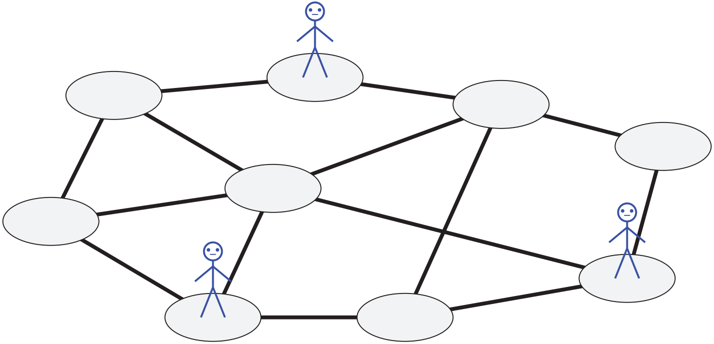
Universal Systems of Oblivious Mobile Robots

SIROCCO 2016

Paola Flocchini, Nicola Santoro,
Giovanni Viglietta, Masafumi Yamashita

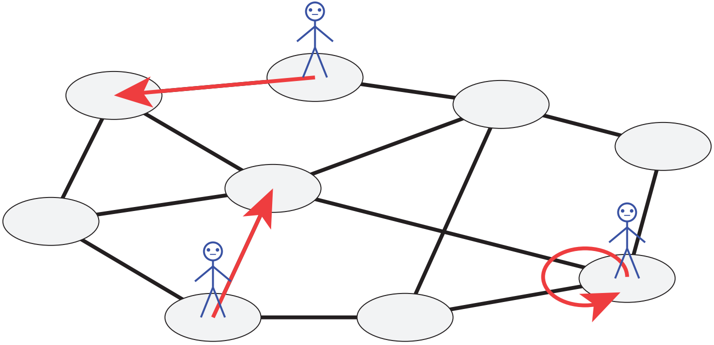
Helsinki – July 20, 2016

Model: anonymous robots moving on a network



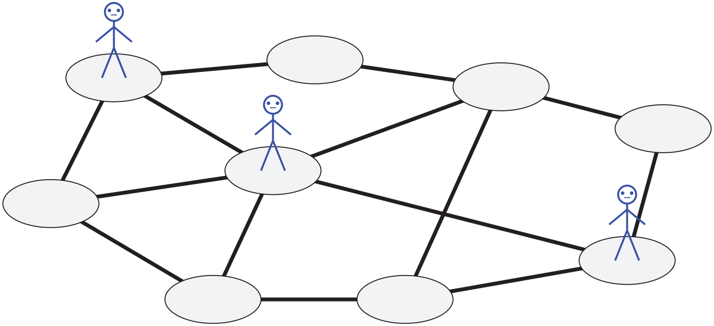
We consider a set of anonymous *robots* living on a *network*.

Model: anonymous robots moving on a network



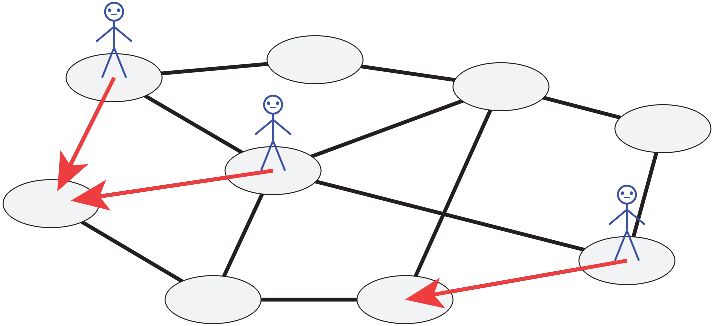
At each *round*, all robots move simultaneously to an adjacent vertex (or stay still).

Model: anonymous robots moving on a network



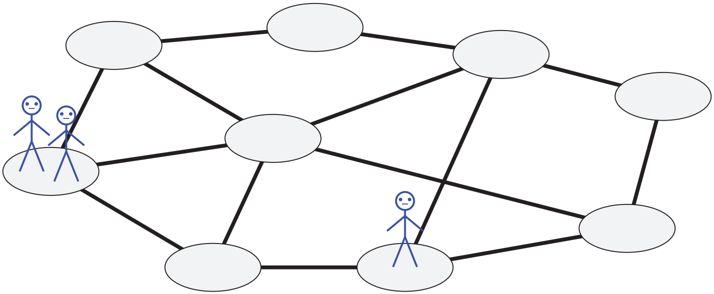
At each *round*, all robots move simultaneously to an adjacent vertex (or stay still).

Model: anonymous robots moving on a network



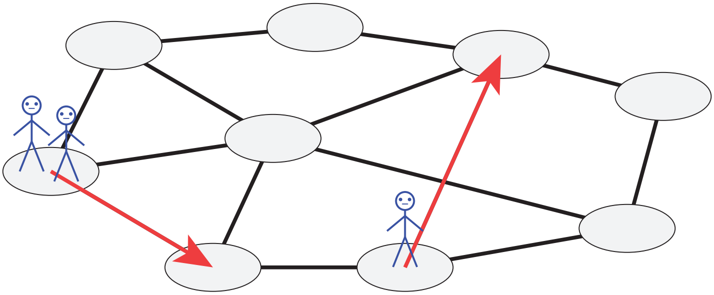
A robot's destination is computed via a deterministic algorithm whose input is only the current configuration (*obliviousness*).

Model: anonymous robots moving on a network



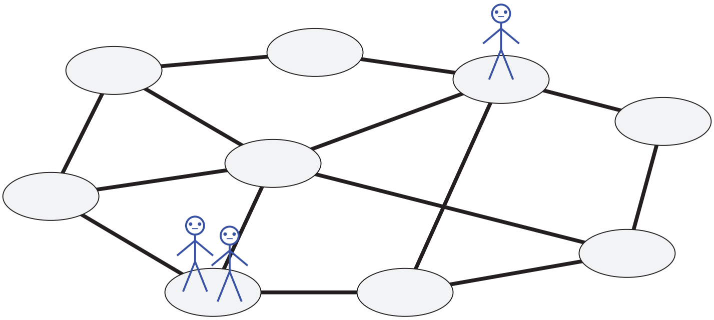
A robot's destination is computed via a deterministic algorithm whose input is only the current configuration (*obliviousness*).

Model: anonymous robots moving on a network



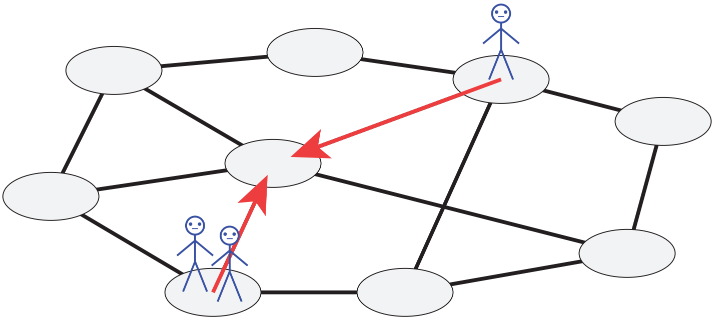
Several robots may occupy the same vertex.

Model: anonymous robots moving on a network



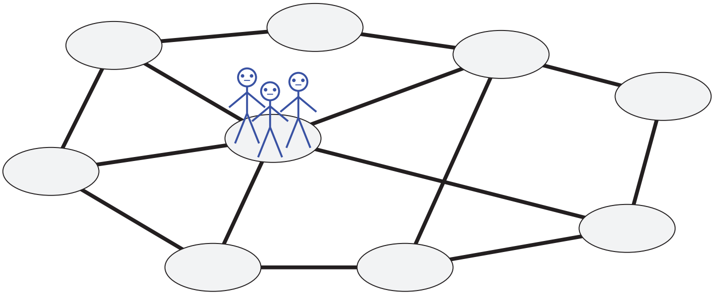
Several robots may occupy the same vertex.

Model: anonymous robots moving on a network



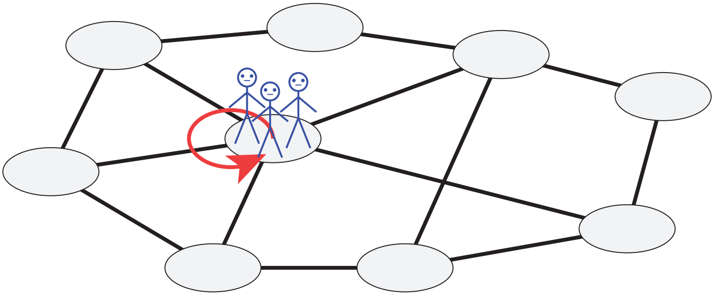
A basic well-studied problem is *gathering*.

Model: anonymous robots moving on a network



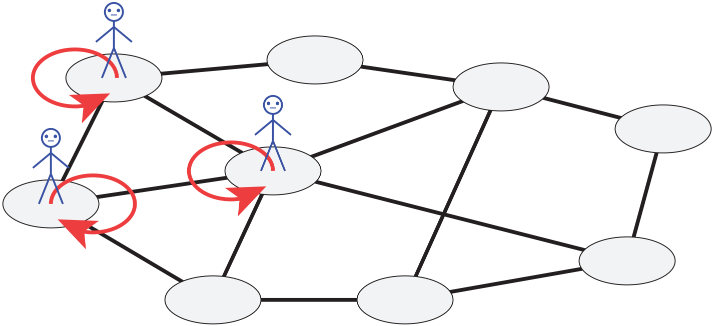
A basic well-studied problem is *gathering*.

Model: anonymous robots moving on a network



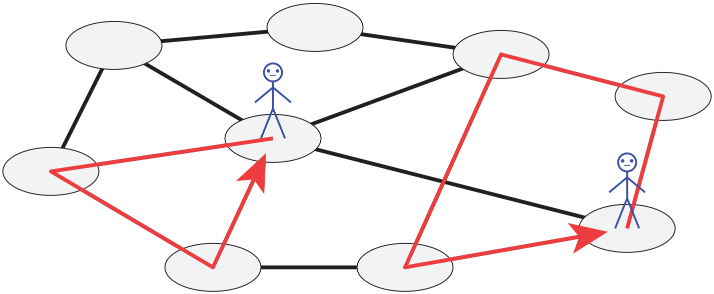
A basic well-studied problem is *gathering*.

Model: anonymous robots moving on a network



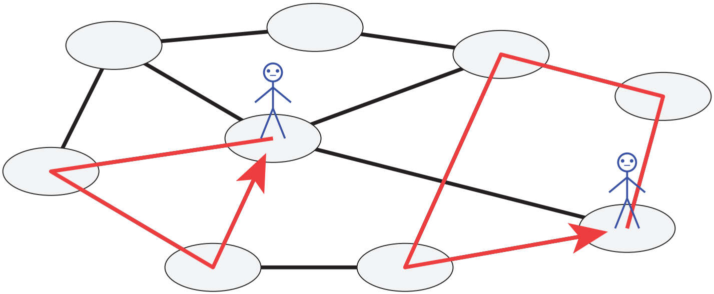
A more general problem is *pattern formation* (e.g., a triangle).

Model: anonymous robots moving on a network



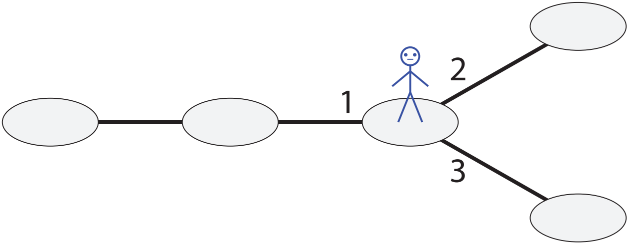
Another common task may be to implement a self-stabilizing *clock* with a specific period (e.g., 12).

