improved algorithm.

Improved Algorithm: Correctness



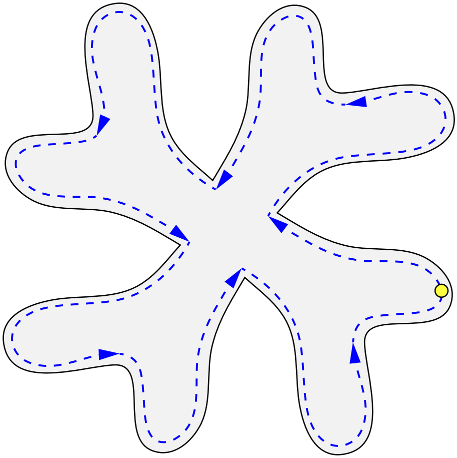
Eventually, both searchers have a correct map of the polygon,

Improved Algorithm: Correctness



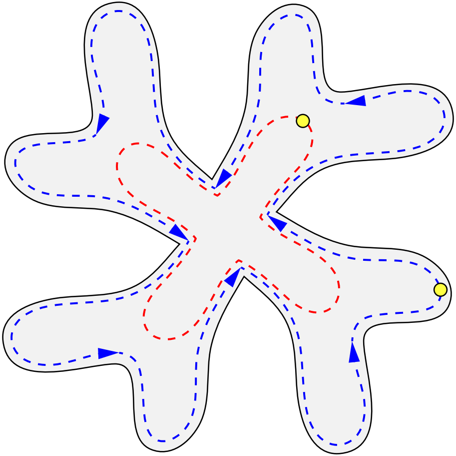
and execute the PATROL phase.

Improved Algorithm: Correctness



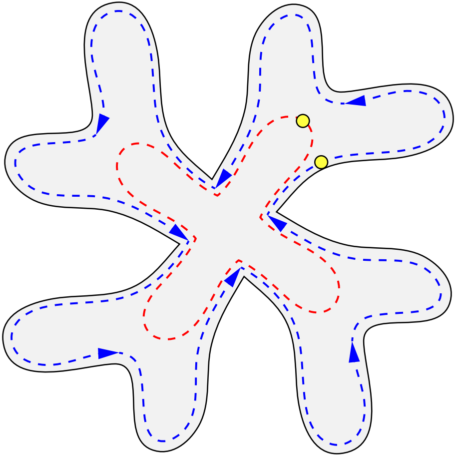
At some point, one searchers begins a series of perimeter tours.

Improved Algorithm: Correctness



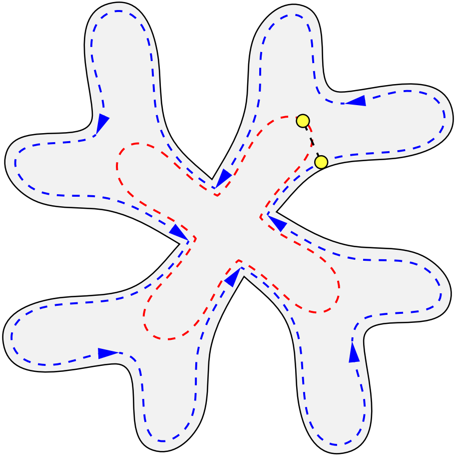
Meanwhile, the other searcher is performing one of its own tours.

Improved Algorithm: Correctness



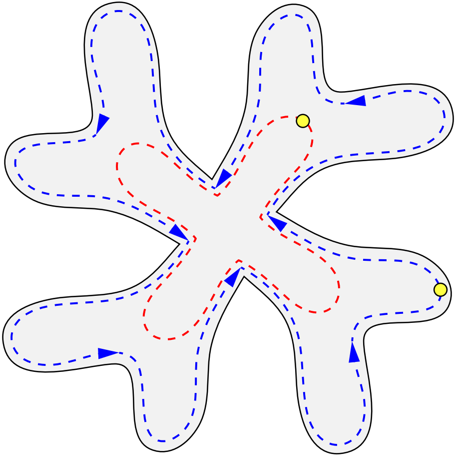
If the second searcher does not move, the first searcher sees it

Improved Algorithm: Correctness



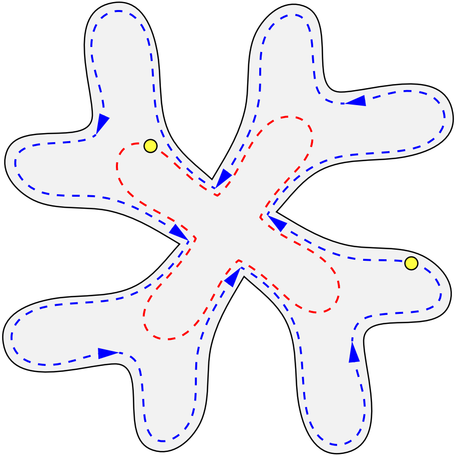
by the time it has completed a perimeter tour.

Improved Algorithm: Correctness



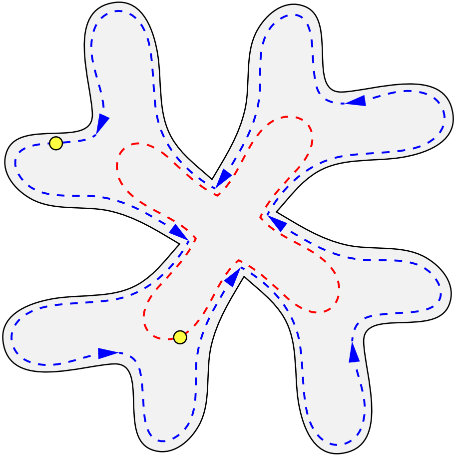
Hence, every time the first searcher performs one perimeter tour,

Improved Algorithm: Correctness



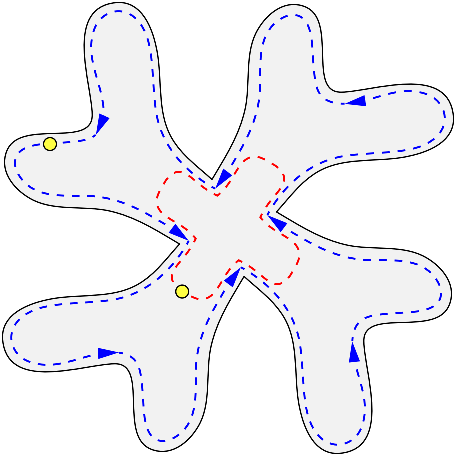
the second searcher must make some "progress" on its own tour:

Improved Algorithm: Correctness



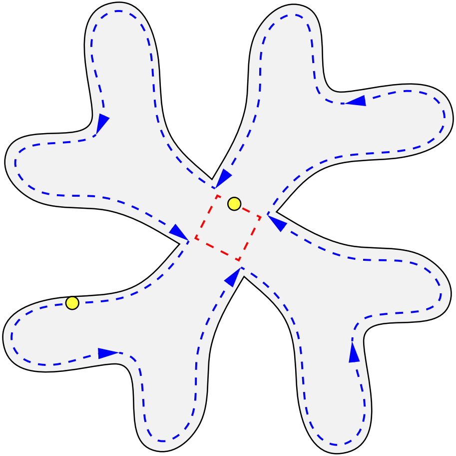
it should at least move to another triangle of the triangulation.

Improved Algorithm: Correctness



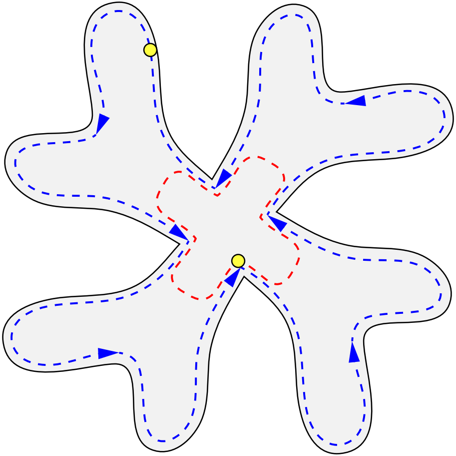
Since the first searcher performs $\Theta(n^2)$ perimeter tours,

Improved Algorithm: Correctness



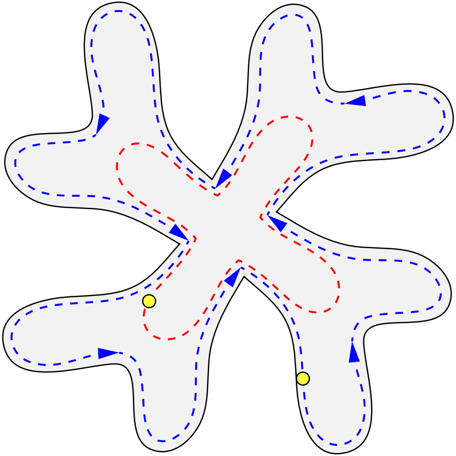
and the other performs $O(n)$ tours which cover $O(n)$ triangles,

Improved Algorithm: Correctness



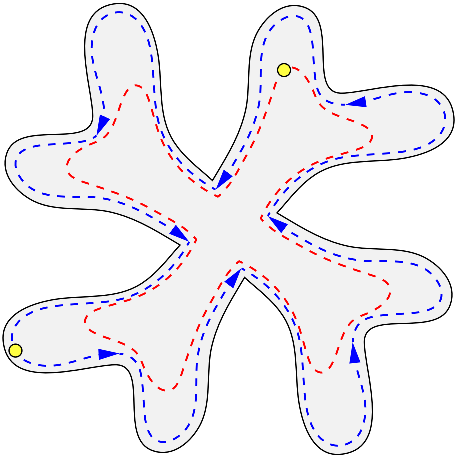
eventually both searchers will be performing a perimeter tour.

Improved Algorithm: Correctness



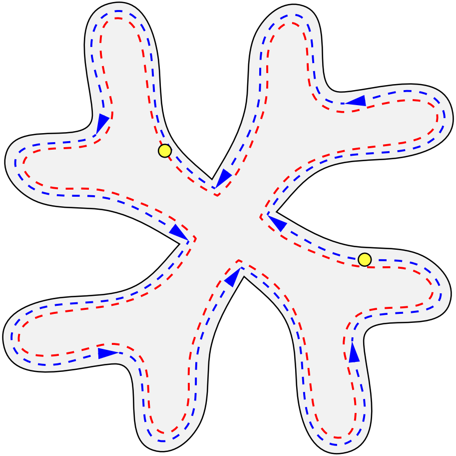
eventually both searchers will be performing a perimeter tour.

Improved Algorithm: Correctness



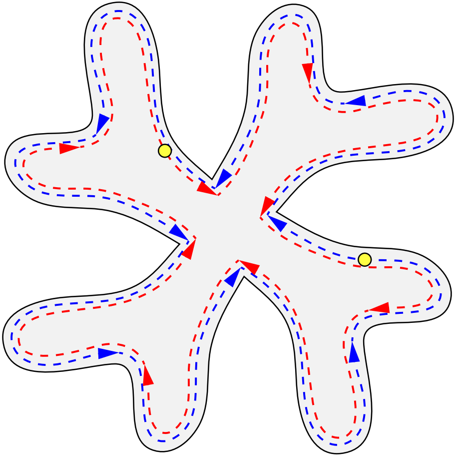
eventually both searchers will be performing a perimeter tour.