Model: anonymous robots moving on a network



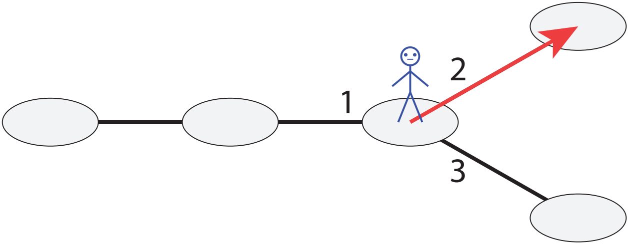
Our goal is to define what it means for such a system to *compute a function*, and determine what functions it can compute.

Model: port labeling



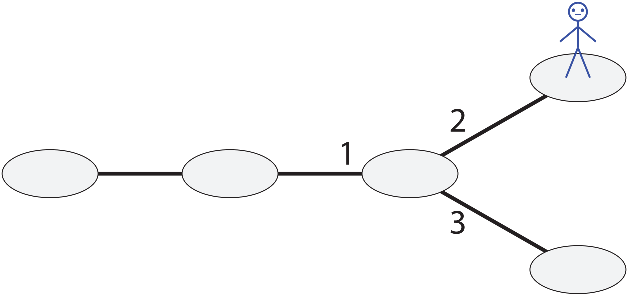
Each vertex of the network has *port labels* on incident edges.

Model: port labeling



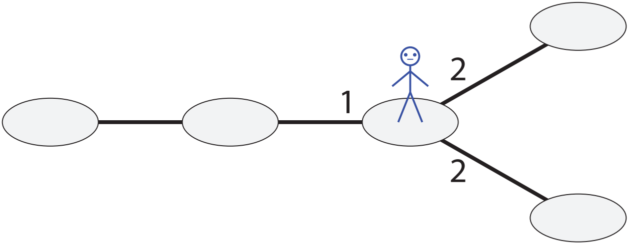
Each vertex of the network has *port labels* on incident edges.

Model: port labeling



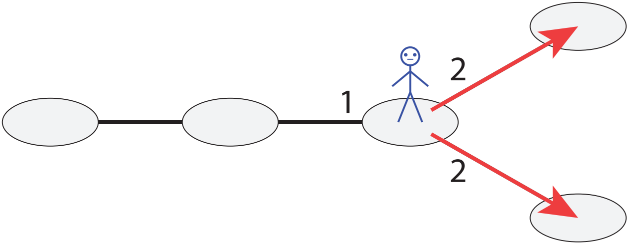
Each vertex of the network has *port labels* on incident edges.

Model: port labeling



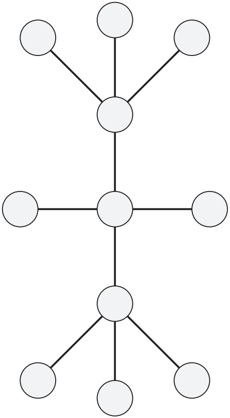
Labels need not be unique. This may cause non-determinism in executions (even though algorithms are deterministic!).

Model: port labeling



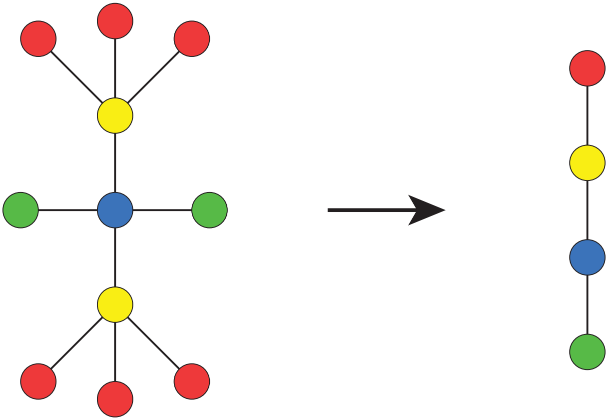
Labels need not be unique. This may cause non-determinism in executions (even though algorithms are deterministic!).

Equivalent configurations



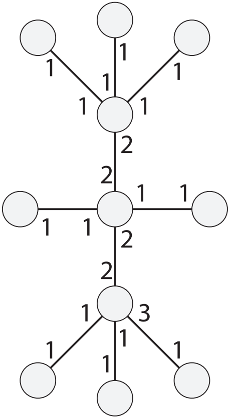
Identifying *equivalent* vertices yields the *quotient graph*.

Equivalent configurations



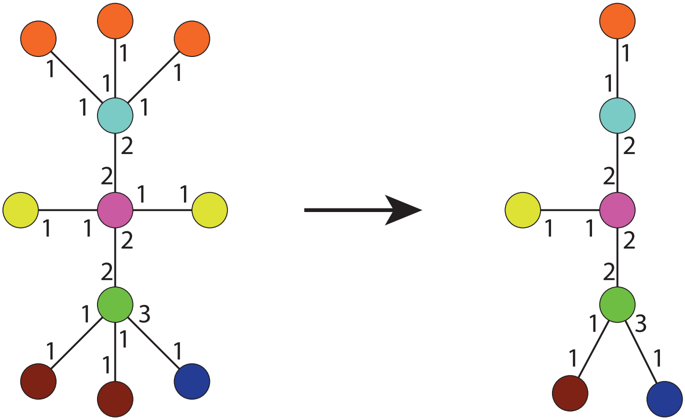
Identifying *equivalent* vertices yields the *quotient graph*.

Equivalent configurations



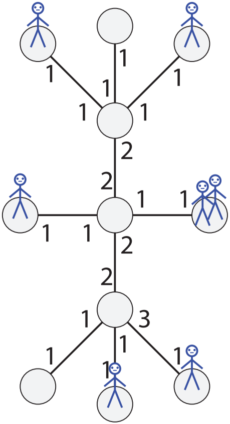
Vertices can also be distinguished by their surrounding labels...

Equivalent configurations



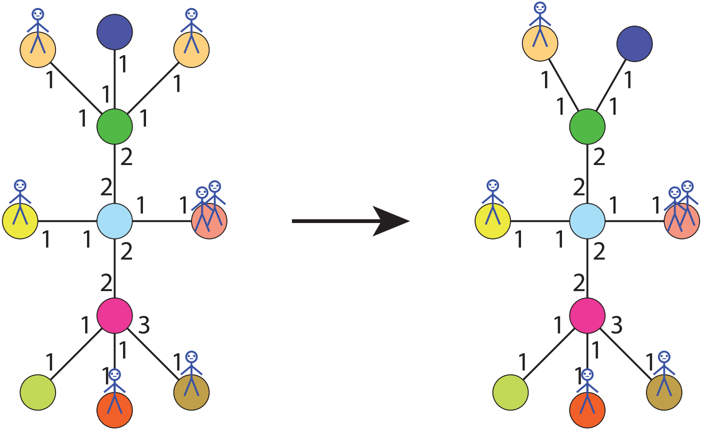
Vertices can also be distinguished by their surrounding labels...

Equivalent configurations



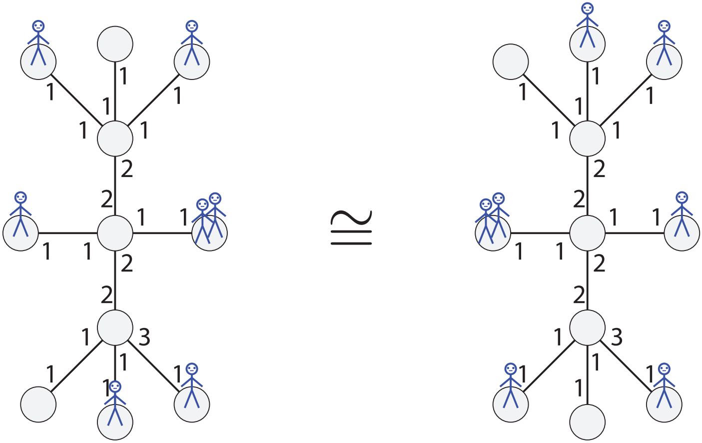
...And by the amount of robots that occupy them.

Equivalent configurations



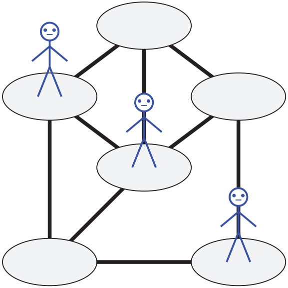
...And by the amount of robots that occupy them.

Equivalent configurations



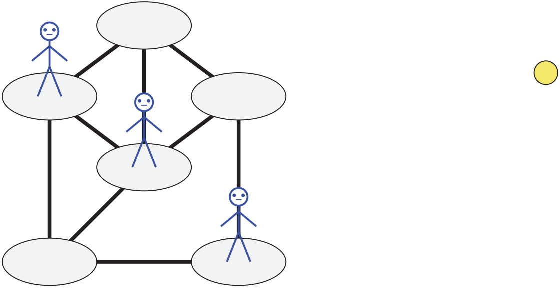
Vertices and robots are otherwise *indistinguishable*.

Configuration graph



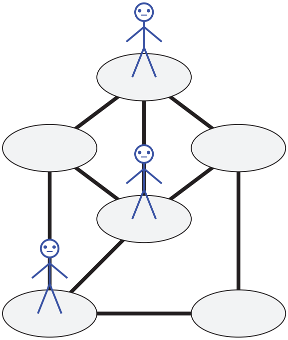
Non-equivalent configurations can be arranged in a graph.

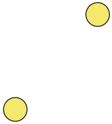
Configuration graph



Non-equivalent configurations can be arranged in a graph.

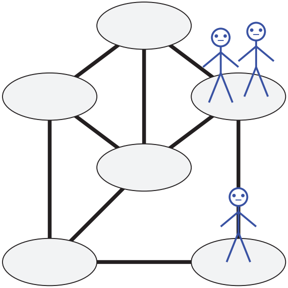
Configuration graph

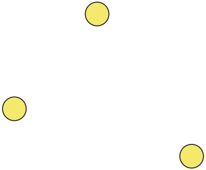




Non-equivalent configurations can be arranged in a graph.

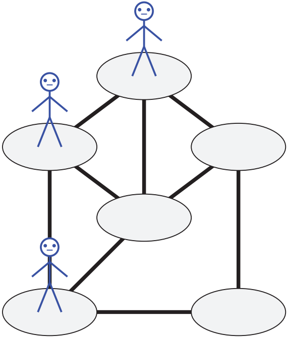
Configuration graph

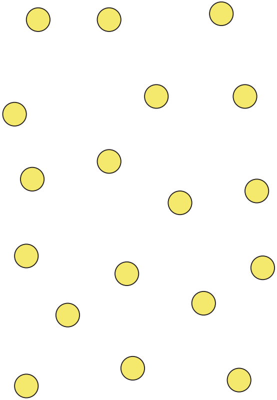




Non-equivalent configurations can be arranged in a graph.

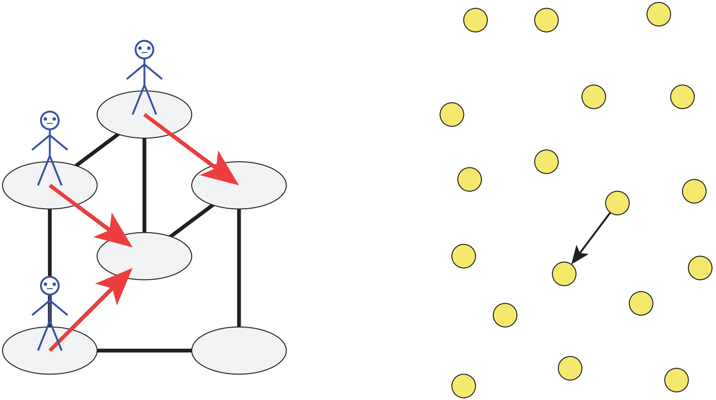
Configuration graph





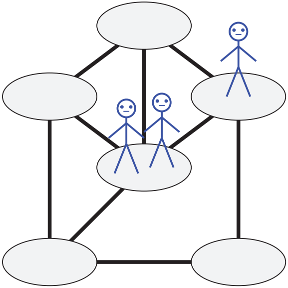
Non-equivalent configurations can be arranged in a graph.