Improved Algorithm: Correctness



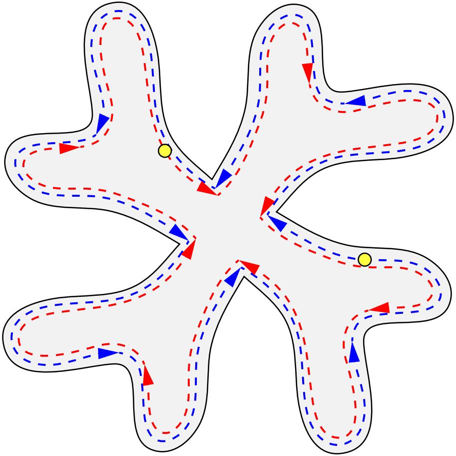
eventually both searchers will be performing a perimeter tour.

Improved Algorithm: Correctness



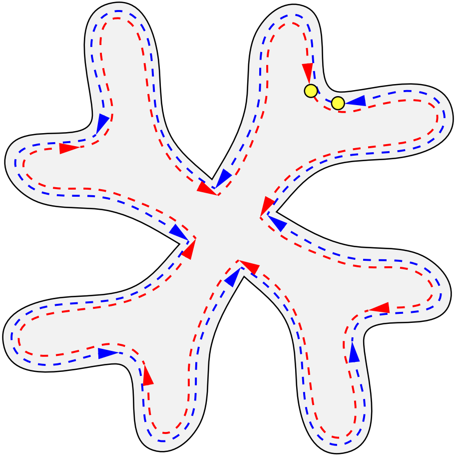
If the searchers disagree on the notion of "clockwise",

Improved Algorithm: Correctness



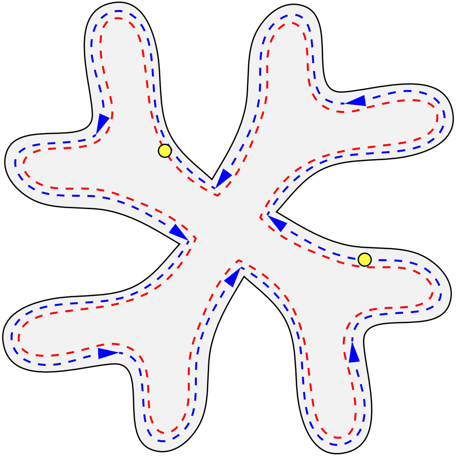
they tour the perimeter in opposite directions.

Improved Algorithm: Correctness



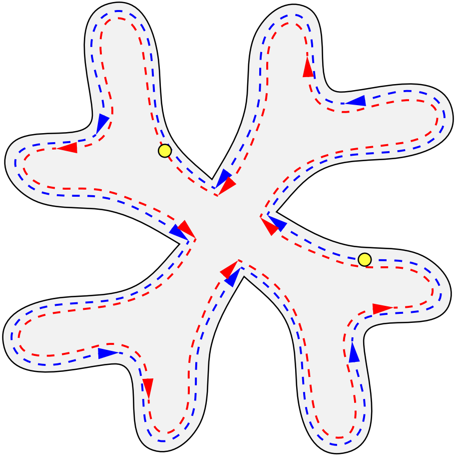
Hence they eventually meet on the perimeter.

Improved Algorithm: Correctness



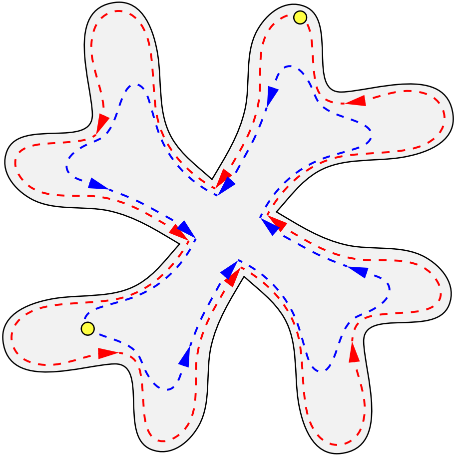
If the searchers agree on the notion of “clockwise”,

Improved Algorithm: Correctness



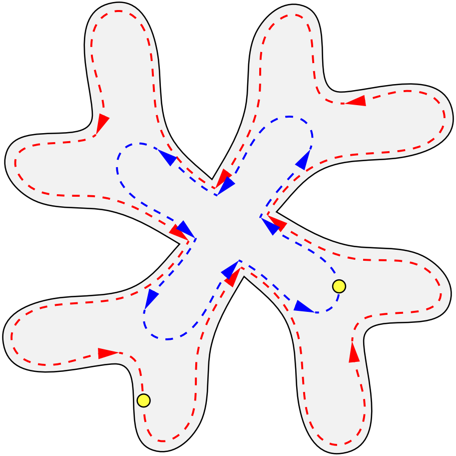
they tour the perimeter in the same direction.

Improved Algorithm: Correctness



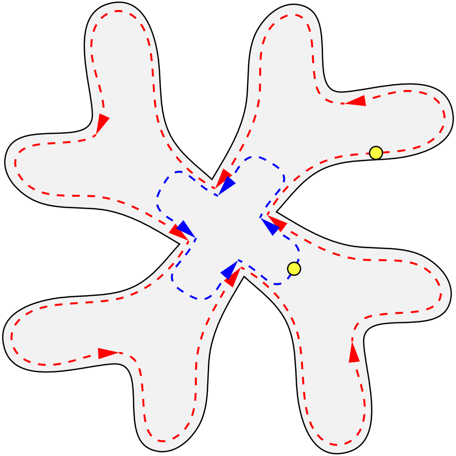
Hence they may not meet on a perimeter tour,

Improved Algorithm: Correctness



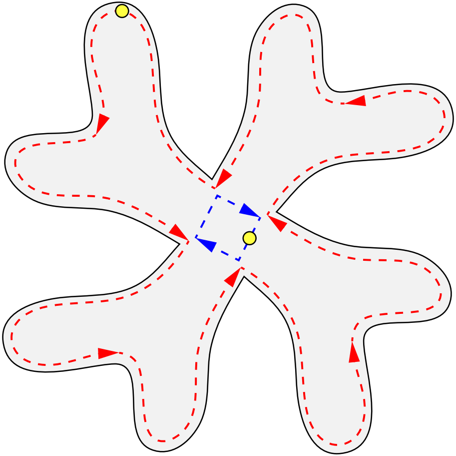
and one searcher may start spiraling toward the central area.

Improved Algorithm: Correctness



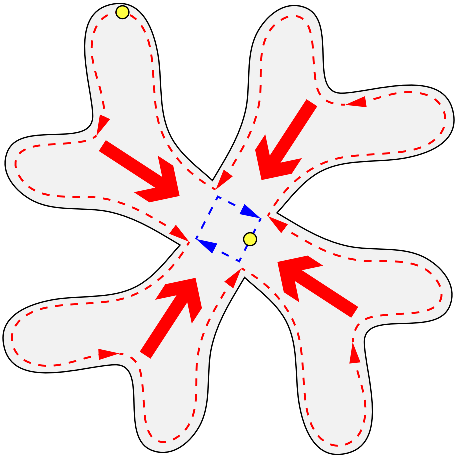
and one searcher may start spiraling toward the central area.

Improved Algorithm: Correctness



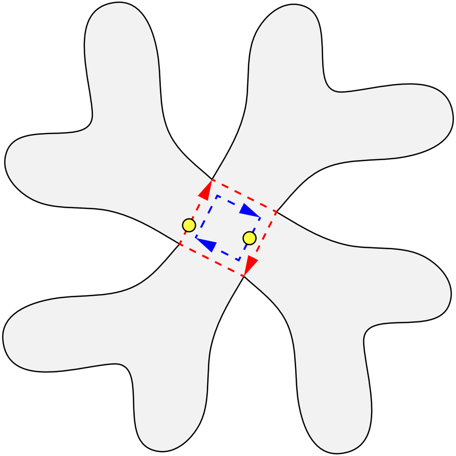
and one searcher may start spiraling toward the central area.

Improved Algorithm: Correctness



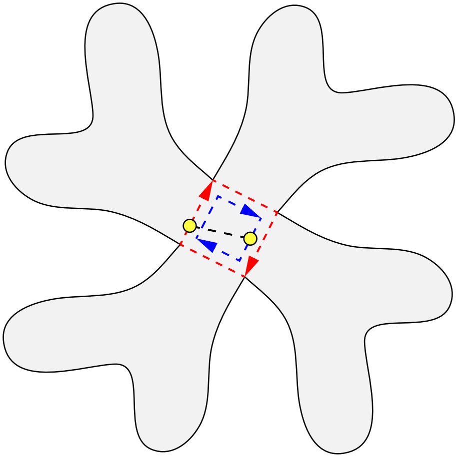
If the other searcher spirals toward the central area too,

Improved Algorithm: Correctness



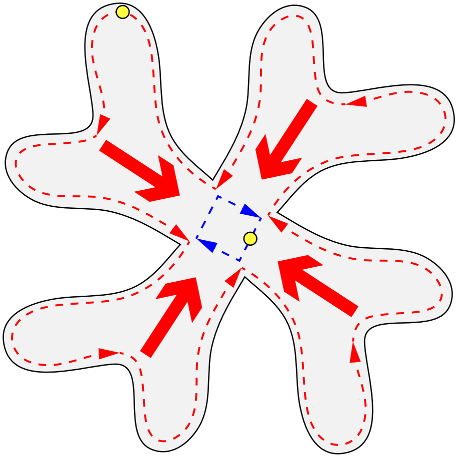
and they are both on a tour of the central area at the same time,

Improved Algorithm: Correctness



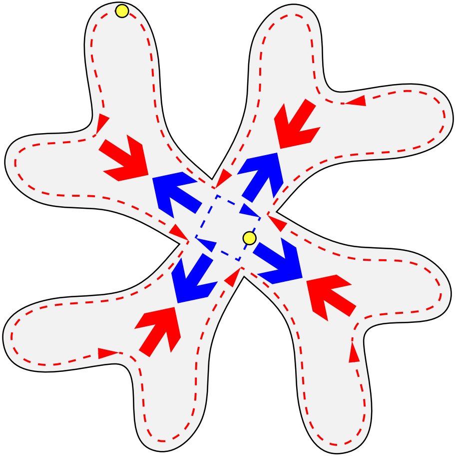
they see each other, because there is no hole around the center.