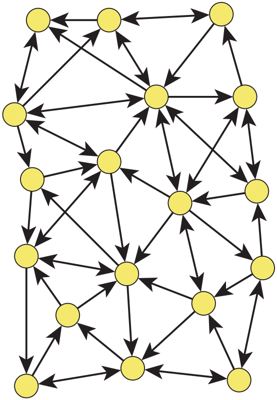
Configuration graph



An edge in the graph means that two configurations are connected by a simultaneous move of all the robots.

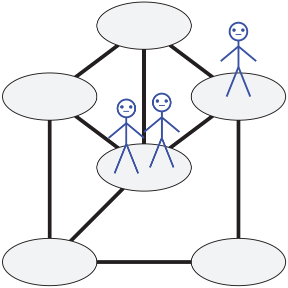
Configuration graph

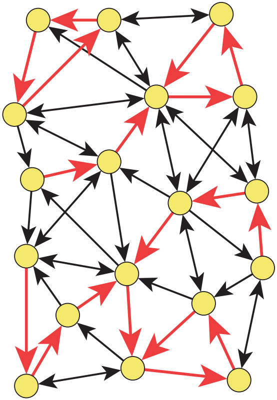




An edge in the graph means that two configurations are connected by a simultaneous move of all the robots.

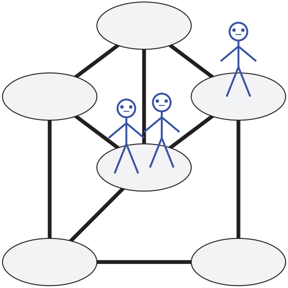
Configuration graph

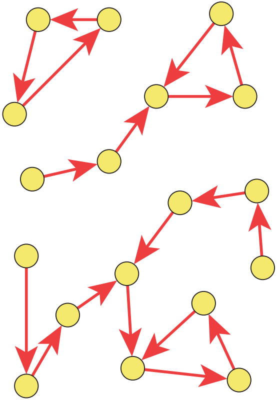




An *algorithm* is a selection of an outgoing edge for each vertex of the *configuration graph*.

Configuration graph



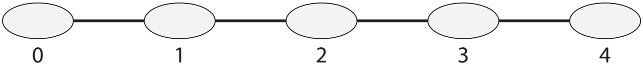


An *algorithm* is a selection of an outgoing edge for each vertex of the *configuration graph*.

Example: oriented path

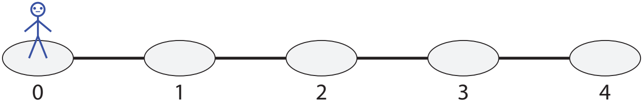
Let us study the configuration graph of a *path* with left-right *orientation*.

Example: oriented path



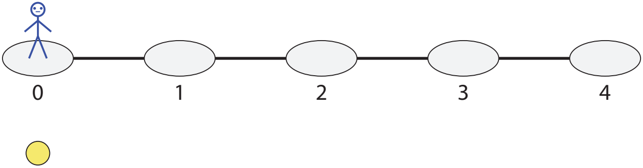
Let us study the configuration graph of a *path* with left-right *orientation*.

Example: oriented path



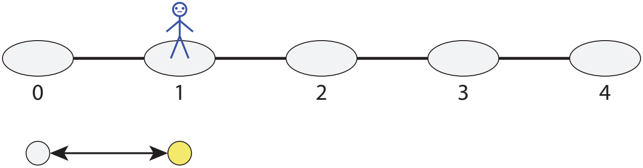
If there is only one robot in the network, the configuration graph is a path of the same length.

Example: oriented path



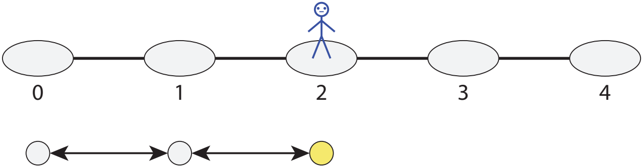
If there is only one robot in the network, the configuration graph is a path of the same length.

Example: oriented path



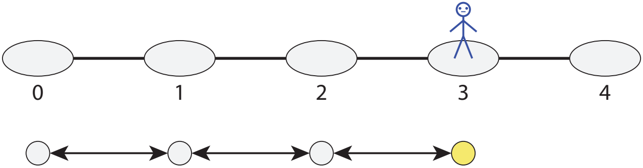
If there is only one robot in the network, the configuration graph is a path of the same length.

Example: oriented path



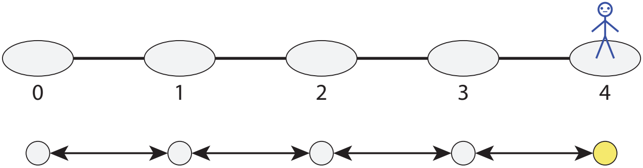
If there is only one robot in the network, the configuration graph is a path of the same length.

Example: oriented path



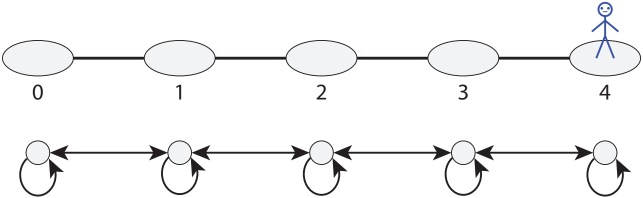
If there is only one robot in the network, the configuration graph is a path of the same length.

Example: oriented path



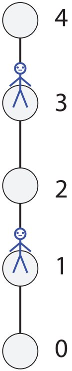
If there is only one robot in the network, the configuration graph is a path of the same length.

Example: oriented path



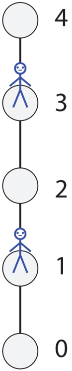
If there is only one robot in the network, the configuration graph is a path of the same length.

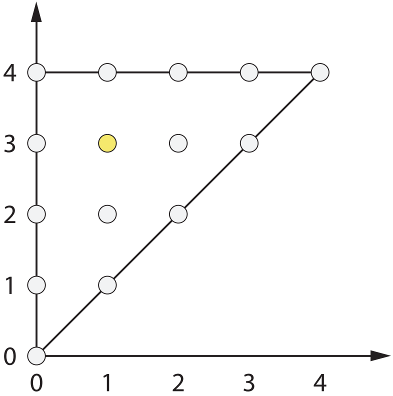
Example: oriented path



If there are two robots, the configuration graph is a triangular mesh.

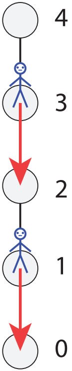
Example: oriented path

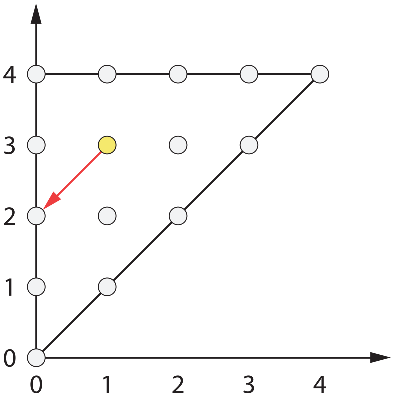




If there are two robots, the configuration graph is a triangular mesh.

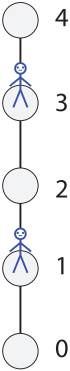
Example: oriented path

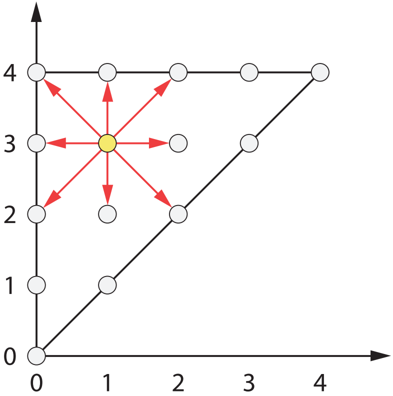




If there are two robots, the configuration graph is a triangular mesh.

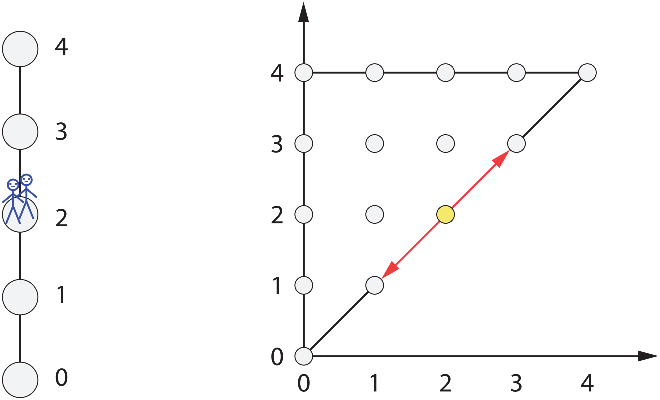
Example: oriented path





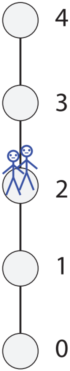
If there are two robots, the configuration graph is a triangular mesh.

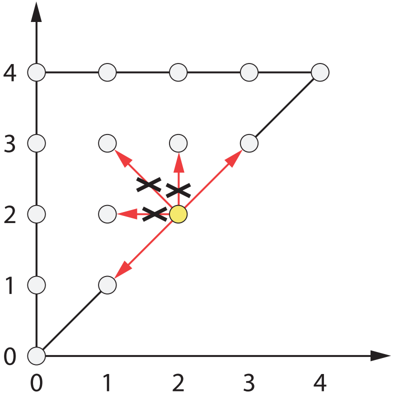
Example: oriented path



When the two robots meet, they can no longer be separated.

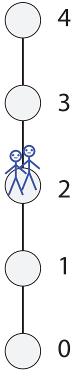
Example: oriented path

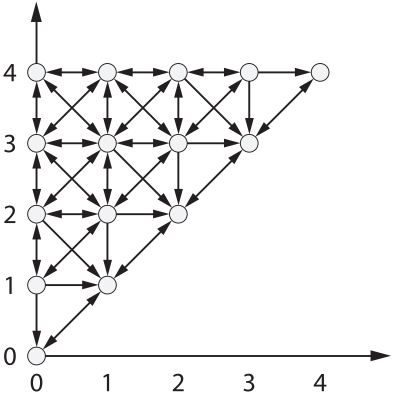




When the two robots meet, they can no longer be separated.

Example: oriented path



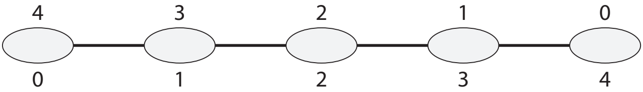


When the two robots meet, they can no longer be separated.

Example: unoriented path

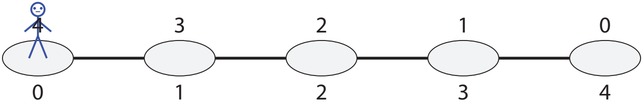
Suppose now the path is *unoriented*.

Example: unoriented path



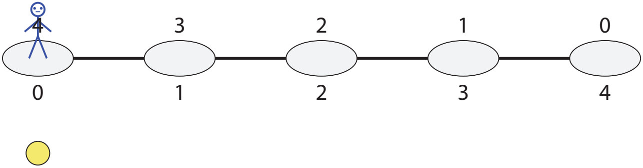
Suppose now the path is *unoriented*.

Example: unoriented path



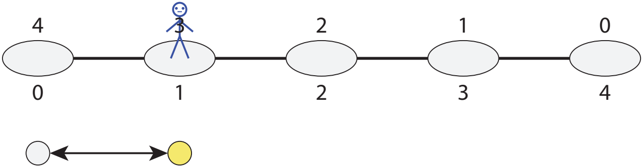
If there is only one robot in the network, the configuration graph is a path of roughly half the length.

Example: unoriented path



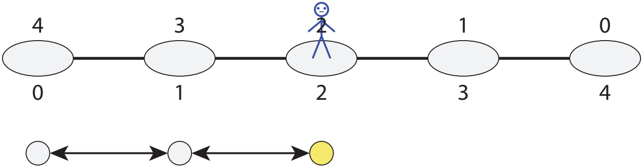
If there is only one robot in the network, the configuration graph is a path of roughly half the length.

Example: unoriented path



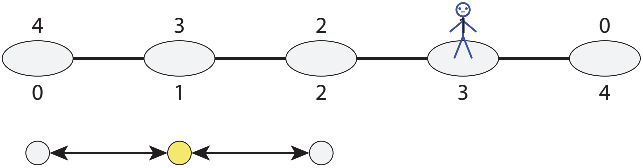
If there is only one robot in the network, the configuration graph is a path of roughly half the length.

Example: unoriented path



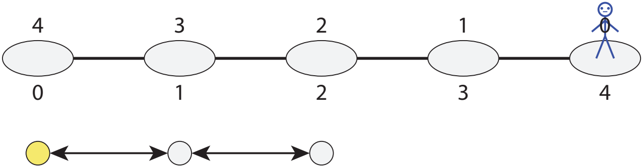
If there is only one robot in the network, the configuration graph is a path of roughly half the length.

Example: unoriented path



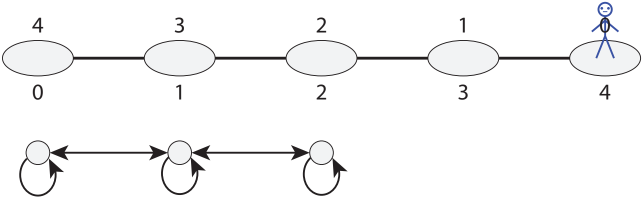
If there is only one robot in the network, the configuration graph is a path of roughly half the length.

Example: unoriented path



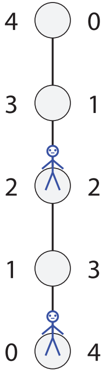
If there is only one robot in the network, the configuration graph is a path of roughly half the length.

Example: unoriented path



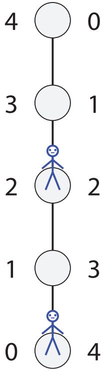
If there is only one robot in the network, the configuration graph is a path of roughly half the length.

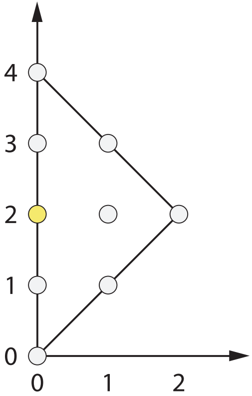
Example: unoriented path



If there are two robots, the configuration graph is a triangular mesh of roughly half the size.

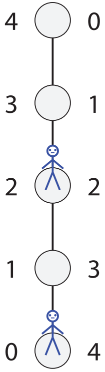
Example: unoriented path

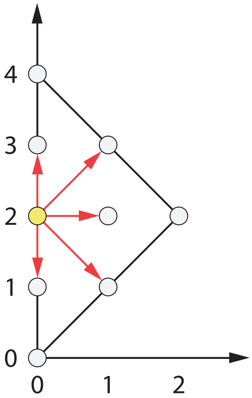




If there are two robots, the configuration graph is a triangular mesh of roughly half the size.

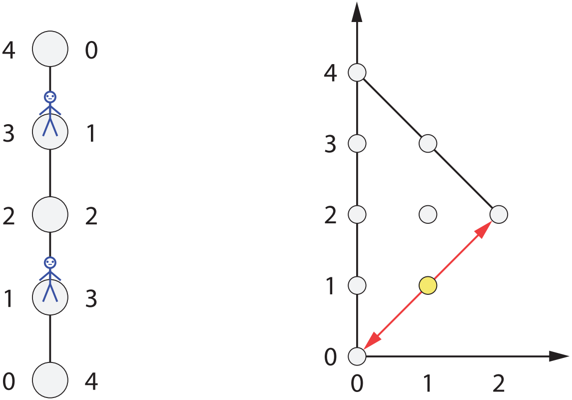
Example: unoriented path





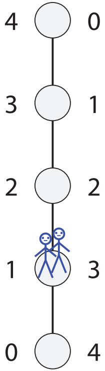
If there are two robots, the configuration graph is a triangular mesh of roughly half the size.

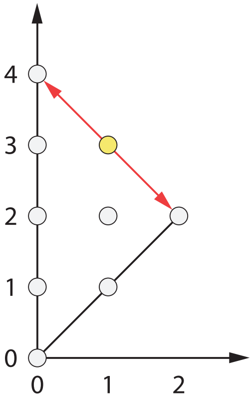
Example: unoriented path



If the robots are in distinct symmetric locations, they must move symmetrically.

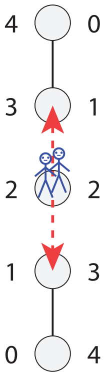
Example: unoriented path

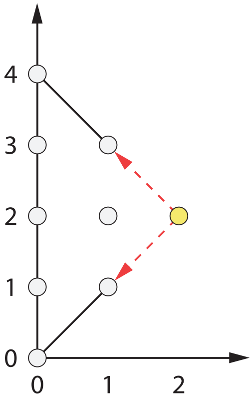




If the robots are on a same non-central vertex, they must remain together.

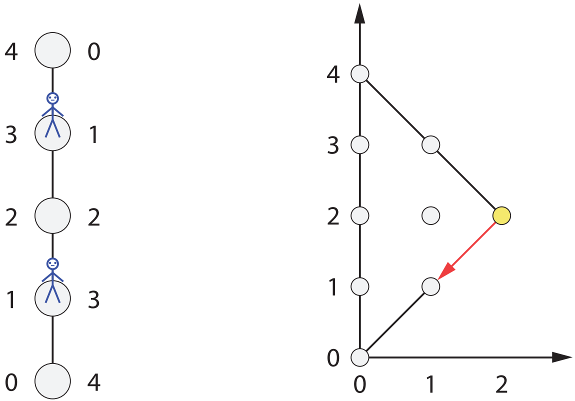
Example: unoriented path





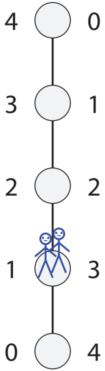
If both robots are on the central vertex, the symmetry of the network makes their next move *non-deterministic*.

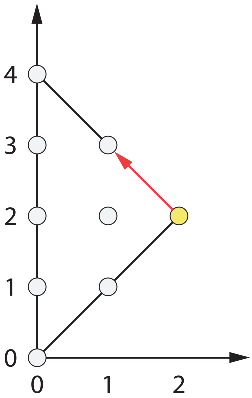
Example: unoriented path



If both robots are on the central vertex, the symmetry of the network makes their next move *non-deterministic*.

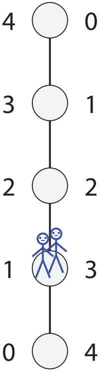
Example: unoriented path

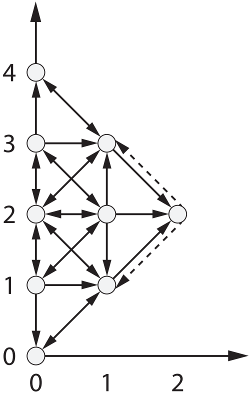




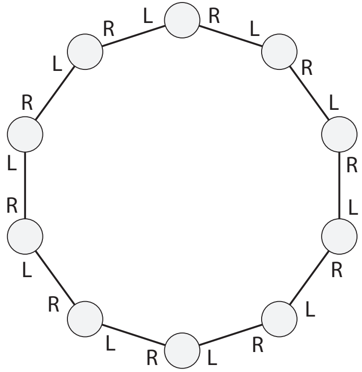
If both robots are on the central vertex, the symmetry of the network makes their next move *non-deterministic*.

Example: unoriented path



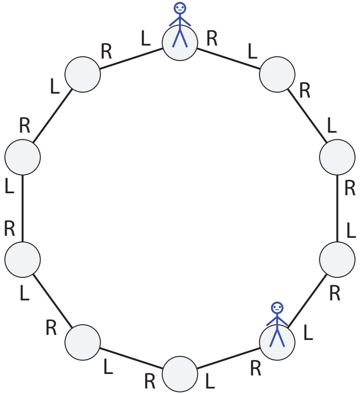


If both robots are on the central vertex, the symmetry of the network makes their next move *non-deterministic*.



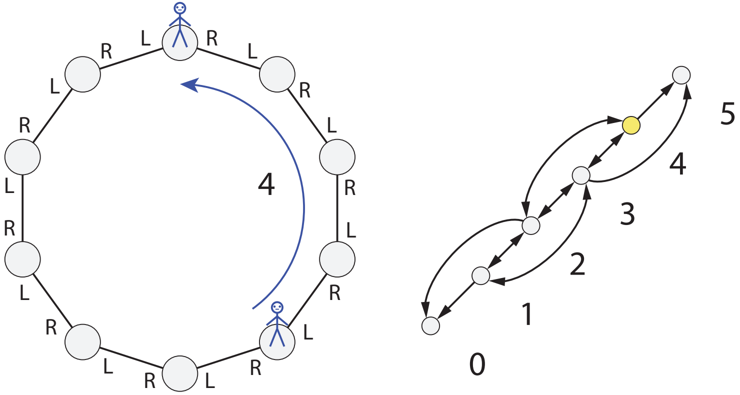
Let the network be a *ring* with left-right orientation.

Example: oriented ring



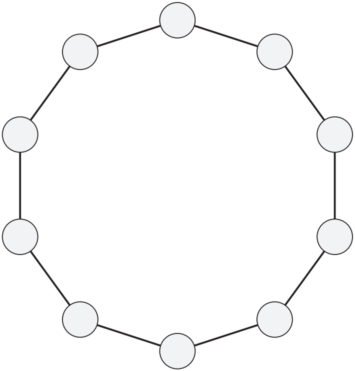
If two robots are on it, the configuration is fully determined by their distance.

Example: oriented ring



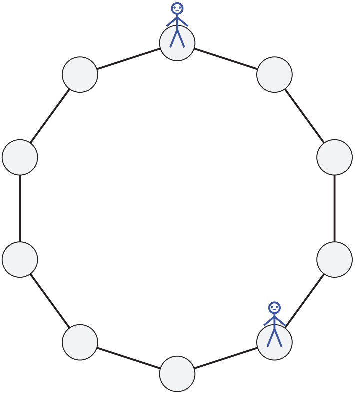
If two robots are on it, the configuration is fully determined by their distance.

Example: unoriented ring



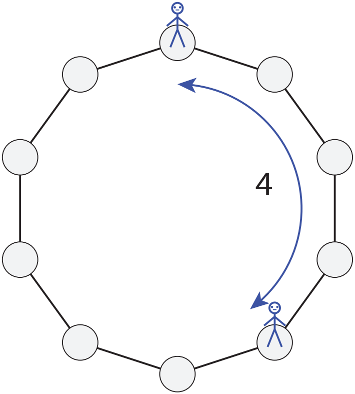
Let the network be an *unoriented ring*.