Improved Algorithm: Correctness



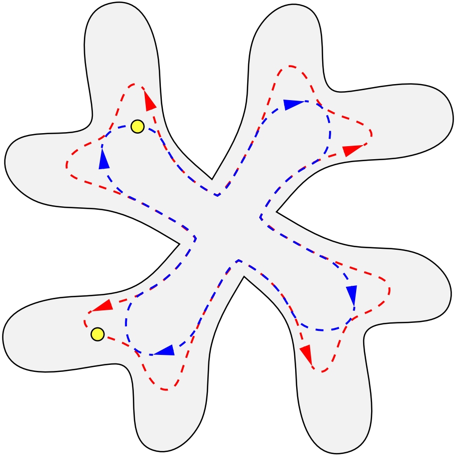
Otherwise, they start spiraling in opposite directions.

Improved Algorithm: Correctness



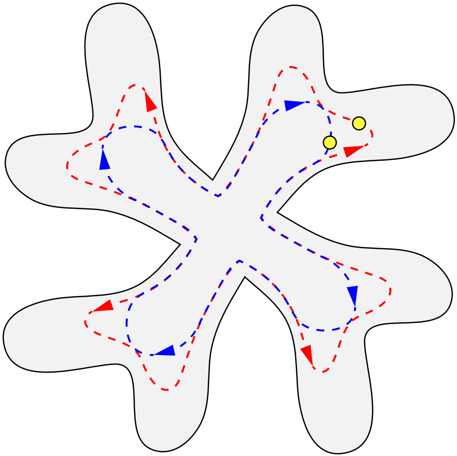
Otherwise, they start spiraling in opposite directions.

Improved Algorithm: Correctness



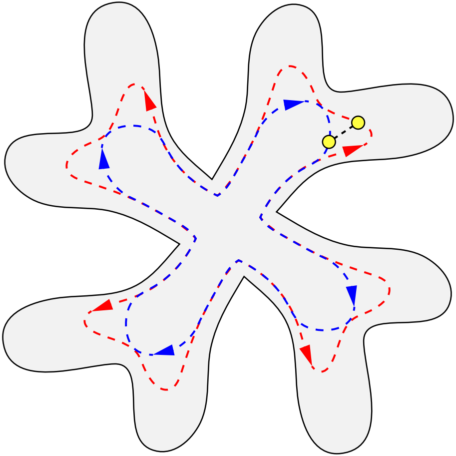
Eventually, they perform the same tour or two “adjacent” tours.

Improved Algorithm: Correctness



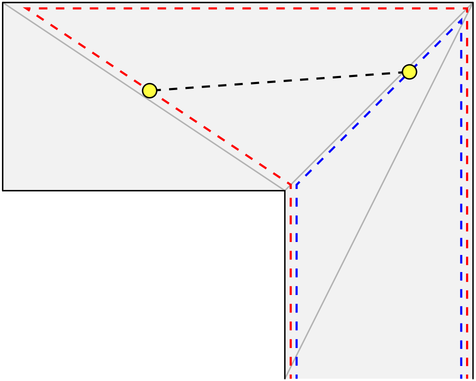
Since they go in opposite directions, they must meet.

Improved Algorithm: Correctness



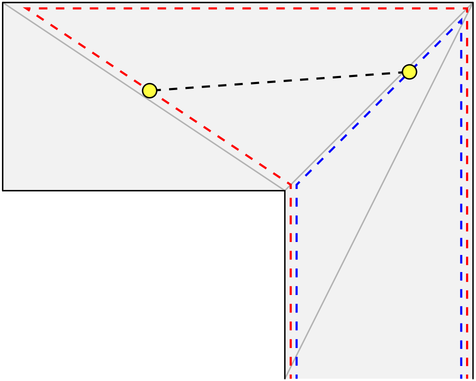
Since they go in opposite directions, they must meet.

Improved Algorithm: Correctness



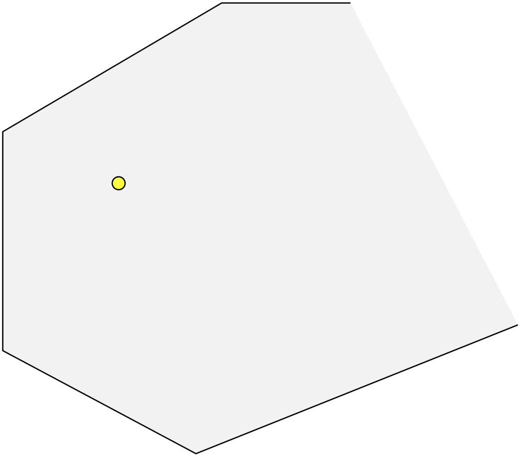
The meeting occurs whenever they reach the same triangle.

Improved Algorithm: Correctness



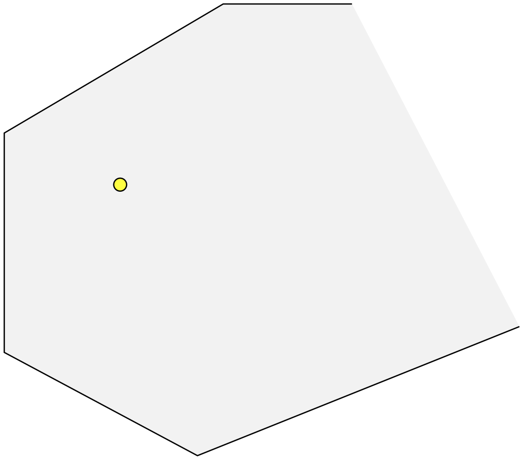
Theorem: if there is no central hole, 2 searchers can meet.

Meeting with no Memory



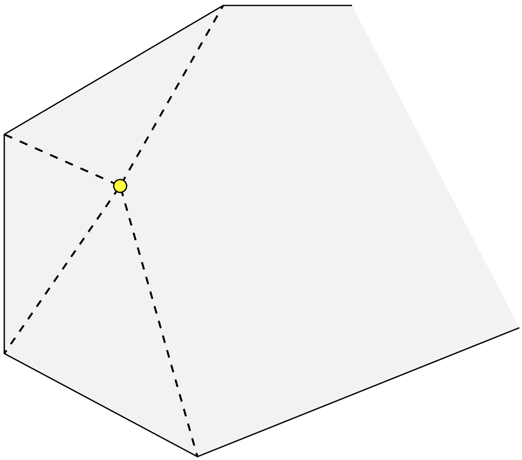
Suppose that searchers are *memoryless*.

Meeting with no Memory



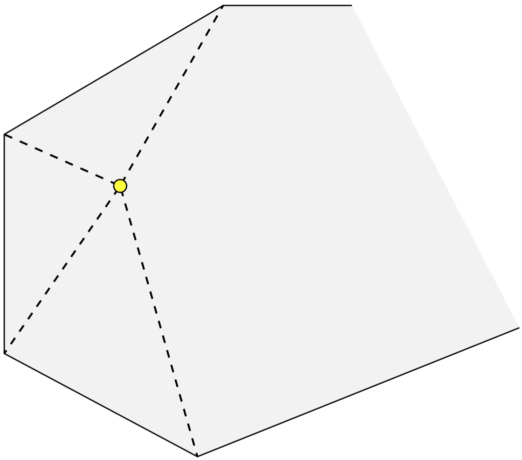
They must decide where to go based solely on their current view.

Meeting with no Memory



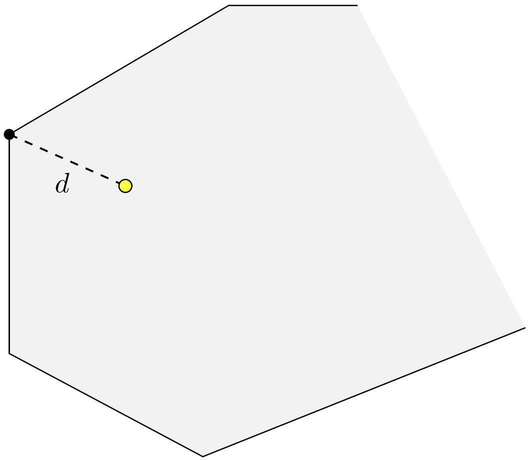
But they can simulate memory by moving to certain points,

Meeting with no Memory



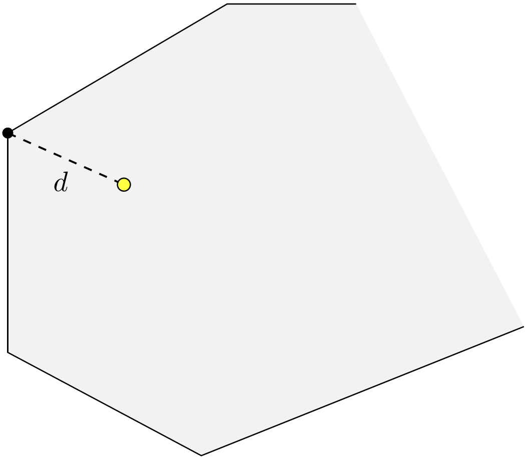
such as points at specific distances from some vertices.

Meeting with no Memory



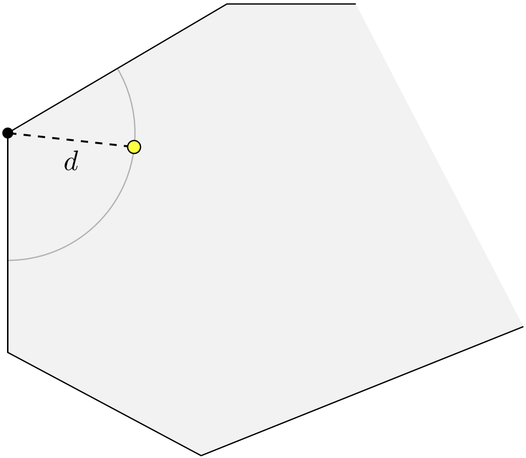
The visible vertex closest to a searcher is its *virtual vertex*.

Meeting with no Memory



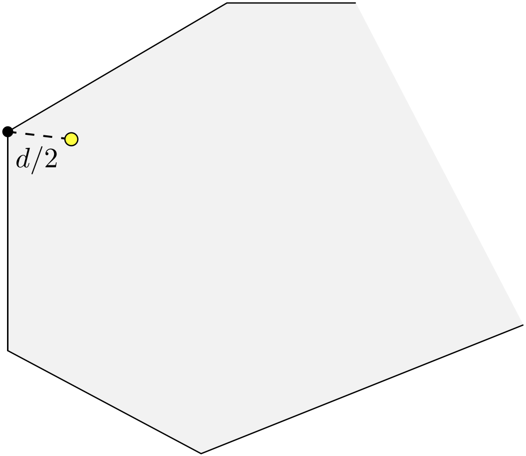
Its distance d from its virtual vertex represents its memory.