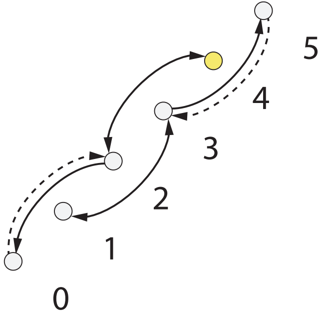
Example: unoriented ring



A configuration of two robots is again determined by their distance, but now fewer moves are possible.

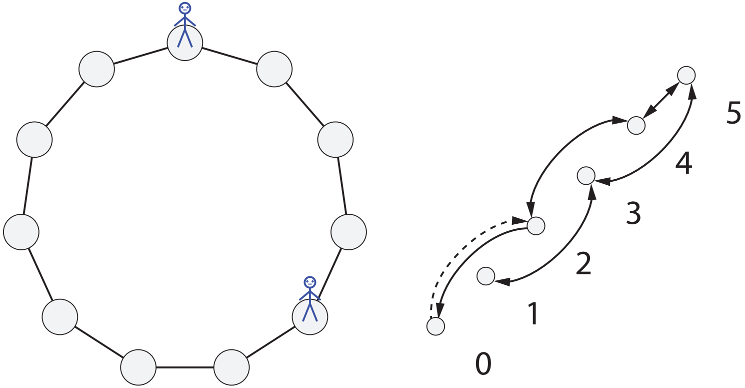
Example: unoriented ring





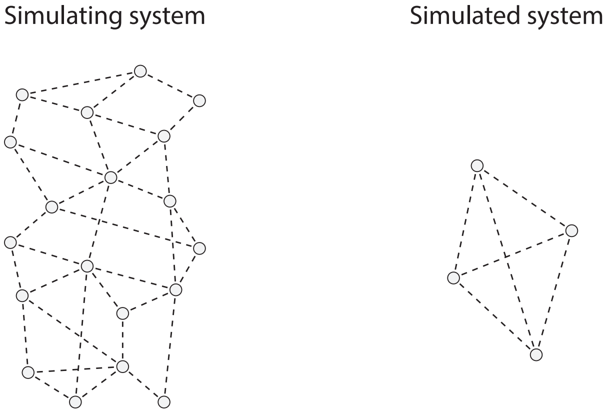
A configuration of two robots is again determined by their distance, but now fewer moves are possible.

Example: unoriented ring



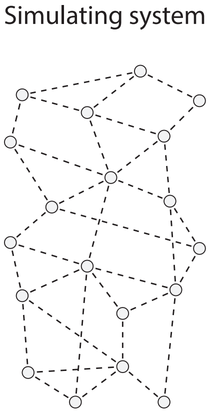
If the network's size is even, the configuration graph consists of two independent paths; if the size is odd, it consists of a single path.

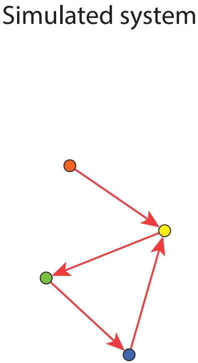
Simulating algorithms



What does it mean for an *algorithm* for a given system (network, swarm of robots) to *simulate* an algorithm for another system?

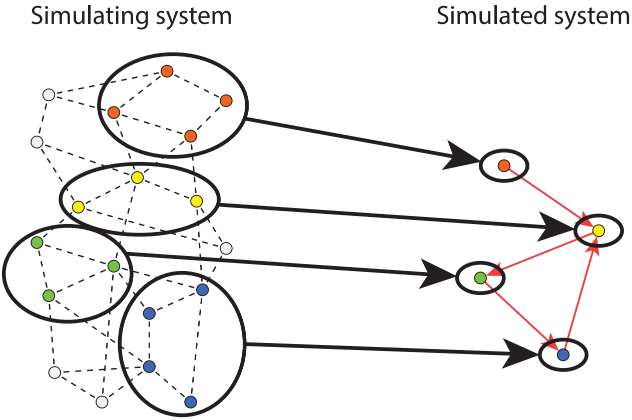
Simulating algorithms





What does it mean for an *algorithm* for a given system (network, swarm of robots) to *simulate* an algorithm for another system?

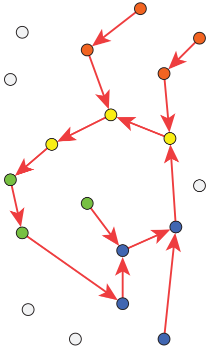
Simulating algorithms



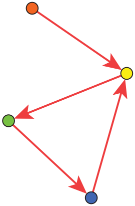
Each configuration of the simulated system must be *represented* by a set of configurations of the simulating system...

Simulating algorithms

Simulating system

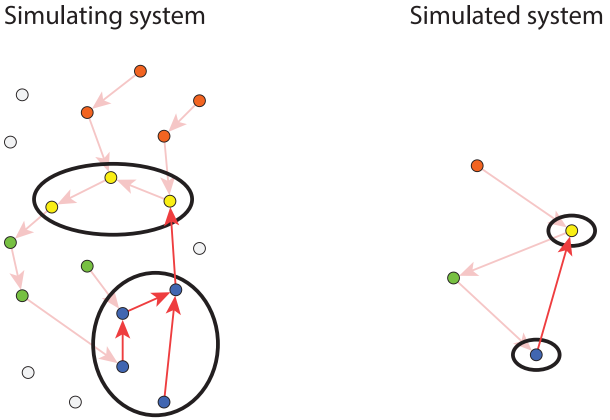


Simulated system



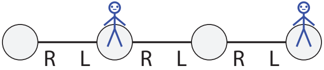
...In such a way that the simulating algorithm "respects" this correspondence.

Simulating algorithms



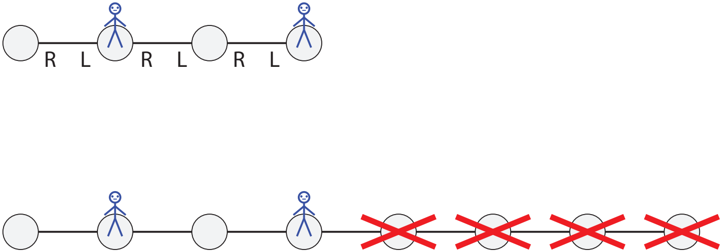
...In such a way that the simulating algorithm "respects" this correspondence.

Examples: simulations between paths and rings

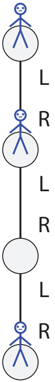


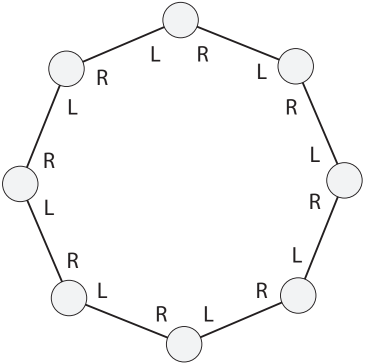
Any algorithm for an oriented n-path can be simulated on an unoriented $2n$-path by the same number of robots.

Examples: simulations between paths and rings



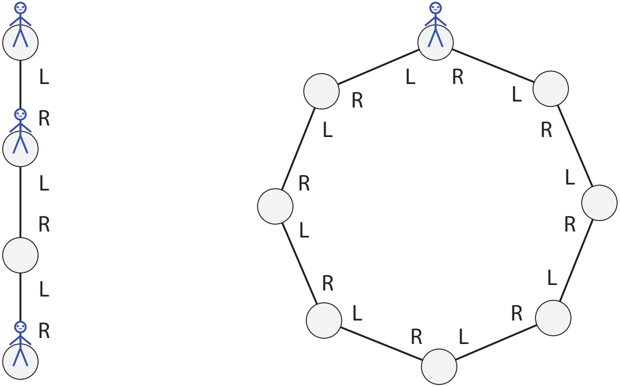
The simulation takes place on one half of the path, while the other half remains empty and defines an implicit orientation.





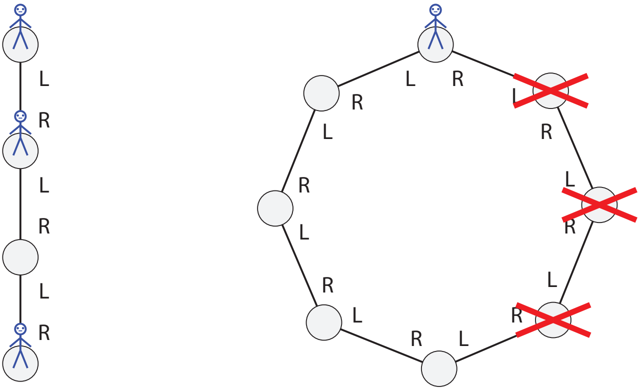
Any algorithm for an oriented n-path can be simulated on an oriented ring of size $2n$ by adding one robot.

Examples: simulations between paths and rings



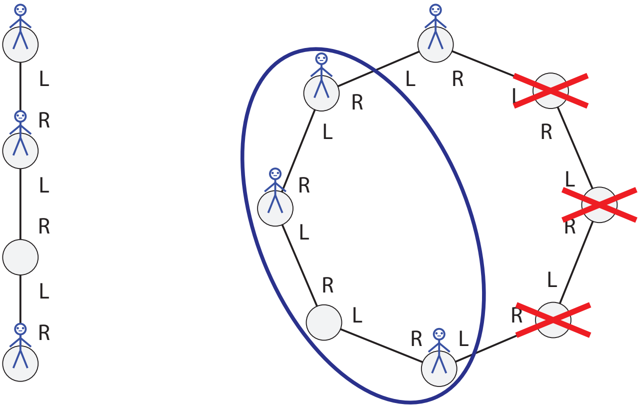
One half of the ring consists of a stationary *pivot robot* followed by $n - 1$ empty vertices.

Examples: simulations between paths and rings



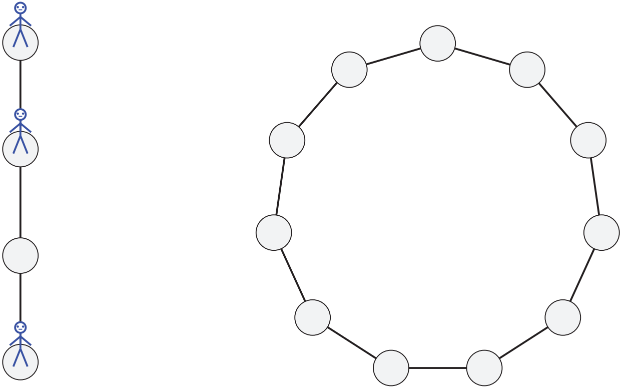
One half of the ring consists of a stationary *pivot robot* followed by $n - 1$ empty vertices.

Examples: simulations between paths and rings



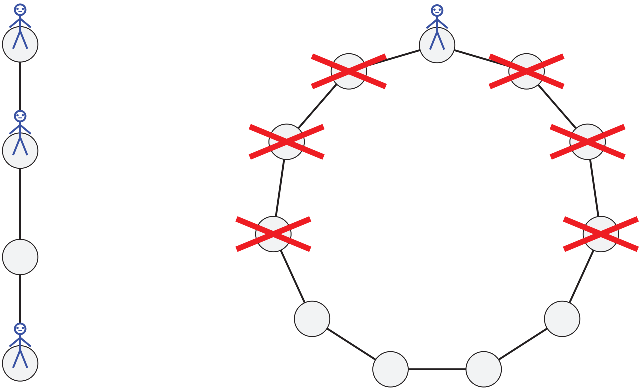
The simulation takes place on the remaining vertices, which are uniquely identified.

Examples: simulations between paths and rings



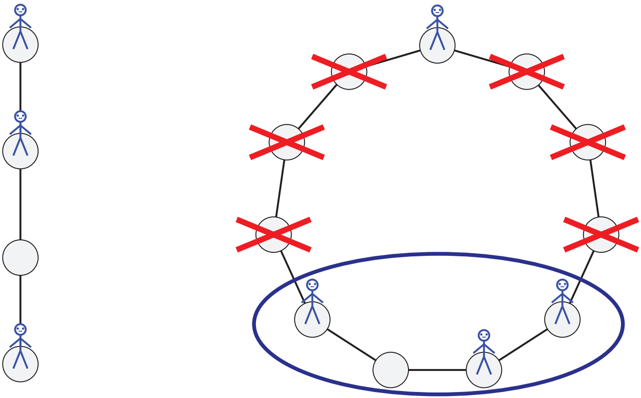
Any algorithm for an unoriented n-path can be simulated on a larger unoriented ring of size $3n - 1$ by adding one robot.

Examples: simulations between paths and rings



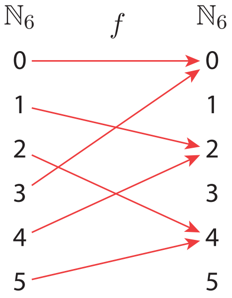
Place a static *pivot robot* surrounded on both sides by $n - 1$ empty vertices.

Examples: simulations between paths and rings



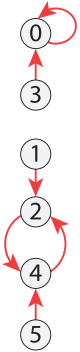
The simulation takes place on the remaining *n* vertices, which are uniquely determined.

Computing functions



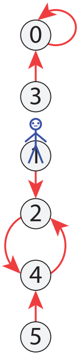
What does it mean to *compute* a function $f : \mathbb{N}_k \to \mathbb{N}_k$?

Computing functions



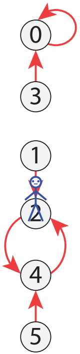
Interpret the function as a graph on $\mathbb{N}_k$...

Computing functions



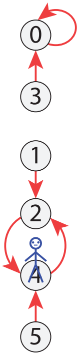
...Representing an algorithm for one robot.

Computing functions



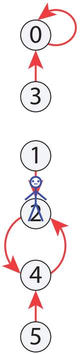
...Representing an algorithm for one robot.

Computing functions



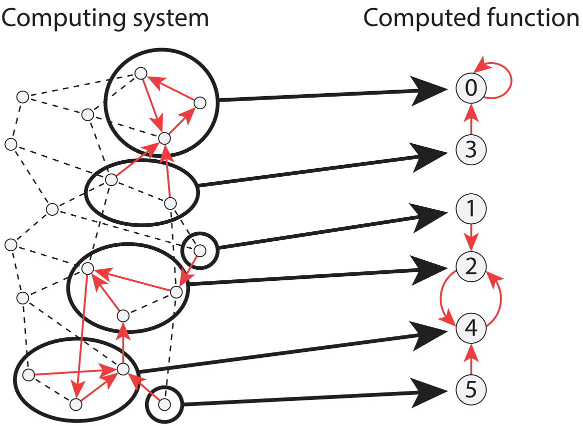
...Representing an algorithm for one robot.

Computing functions



...Representing an algorithm for one robot.

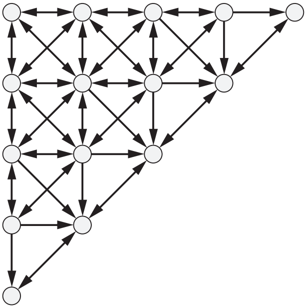
Computing functions



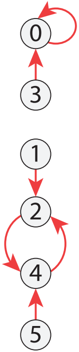
Computing the function means to simulate that algorithm.

Computing functions

Computing system
(oriented path, 2 robots)



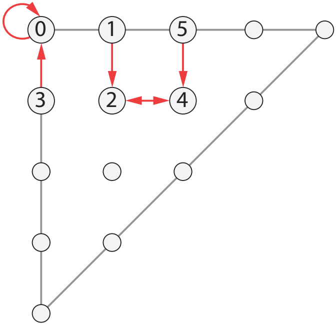
Computed function



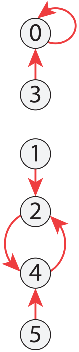
The computing system could be a long-enough oriented path with 2 robots.

Computing functions

Computing system
(oriented path, 2 robots)

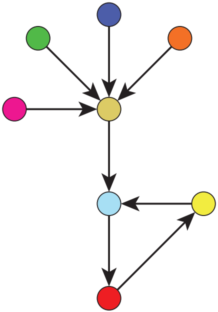


Computed function



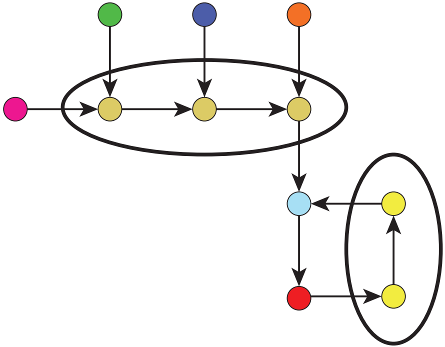
Indeed, the function's graph can be *drawn* on a grid...

Computing functions



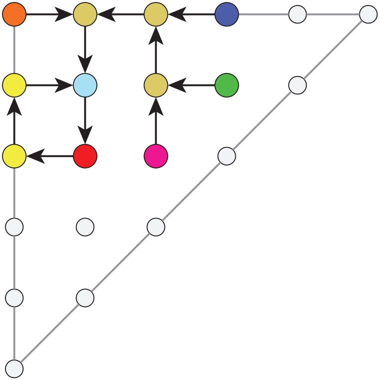
...Provided that we reduce the maximum degree to 3 and we make the length of every cycle even, by introducing extra vertices.

Computing functions



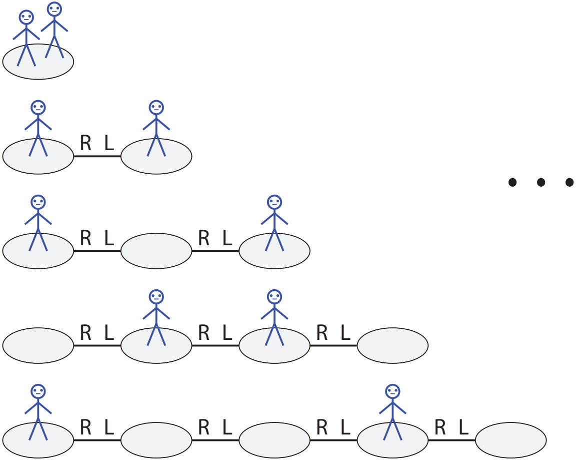
...Provided that we reduce the maximum degree to 3 and we make the length of every cycle even, by introducing extra vertices.

Computing functions



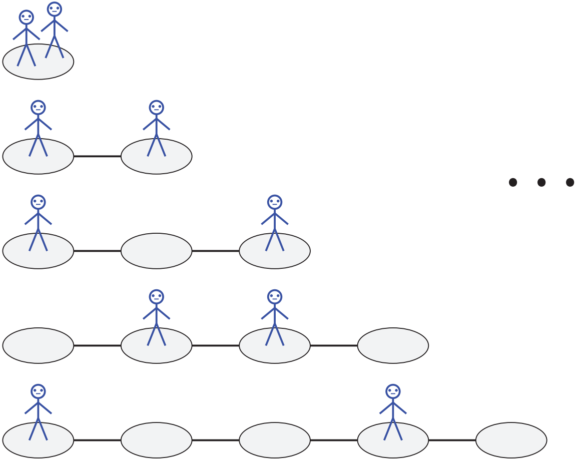
...Provided that we reduce the maximum degree to 3 and we make the length of every cycle even, by introducing extra vertices.

Universality



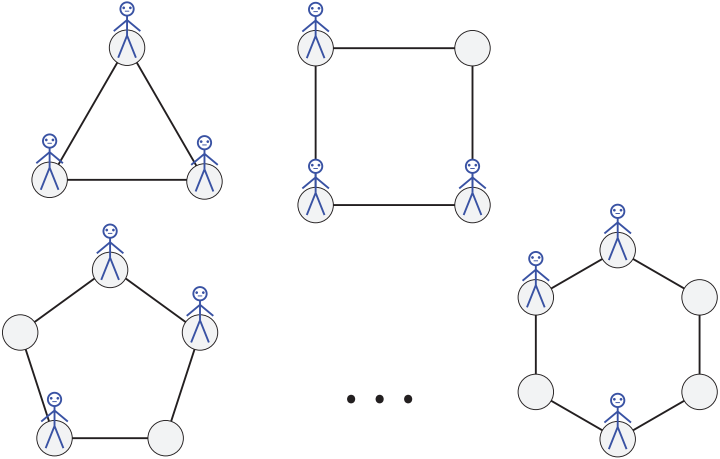
Therefore, the set of all oriented paths with 2 robots is *universal*, in that it can compute all (finite) functions.

Universality

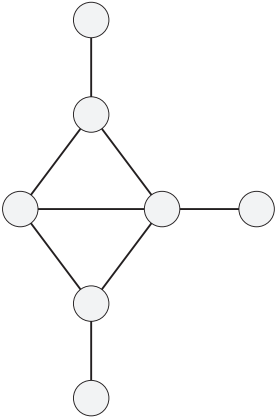


Since unoriented paths can simulate oriented paths, also the set of all unoriented paths with 2 robots is universal.

Universality

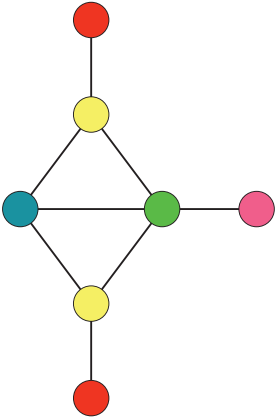


Similarly, the set of all (oriented or unoriented) rings with 3 robots is universal.



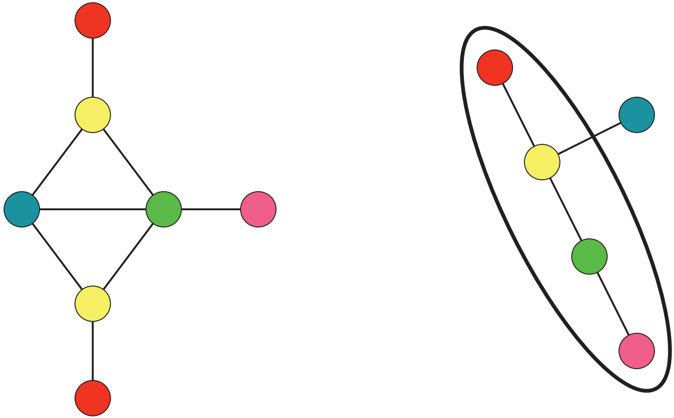
Suppose that a network's quotient graph contains a sub-path of length, say, 4.