Meeting with no Memory



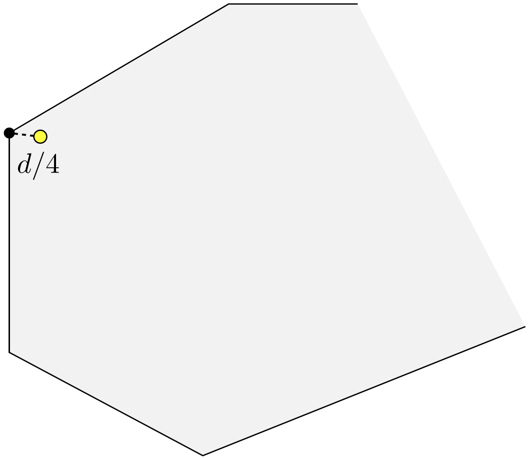
As long as this distance is d, the data represented is the same.

Meeting with no Memory



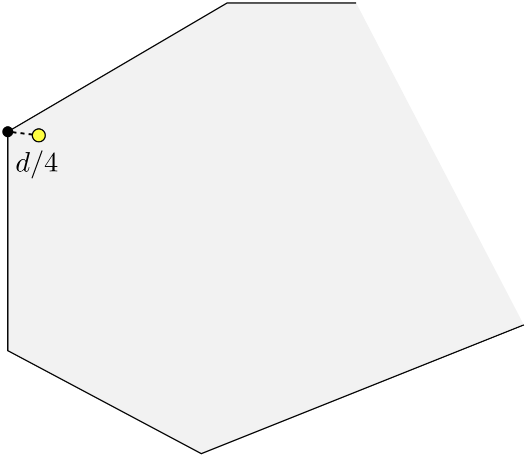
Also, d and $d/2$ represent the same data.

Meeting with no Memory



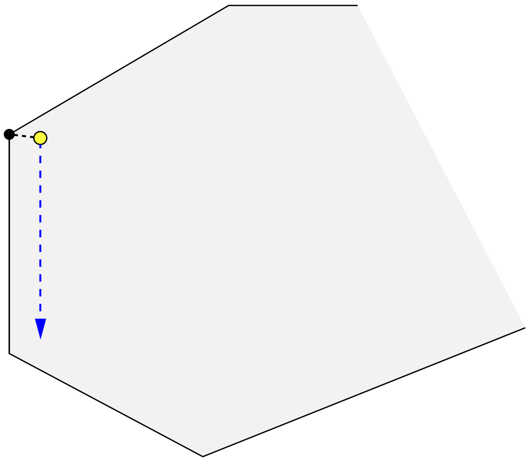
So the searcher can get arbitrarily close to its virtual vertex

Meeting with no Memory



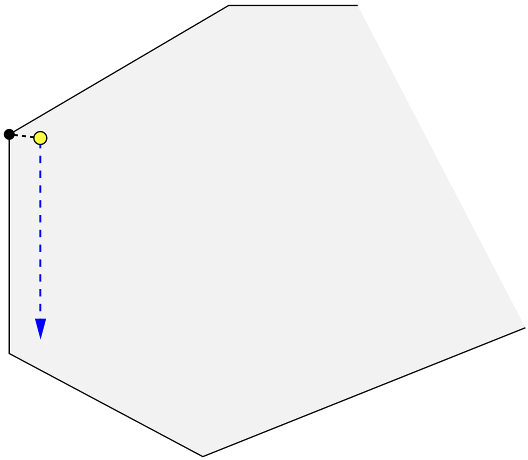
without "forgetting" anything.

Meeting with no Memory



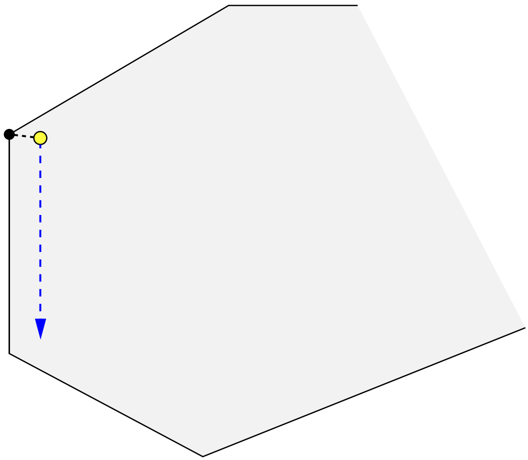
Searchers execute the previous Meeting algorithms

Meeting with no Memory



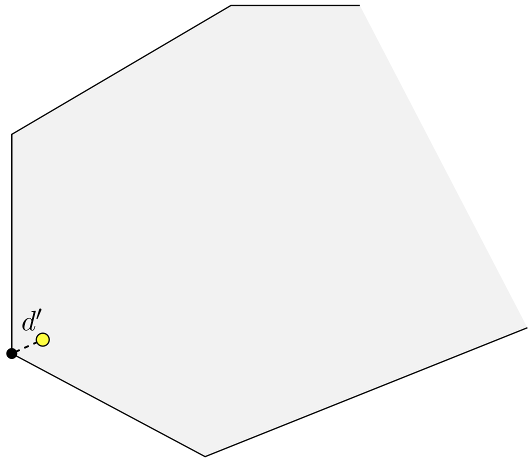
pretending to be exactly on their virtual vertices

Meeting with no Memory



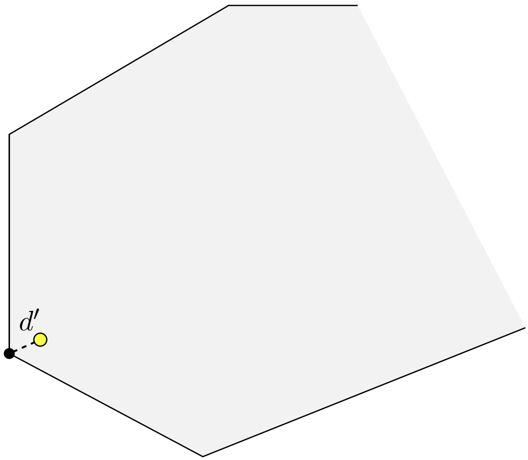
and decoding d to retrieve their memory.

Meeting with no Memory



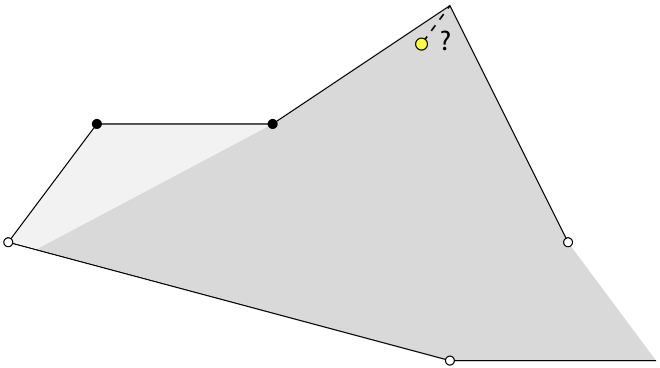
When they move to another vertex, they choose a distance d'

Meeting with no Memory



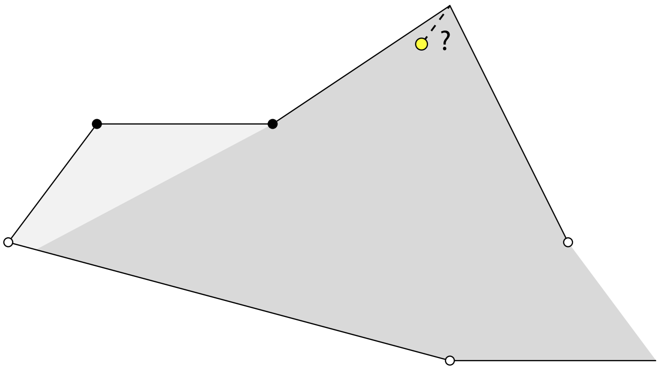
representing the updated memory contents.

Meeting with no Memory



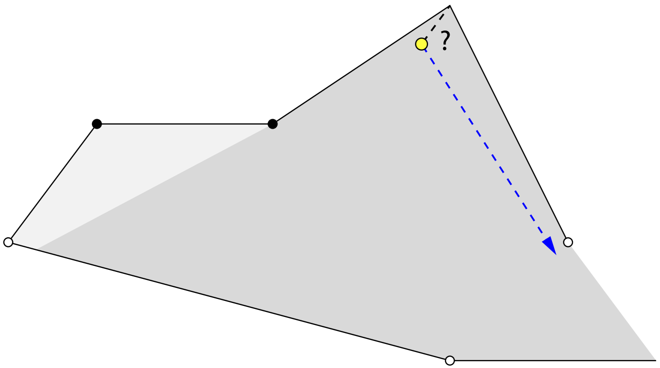
Initially, searchers are located in arbitrary positions.

Meeting with no Memory



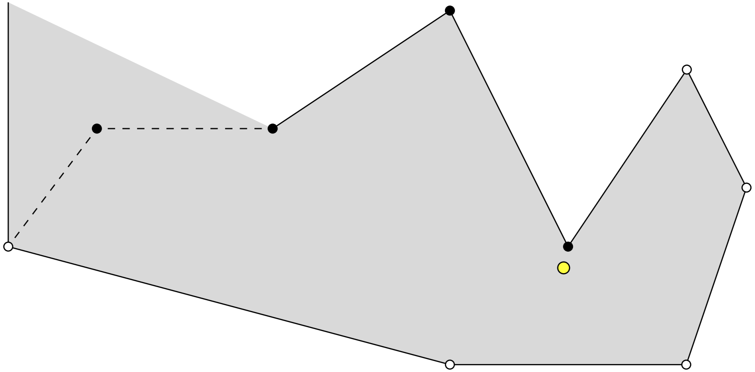
So, the data they encode is arbitrary.

Meeting with no Memory



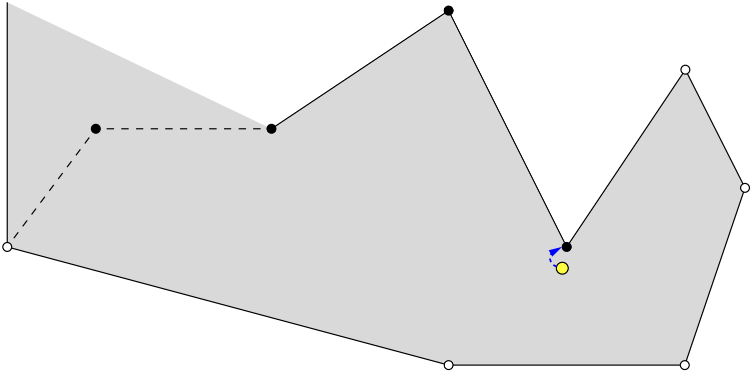
When a searcher changes virtual vertex,

Meeting with no Memory



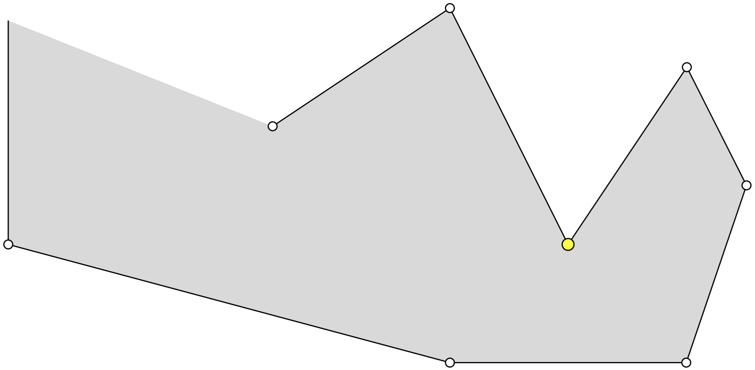
it may discover that the map it is representing is wrong.