Universality



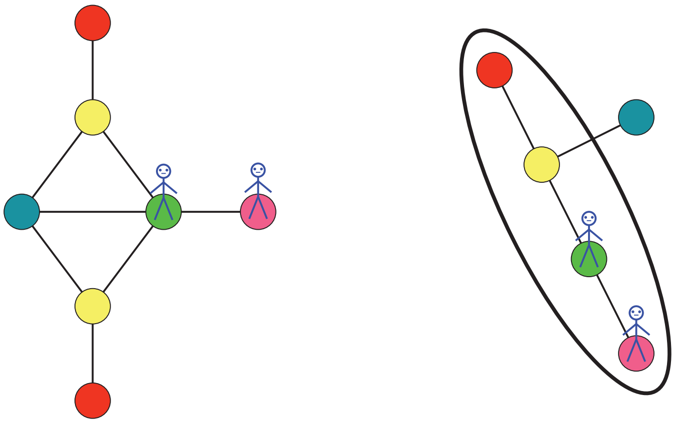
Suppose that a network's quotient graph contains a sub-path of length, say, 4.

Universality



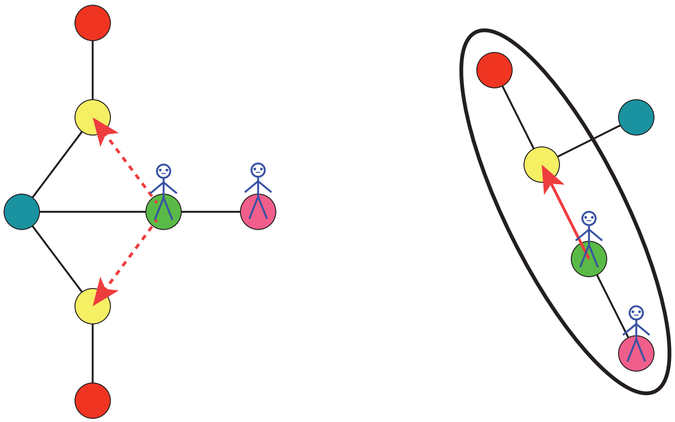
Suppose that a network's quotient graph contains a sub-path of length, say, 4.

Universality



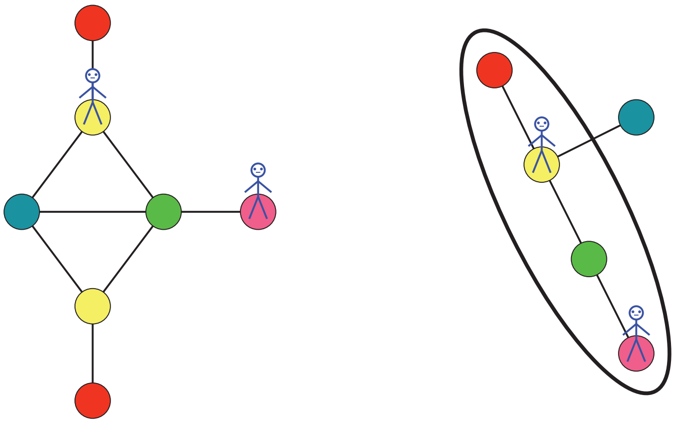
Then on the network we can simulate a path of length 4 (even if robots make non-deterministic moves).

Universality

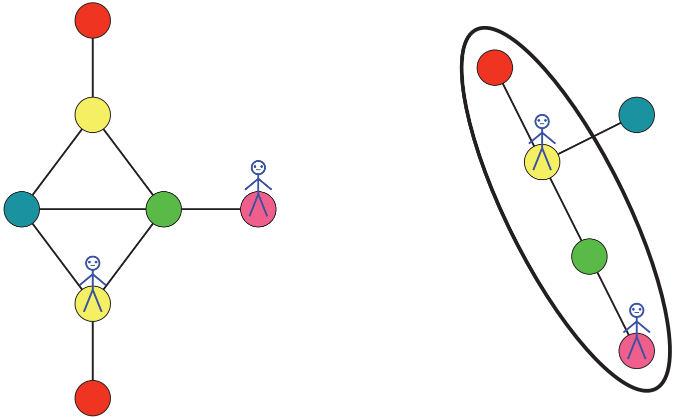


Then on the network we can simulate a path of length 4 (even if robots make non-deterministic moves).

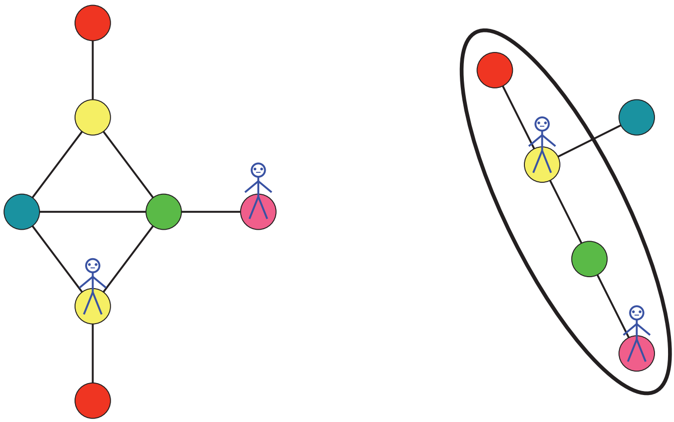
Universality



Then on the network we can simulate a path of length 4 (even if robots make non-deterministic moves).

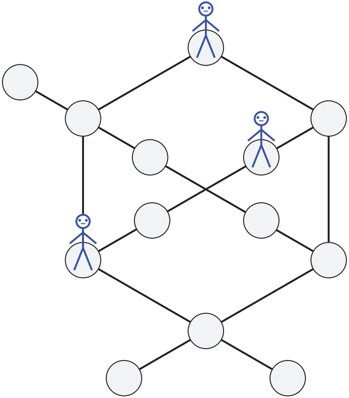


Then on the network we can simulate a path of length 4 (even if robots make non-deterministic moves).



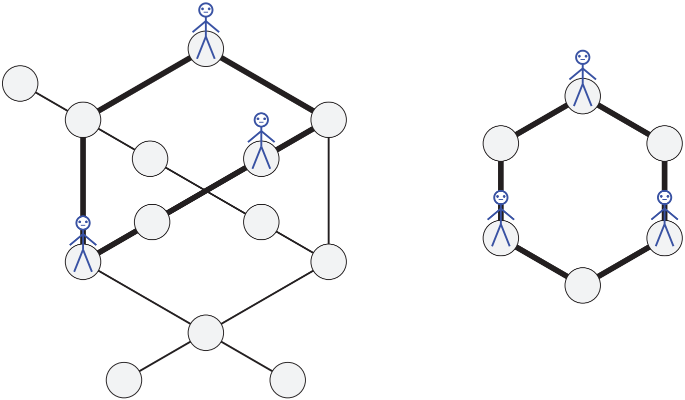
So, any set of networks with 2 robots whose quotient graphs contain arbitrarily long sub-paths is universal.

Universality



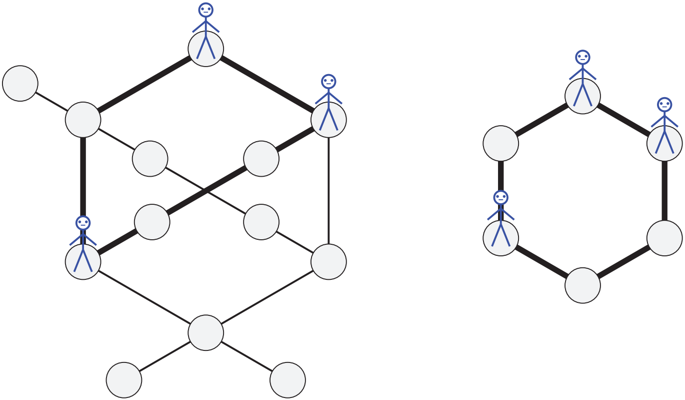
Suppose now that a network's *girth* is, say, 6.

Universality



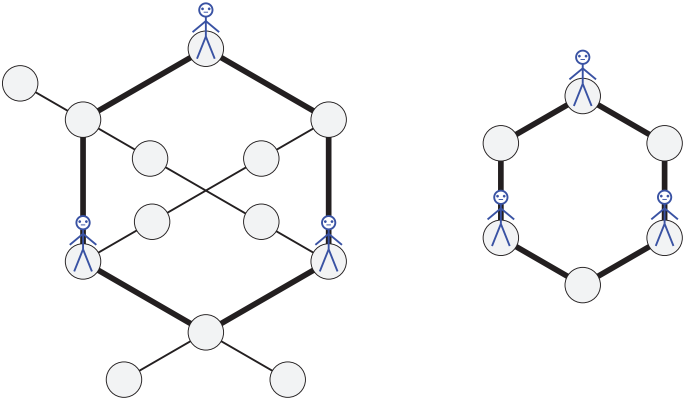
Then we can simulate a ring of size 6 on this network (even if robots make non-deterministic moves).

Universality



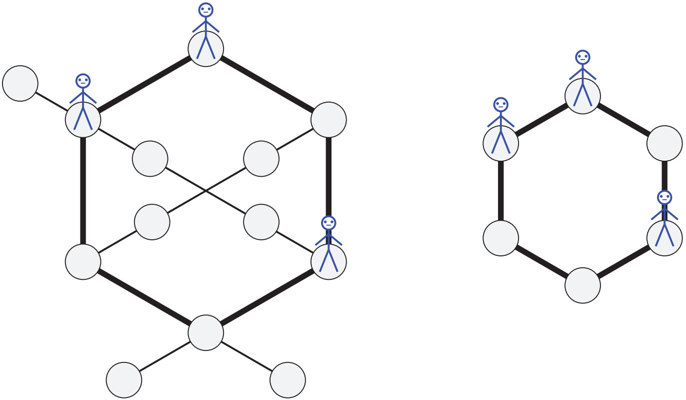
Then we can simulate a ring of size 6 on this network (even if robots make non-deterministic moves).

Universality

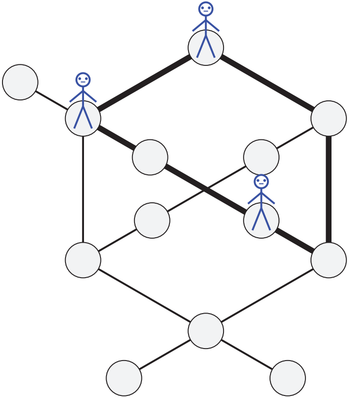


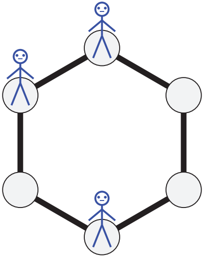
Then we can simulate a ring of size 6 on this network (even if robots make non-deterministic moves).

Universality



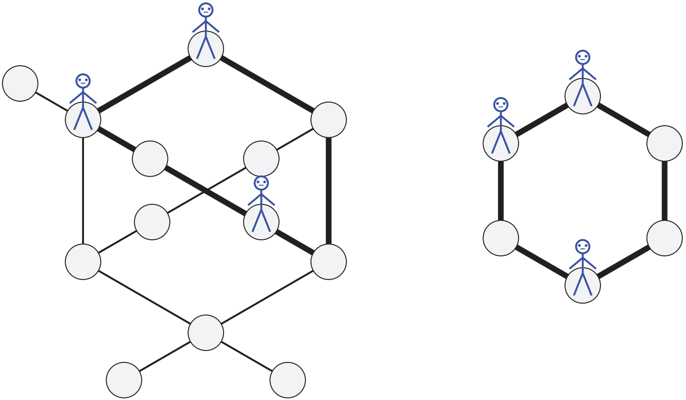
Then we can simulate a ring of size 6 on this network (even if robots make non-deterministic moves).





Then we can simulate a ring of size 6 on this network (even if robots make non-deterministic moves).

Universality



So, any set of networks of arbitrarily large (finite) girths and 3 robots is universal.

Universality

Theorem

Any set of networks with 2 robots whose quotient graphs contain unboundedly long sub-paths is universal.

Theorem

Any set of networks with 3 robots and unboundedly large (finite) girths is universal.