Meeting with no Memory



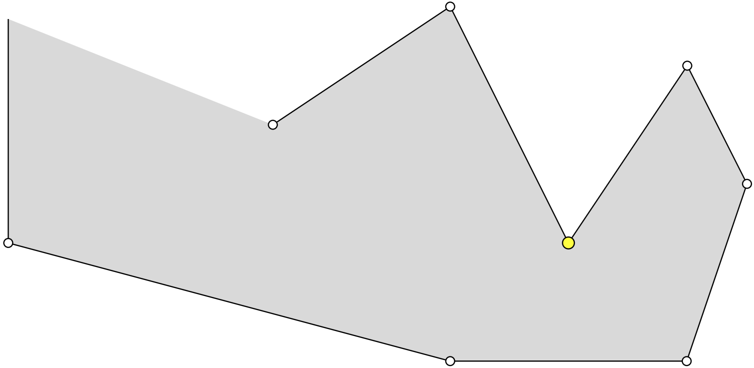
If that happens, it goes to its virtual vertex,

Meeting with no Memory



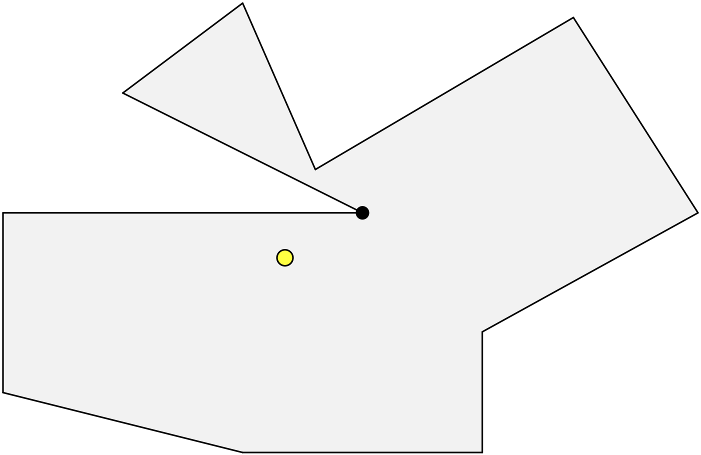
which corresponds to resetting its own memory.

Meeting with no Memory



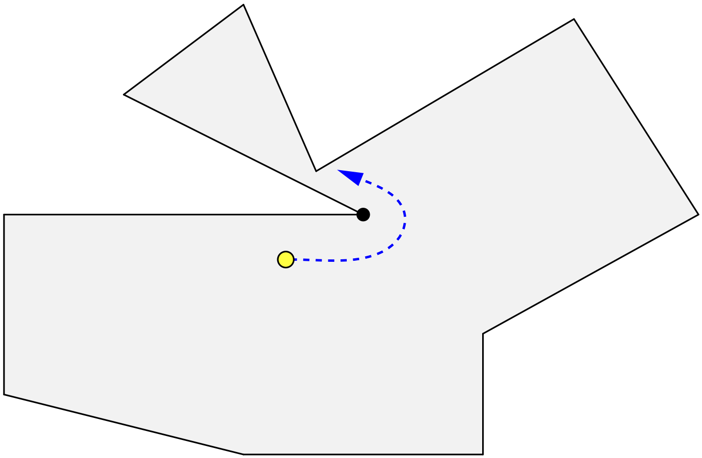
After the first reset, the (partial) map will be correct.

Traveling Around Reflex Vertices



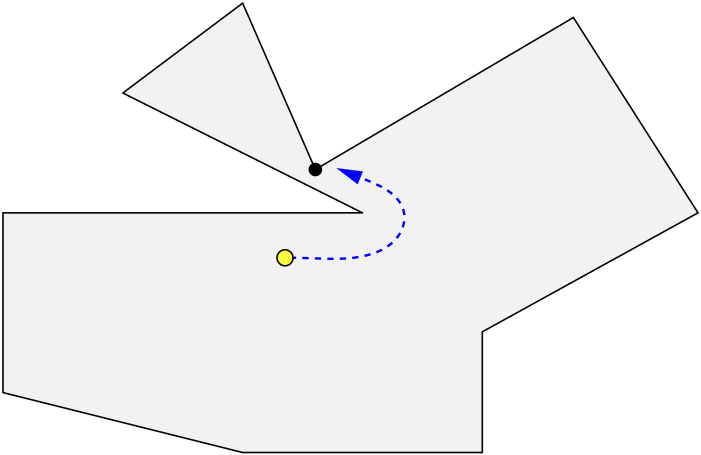
Suppose the virtual vertex is reflex

Traveling Around Reflex Vertices



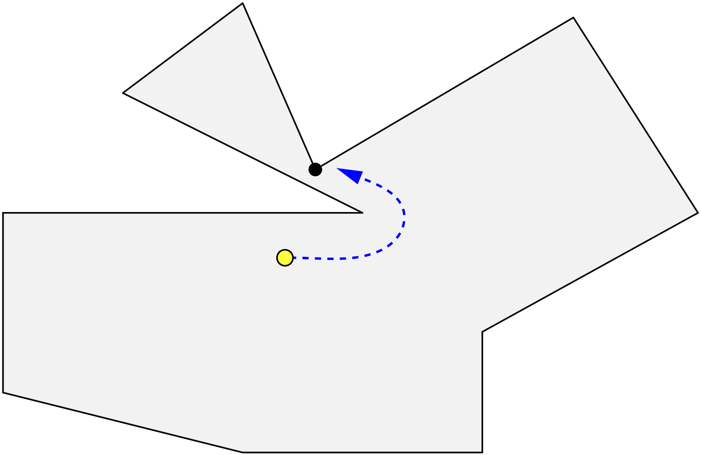
and the searcher has to move around it.

Traveling Around Reflex Vertices



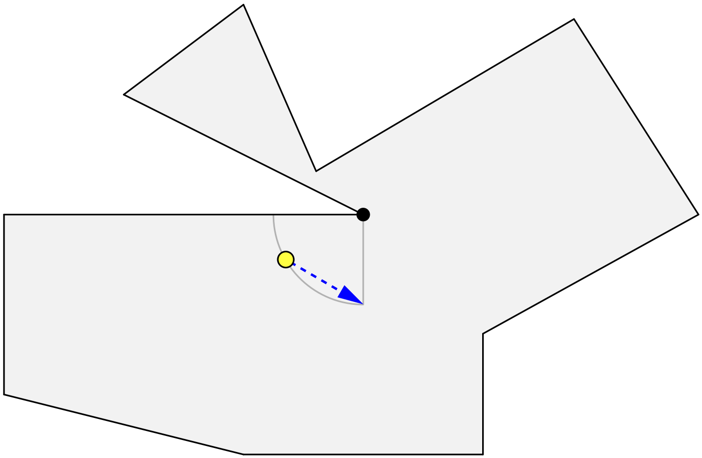
This may accidentally cause the virtual vertex to change.

Traveling Around Reflex Vertices



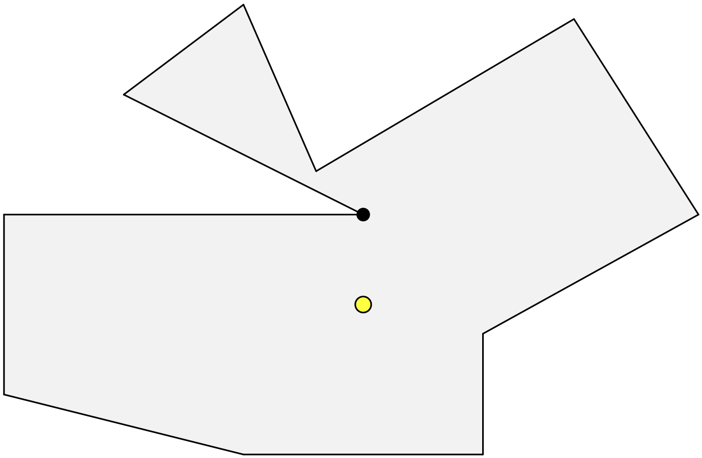
So the searcher has to carefully devise a series of moves.

Traveling Around Reflex Vertices



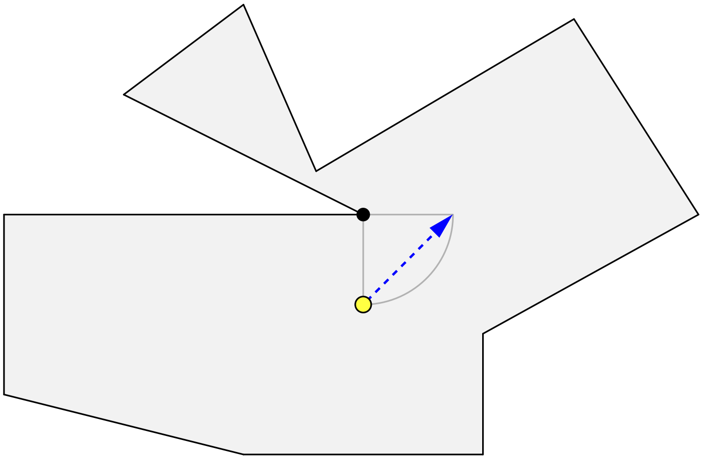
First it moves to the line perpendicular to the visible edge,

Traveling Around Reflex Vertices



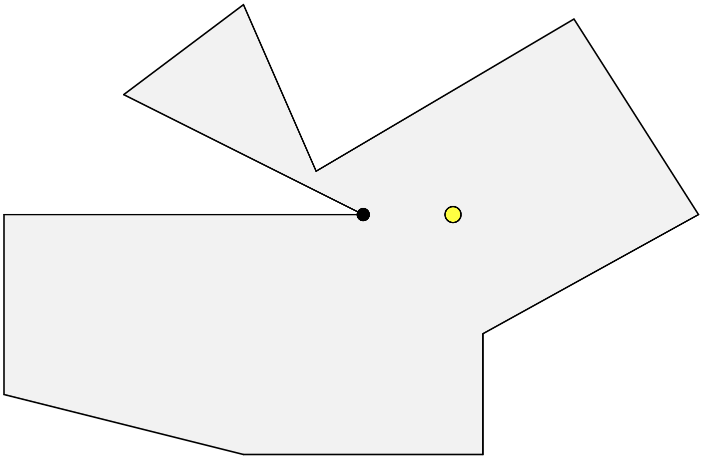
maintaining its distance from the virtual vertex.

Traveling Around Reflex Vertices



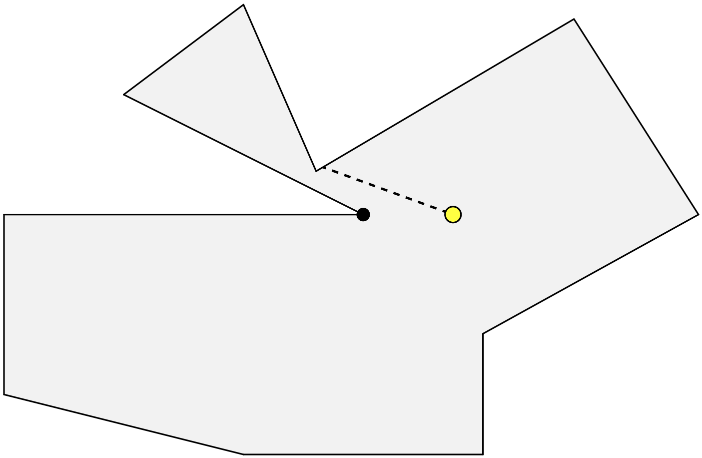
Then it moves to the extension of the visible edge,

Traveling Around Reflex Vertices



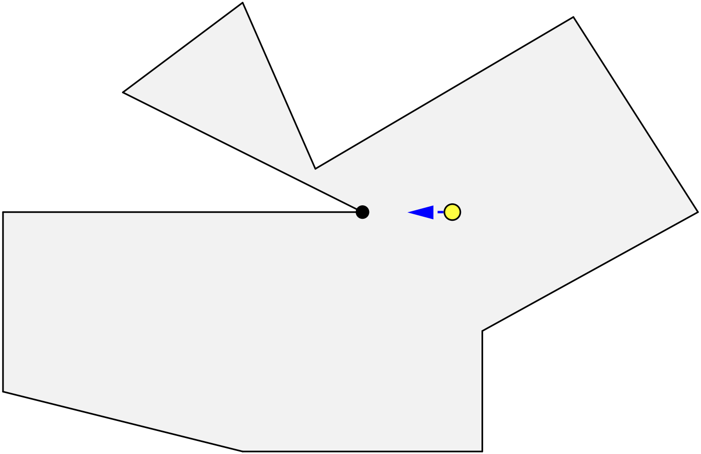
always maintaining the same distance.

Traveling Around Reflex Vertices



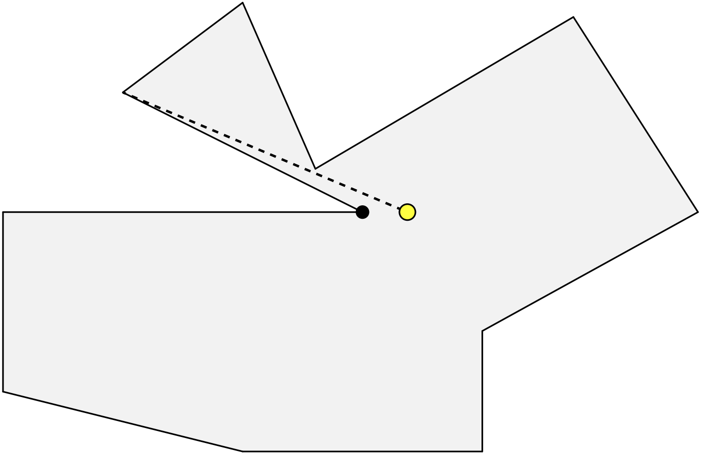
If the other edge is not completely visible,

Traveling Around Reflex Vertices



it halves its distance from the virtual vertex

Traveling Around Reflex Vertices



until both its adjacent vertices are visible.

Representing Composite Data Structures

$$0.\color{blue}{34672345}\ldots \qquad 0.\color{red}{17838946}\ldots$$

We want to represent arbitrary data as a single real number.

Representing Composite Data Structures

$$0.\textcolor{blue}{34672345}\ldots \qquad 0.\textcolor{red}{17838946}\ldots$$

This boils down to "merging" two real numbers into one.

Representing Composite Data Structures

$$0.34672345\ldots \qquad 0.17838946\ldots$$

$$\downarrow$$

$$0.3147687328394456\ldots$$

A naive approach would be to interleave their digits.

Representing Composite Data Structures

$$0.34672345\ldots \qquad 0.17838946\ldots$$

$$0.3147687328394456\ldots$$

Unfortunately, this function is not computable by a real RAM,

Representing Composite Data Structures

$$0.34672345\ldots \qquad 0.17838946\ldots$$

$$0.3147687328394456\ldots$$

because its discontinuities are everywhere dense in its domain.

Representing Composite Data Structures

$0.34672345 \qquad 0.17838946$

$\downarrow$

0.3147687328394456

However, if the numbers have finitely many digits,

Representing Composite Data Structures

$0.34672345 \qquad 0.17838946$

0.3147687328394456

this function is computable with arithmetic operations only!

Representing Composite Data Structures

$$k\text{th real root of } a_n x^n + a_{n+1} x^{n+1} + \cdots + a_1 x + a_0$$

$$\downarrow$$

$$(k, a_n, a_{n+1}, \ldots, a_1, a_0) \in \mathbb{Z}^{n+2}$$

This allows us to represent (sequences of) *algebraic* numbers

Representing Composite Data Structures

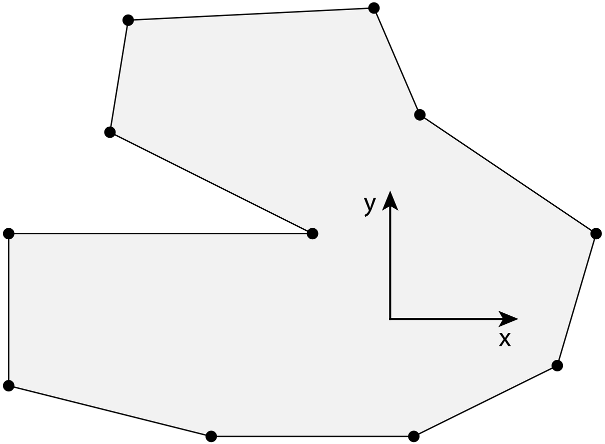
$$k\text{th real root of } a_n x^n + a_{n+1} x^{n+1} + \cdots + a_1 x + a_0$$

$$\Downarrow$$

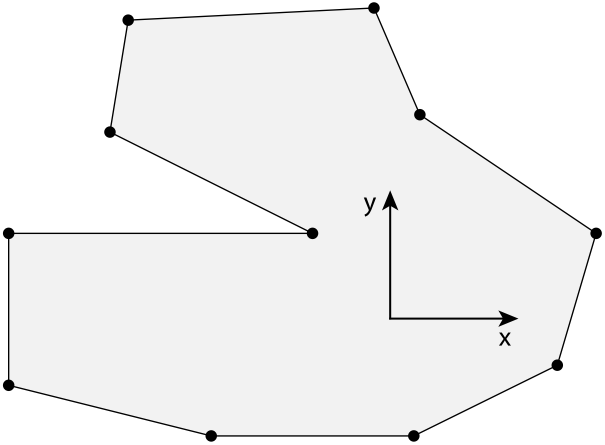
$$(k, a_n, a_{n+1}, \ldots, a_1, a_0) \in \mathbb{Z}^{n+2}$$

and do exact computations on them with standard techniques.

Representing Composite Data Structures

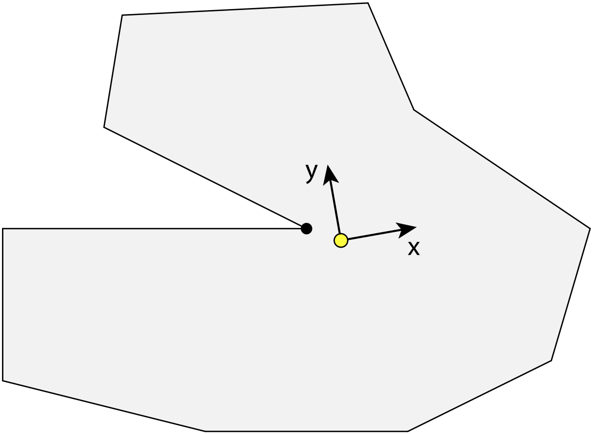


We stipulate that the polygon's vertices are algebraic points



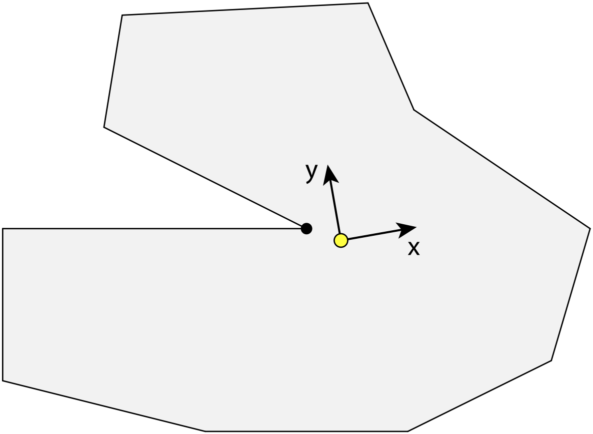
as expressed in some global coordinate system.

Virtual Coordinate Systems



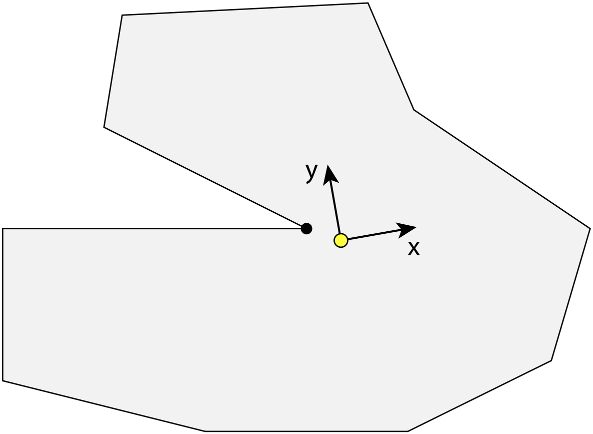
Unfortunately, each searcher has its own coordinate system,

Virtual Coordinate Systems



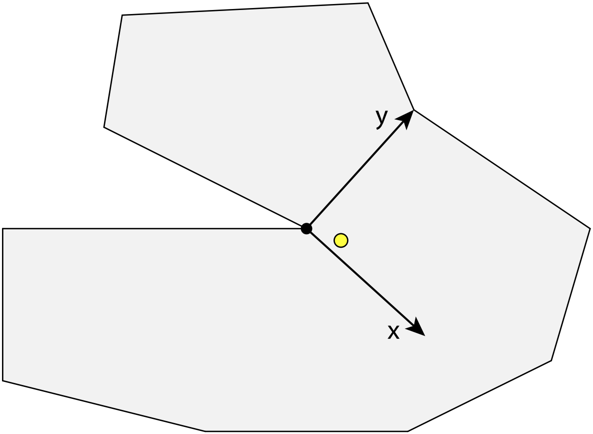
in which the vertices may not be algebraic points

Virtual Coordinate Systems



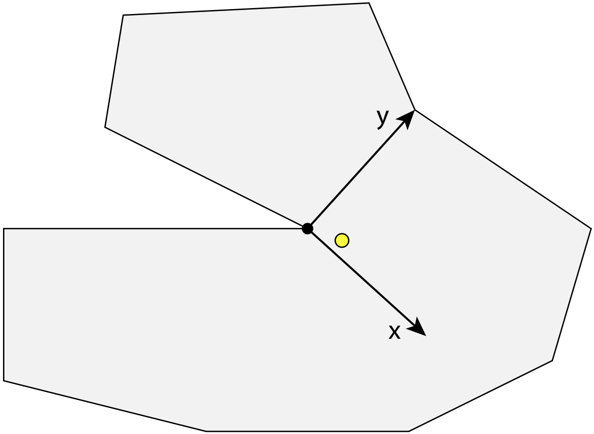
and may be impossible to memorize with our method!