Conjecture

A set of anonymous networks with 3 robots is universal if and only if they have unboundedly large (finite) girths or their quotient graphs contain unboundedly long sub-paths.

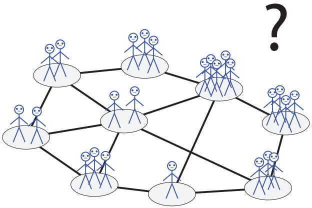
Minimizing network sizes

$$f\colon \mathbb{N}_{24} \to \mathbb{N}_{24}$$

Suppose we wanted to compute *all* the functions on a given set, say, $\mathbb{N}_{24}$.

Minimizing network sizes

$$f : \mathbb{N}_{24} \to \mathbb{N}_{24}$$



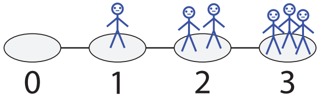
What is the *smallest* network on which we can compute all of them (possibly using a large number of robots)?

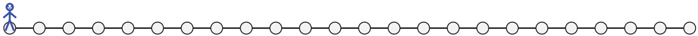
Minimizing network sizes

$$24 = 4!$$

0 1 2 3
0 1 3 2
1 0 3 2
1 3 0 2
1 3 2 0
3 1 2 0
...

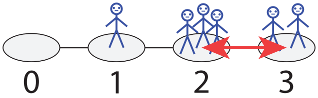
Recall that the 24 permutations of 4 objects can be ordered in such a way that two consecutive permutations differ by a transposition of two adjacent objects.

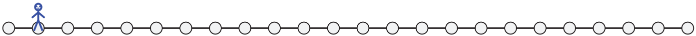




So, by putting a different number of robots on each vertex of a 4-path, we can encode the position of a robot on a 24-path...

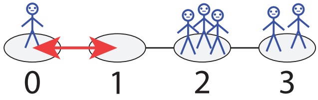
Minimizing network sizes

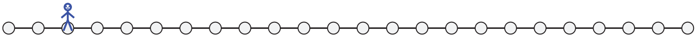




...And simulate its movements by properly exchanging the robots occupying two adjacent vertices.

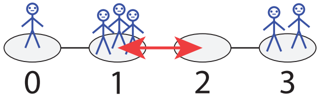
Minimizing network sizes

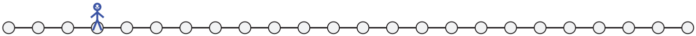




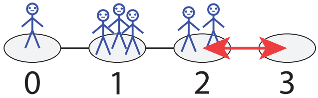
...And simulate its movements by properly exchanging the robots occupying two adjacent vertices.

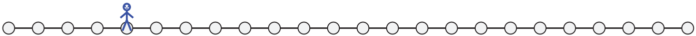
Minimizing network sizes





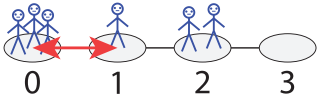
...And simulate its movements by properly exchanging the robots occupying two adjacent vertices.

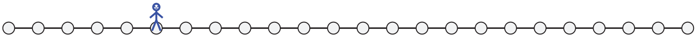




...And simulate its movements by properly exchanging the robots occupying two adjacent vertices.

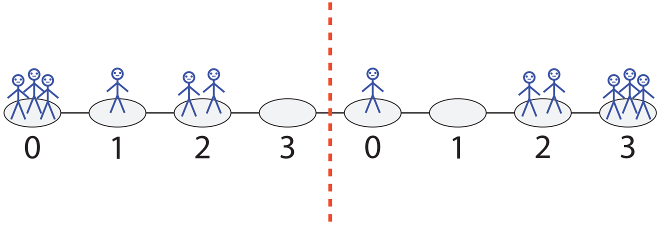
Minimizing network sizes

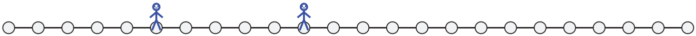




...And simulate its movements by properly exchanging the robots occupying two adjacent vertices.

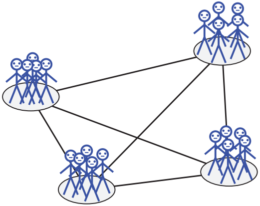
Minimizing network sizes



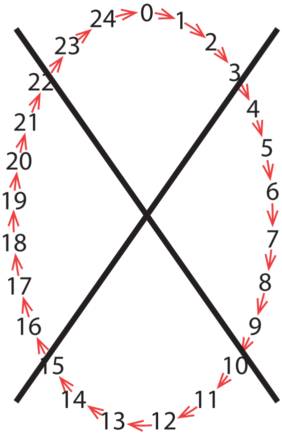


With an 8-path, we can encode the positions of 2 robots on a 24-path, and hence compute any function $f\colon \mathbb{N}_{24} \to \mathbb{N}_{24}$.

Minimizing network sizes

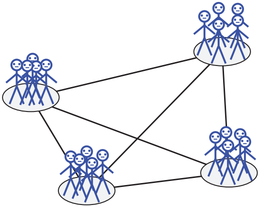


$f : \mathbb{N}_{25} \rightarrow \mathbb{N}_{25}$

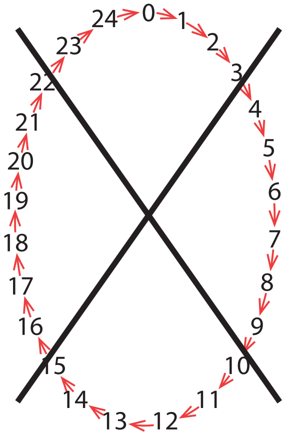


On the other hand, on a network of size 4 we cannot compute a function whose graph is a cycle of length 25.

Minimizing network sizes

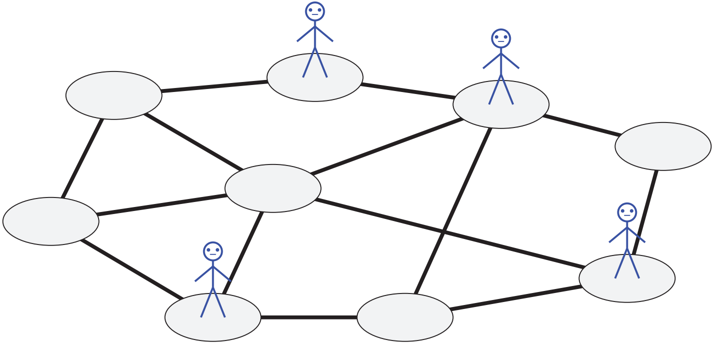


$$f : \mathbb{N}_{25} \to \mathbb{N}_{25}$$



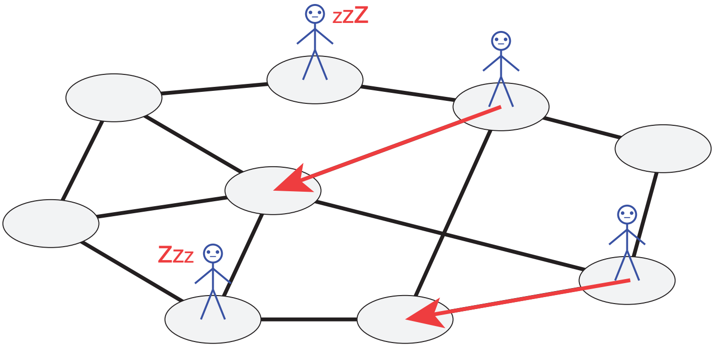
Therefore, we can determine the optimal size up to a factor of 2 (such is the ratio of our upper and lower bounds).

Further work: semi-synchronous robots



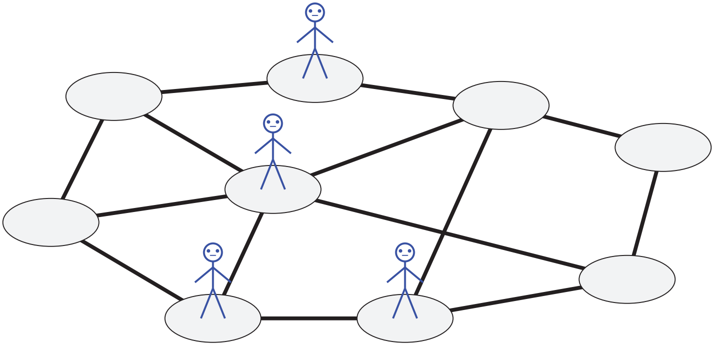
Our main results hold (with some variations) also if robots are *semi-synchronous*, i.e., they may unpredictably skip some turns.

Further work: semi-synchronous robots



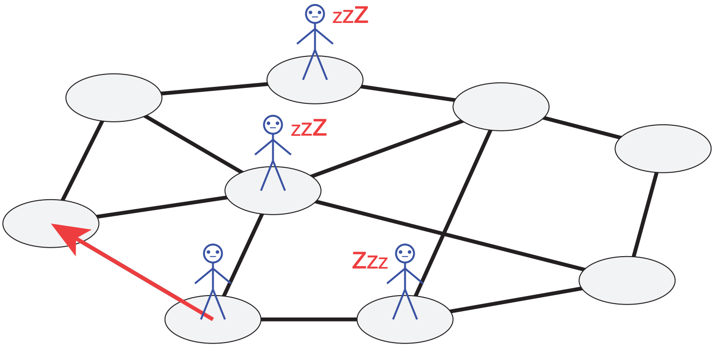
Our main results hold (with some variations) also if robots are *semi-synchronous*, i.e., they may unpredictably skip some turns.

Further work: semi-synchronous robots



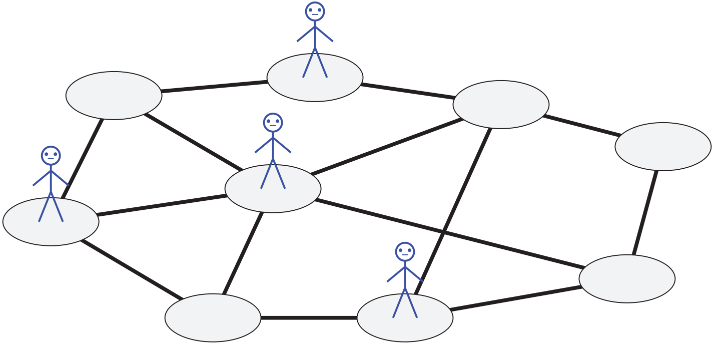
Our main results hold (with some variations) also if robots are *semi-synchronous*, i.e., they may unpredictably skip some turns.

Further work: semi-synchronous robots



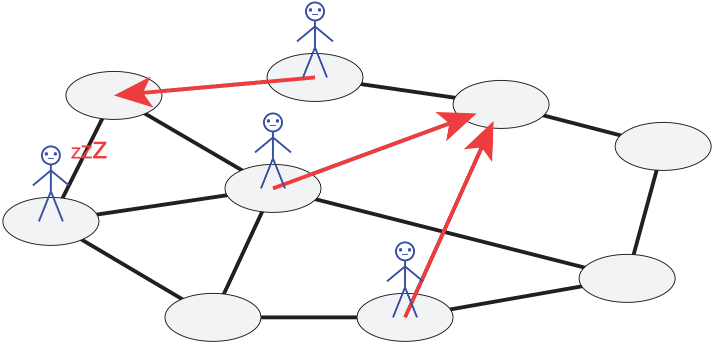
Our main results hold (with some variations) also if robots are *semi-synchronous*, i.e., they may unpredictably skip some turns.

Further work: semi-synchronous robots



Our main results hold (with some variations) also if robots are *semi-synchronous*, i.e., they may unpredictably skip some turns.

Further work: semi-synchronous robots



We can compare the “power” of the two models by comparing the functions that can be computed in each of them.