Virtual Coordinate Systems



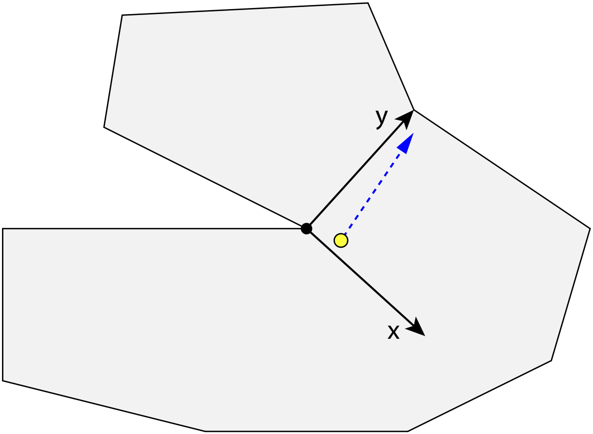
To cope with this, the searcher constructs a *virtual* system.

Virtual Coordinate Systems



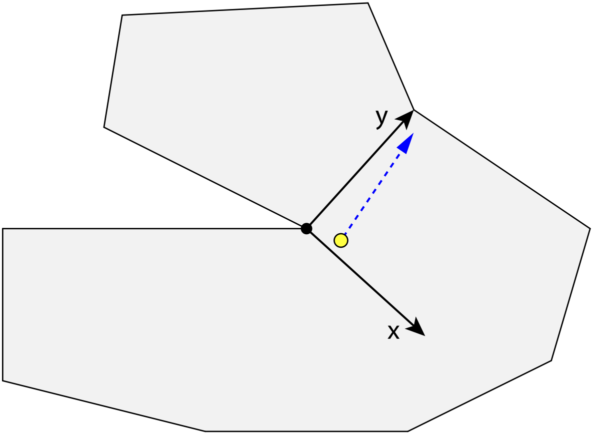
With its current virtual vertex as the origin

Virtual Coordinate Systems



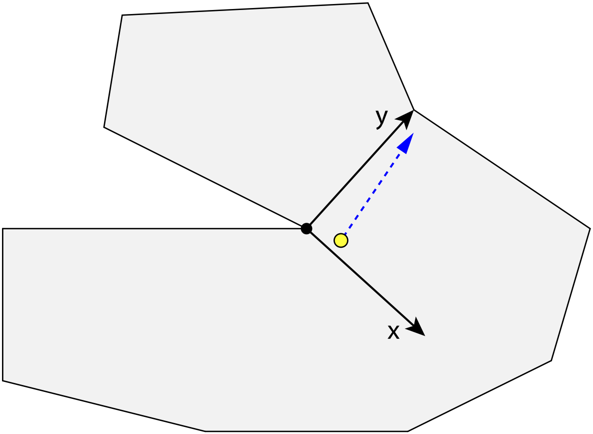
and its next destination vertex at unit distance on the y axis.

Virtual Coordinate Systems



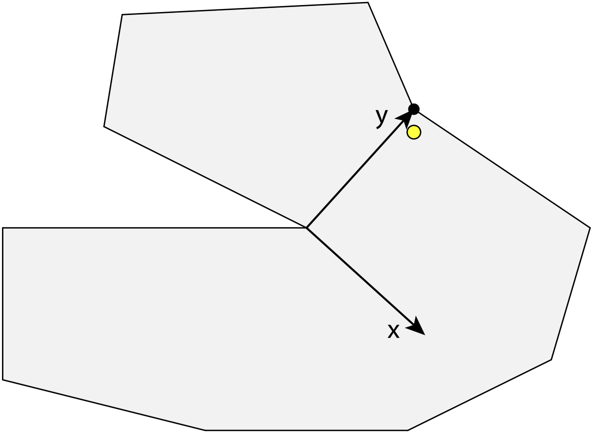
In this coordinate system, all vertices are again algebraic,

Virtual Coordinate Systems



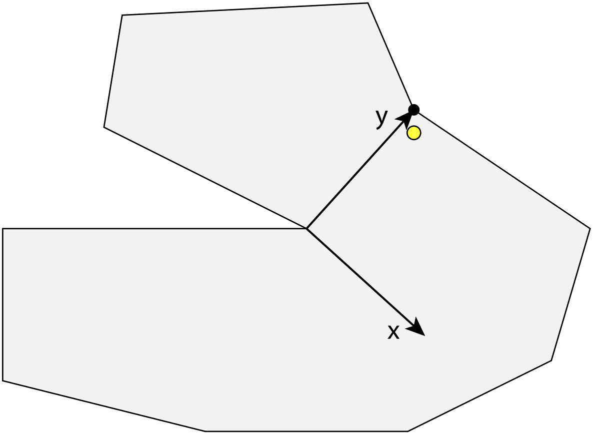
and therefore can be encoded as a single real number!

Virtual Coordinate Systems



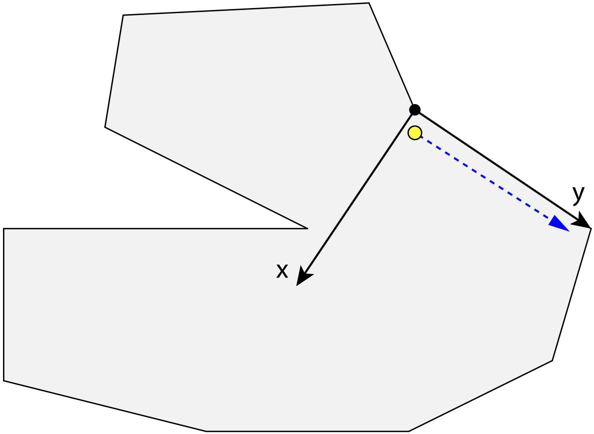
When the searcher moves to another virtual vertex,

Virtual Coordinate Systems



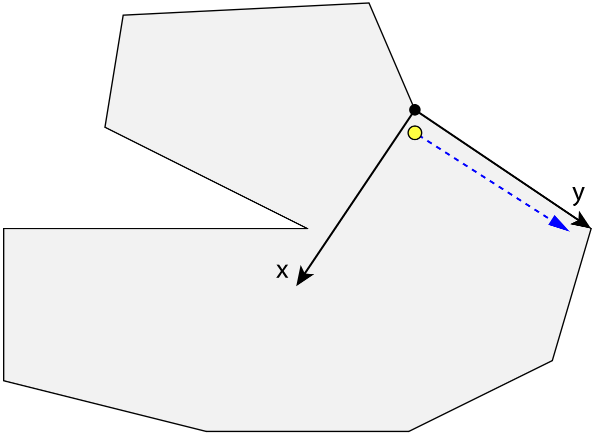
it reconstructs the old coordinate system to decode all the data,

Virtual Coordinate Systems



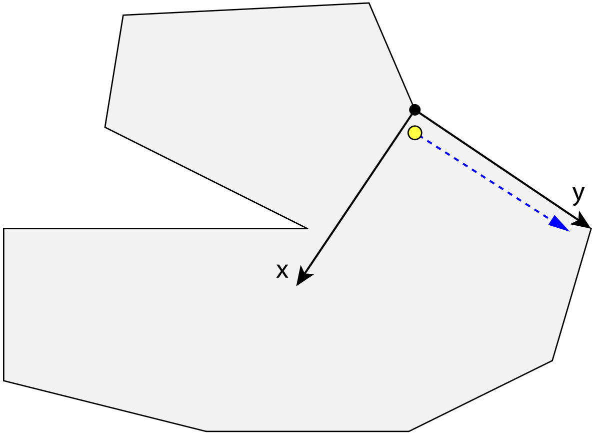
then it constructs the new virtual coordinate system,

Virtual Coordinate Systems



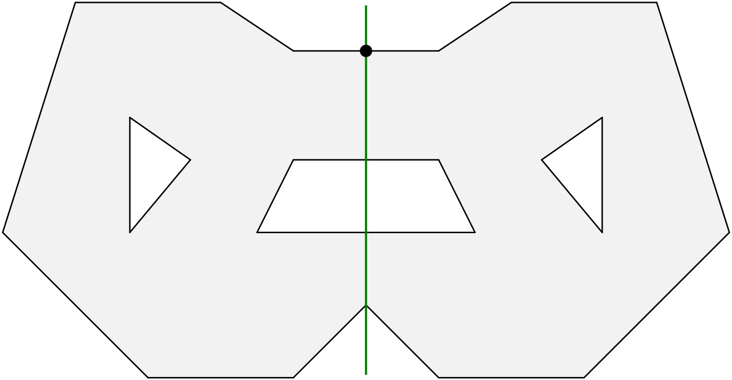
converting the data into the new one,

Virtual Coordinate Systems



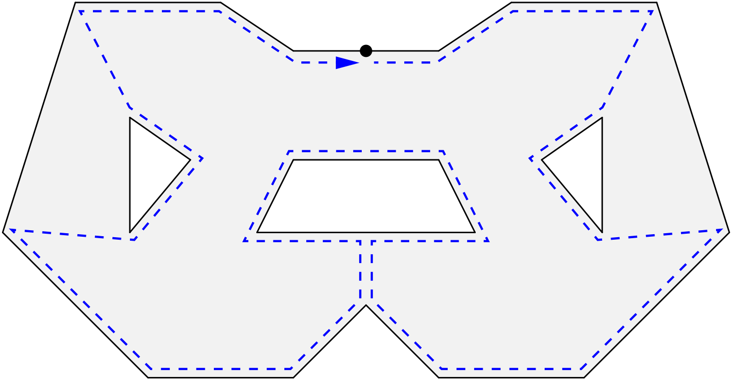
and computing the exact destination point accordingly.

Modifying Patrol Routes



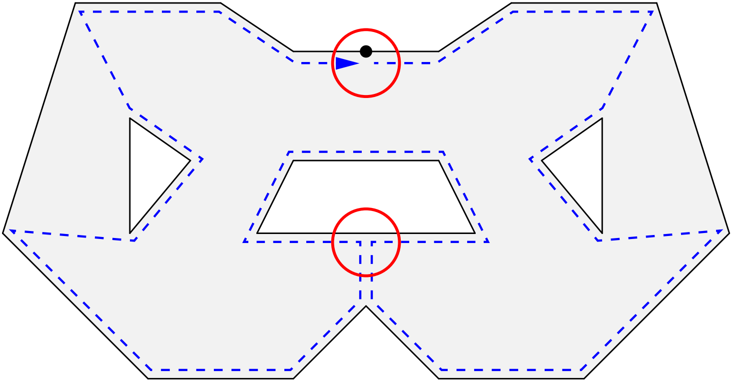
In our algorithms, a searcher may have to stop at points

Modifying Patrol Routes



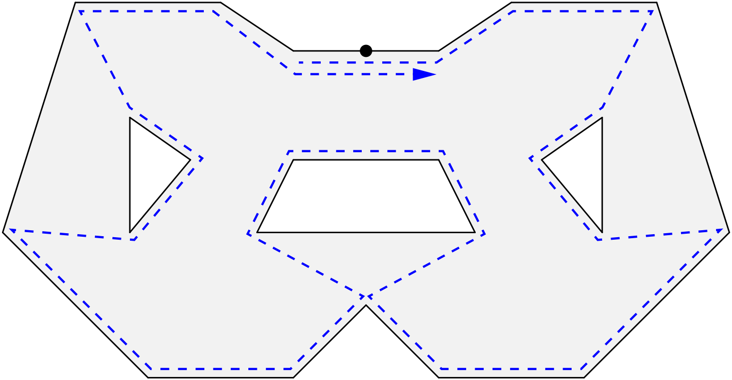
that are not vertices of the polygon.

Modifying Patrol Routes



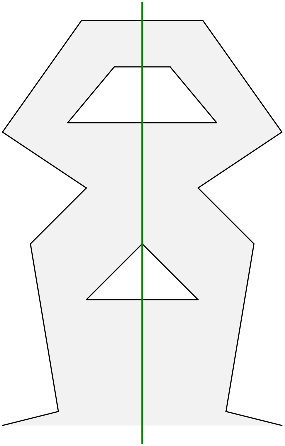
But a searcher always has to stop close to its virtual vertex!

Modifying Patrol Routes



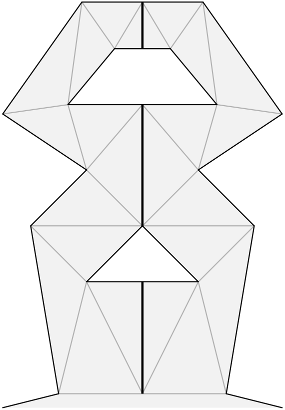
So we modify its patrol route, making it turn only at vertices.

Modifying Patrol Routes



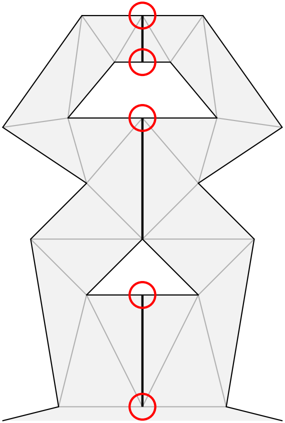
In the improved Meeting algorithm, this is more complicated,

Modifying Patrol Routes



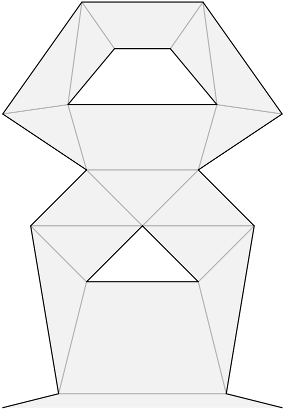
because the augmented polygon has to be triangulated

Modifying Patrol Routes



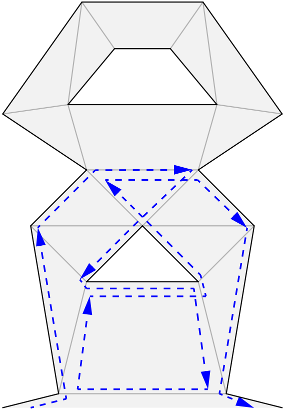
and the triangle's vertices are not always vertices of the polygon.

Modifying Patrol Routes



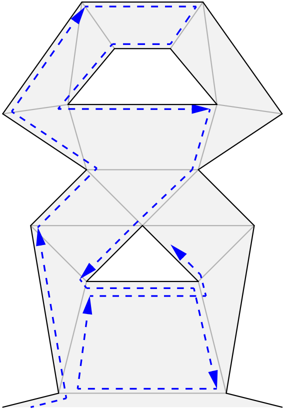
Hence, along with triangles, we also use isosceles trapezoids,

Modifying Patrol Routes



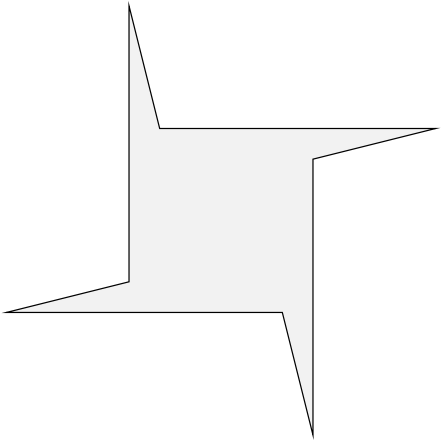
and we make the k-tours turn only at vertices.

Modifying Patrol Routes



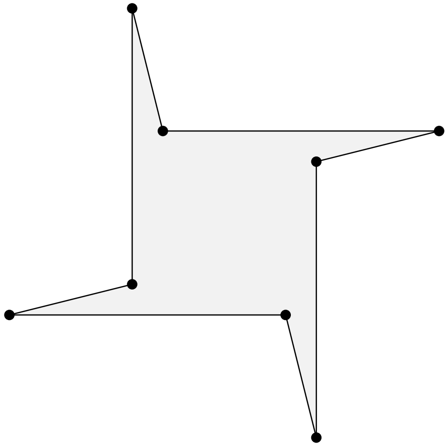
and we make the k-tours turn only at vertices.

Preserving Total Visibility



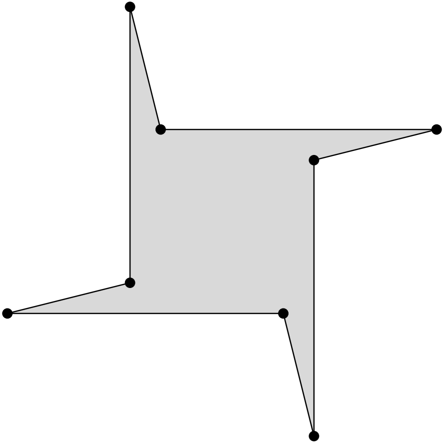
Searchers need to see the entire polygon during their patrol,

Preserving Total Visibility



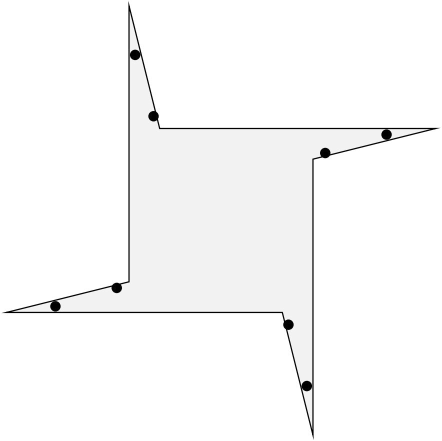
otherwise they may be unable to tell if their map is correct.

Preserving Total Visibility



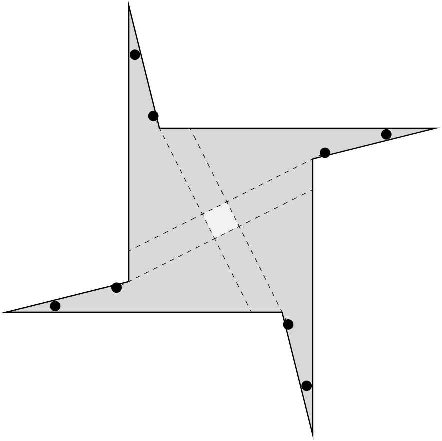
They can do it if they visit all vertices,

Preserving Total Visibility



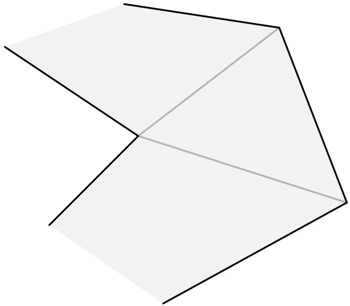
but now they are only getting close to their virtual vertices!

Preserving Total Visibility



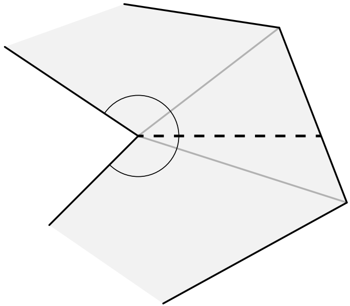
This may prevent them from seeing the entire polygon.

Preserving Total Visibility



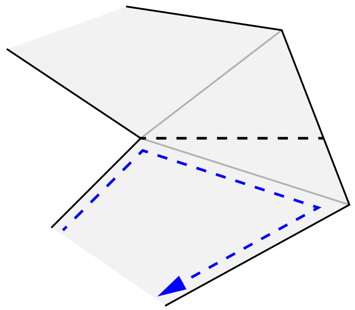
An easy way to avoid this situation

Preserving Total Visibility



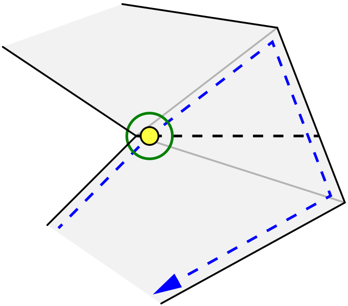
is to stop on the angle bisector of every vertex.

Preserving Total Visibility



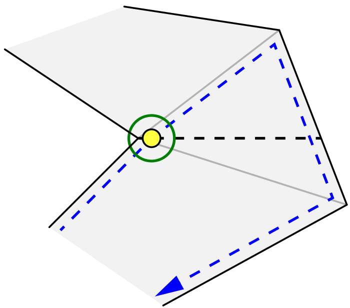
This may not be possible on every k-tour of the polygon.

Preserving Total Visibility



But for every vertex, there is a k-tour where this is possible.

Preserving Total Visibility



Theorem: our algorithms work also with memoryless searchers.

Results on the Meeting Problem

The following results hold even for memoryless searchers:

(assuming the polygon's vertices are algebraic points)

- If the polygon's *symmetricity* is σ, then $\sigma + 1$ searchers are always sufficient and sometimes necessary.
- If the polygon's center is not in a hole, 2 searchers are enough.

  (this includes all polygons with no holes)

Destination points are geometrically constructible using a compass only.