A Distributed Biconnectivity Check

Mazda Ahmadi and Peter Stone

Department of Computer Sciences,
The University of Texas at Austin,
{mazda,pstone}@cs.utexas.edu
http://www.cs.utexas.edu/~{mazda,pstone}

Summary. For many distributed autonomous robotic systems, it is important to maintain communication connectivity among the robots. That is, each robot must be able to communicate with each other robot, perhaps through a series of other robots. Ideally, this property should be robust to the removal of any single robot from the system. In this work, we define a property of a team's communication graph that ensures this property, called biconnectivity. We present a distributed algorithm to check if a team of robots is biconnected, prove its correctness, and analyze it theoretically.

1.1 Introduction

Many applications of distributed autonomous robotic systems can benefit from, or even may require, the team of robots staying within communication connectivity. For example, consider the problem of multirobot surveillance [1, 2], in which a team of robots must collaboratively patrol a given area. If any two robots can directly communicate at all times, the robots can coordinate for efficient behavior. This condition holds trivially in environments that are smaller than the robots' communication range. However in larger environments, the robots must actively maintain physical locations such that any two robots can communicate — possibly through a series of other robots. Otherwise, the robots may lose track of each others' activities and become miscoordinated. Furthermore, since robots are relatively unreliable and/or may need to change tasks (for example if a robot is suddenly called by a human user to perform some other task), in a stable multirobot surveillance system, if one of the robots leaves or crashes, the rest should still be able to communicate. Some examples of other tasks that could benefit from any pair of robots being able to communicate with each other, are space and underwater exploration, search and rescue, and cleaning robots.

We say that robot R_1 is *connected* to robot R_2 if there is a series of robots, each within communication range of the previous, which can pass a message from R_1 to R_2. In order for the team to stay connected, it must be the case that every robot is connected to each other robot either directly or via *two* distinct paths that don't share any robots in common. We call this property *biconnectivity*: the removal of any one robot from the system does not disconnect the remaining robots from each other.

In previous work, we developed algorithms for multirobot surveillance under the assumption that each pair of robots could communicate directly [2]. This communication assumption enabled the robots to negotiate to achieve an efficient task division,

but it constrained us to small environments. This work is the first attempt to extend these algorithms to larger environments.

To the best of our knowledge, the problem of enabling robots to remain connected in the face of robot failures has not been explored before. Typical related work in graph theory is on algorithms to find a biconnected component in a graph with optimal time complexity (e.g. [3]), in dynamic graphs (e.g. [4]), or in a restricted subclass of all graphs (e.g. [5]). In all these cases, the algorithms are either centralized, or if distributed, each node has full knowledge of the whole graph. Some work in distributed computing is closer in spirit to our work, however a main difference between their problem statement and ours is that in distributed computing (e.g. [6, 7]), any node can send a message to any other node. That is, the nodes are not restricted to send messages only through existing edges of the graph.

We tackle this problem by dividing it into three main steps: (1) Checking whether a team of robots is *currently* biconnected, (2) Maintaining biconnectivity should a robot be removed from (or added to) the team, and (3) Constructing a biconnected multi-robot structure from scratch. To be applicable for teams of autonomous robots, all algorithms must be fully distributed.

In this paper we focus on fully achieving and analyzing Step 1. Steps 2 and 3 remain as future work. Note that it is possible to achieve steps 2 and 3, even if inelegantly, by having the robots move back to a base and disperse from there whenever they find that they are no longer biconnected.

For the purposes of this paper, we assume that robots have constant and identical communication ranges. This assumption applies in the case of homogeneous robot teams (or at least teams with homogeneous transmitters) such that the range is not dependent on a robot's battery level. This assumption allows us to assume the connection graph among robots is undirected: if robot A can send a message to robot B, then the reverse is also true. Extension of this work to the case where robots have heterogeneous communication capabilities is also a part of our future work plans.

After the introduction, in Section 1.2 graph theory background and assumptions about the investigated multirobot systems is presented. Section 1.3 presents distributed algorithms to detect if the robots are biconnected. Finally Section 1.4 concludes the paper.

1.2 Preliminaries

We first provide some graph definitions and theorems which will be used later in the paper. For basic graph definitions, such as vertex, edge, neighbor, path and loop please see [8]. Later in the section, definitions and assumptions which are specific to our multirobot system will be presented.

Definition 1. Internally vertex-disjoint paths. *Two paths between v_1 are v_2 are internally vertex-disjoint if they have no vertices in common except v_1 and v_2.*

Definition 2. Biconnected graph. *If in graph G, after removing any vertex, it is possible to find a path from any vertex to any other one, the graph G is said to be biconnected.*

Definition 3. Doubly connected vertices. *In graph G, we say vertex v_1 and v_2 are doubly-connected iff there are two or more internally vertex-disjoint paths between v_1 and v_2.*

Lemma 1. *Undirected graph $G(V, E)$ is biconnected if and only if any two vertices $v_1, v_2 \in V$ are doubly-connected.*
Proof. It is a special case of Menger's Theorem (See Theorem 3.3.1 of [8]).

Note that in undirected graphs one vertex being doubly-connected to all other vertices is not a sufficient condition for the graph to be biconnected. For an example see Figure 1.1 where v is doubly-connected to all other vertices, but removing v makes the graph disconnected. In the following theorem we show that in an undirected graph if there are two vertices that are doubly-connected to all other vertices, then the graph is biconnected.

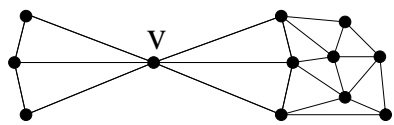

Fig. 1.1. V is doubly-connected to all other vertices, but the graph is not biconnected.

Theorem 1. *Undirected graph $G(V, E)$ is biconnected if and only if there exists two distinct vertices $v_1, v_2 \in V$ such that both v_1 and v_2 are doubly-connected to any other vertex in V.*
Proof. If Graph $G(V, E)$ is biconnected, then by Lemma 1 any two vertices are doubly-connected to all other vertices.

It remains to prove that if there exists $v_1, v_2 \in V$ ($v1 \neq v2$) such that they are doubly-connected to all other vertices in V, then $G(V, E)$ is biconnected. Assume $v_i \in V$ is removed from V. We must show that the graph remains connected. If $v_i = v_1$, then since any other vertices $v_j \in V$ were doubly-connected to v_2, the graph remains connected. Similarly for $v_i = v_2$. Now assume $v_i \neq v_1, v_2$. Since every other vertex $v_j \in V$ was doubly-connected to v_1, and v_i is in at most one of the two paths between v_j and v_1, after removal of v_i, v_j remains connected to v_1. Thus the graph remains connected after removal of any vertex: it is biconnected.

We look at our multirobot system as a graph, such that its vertices are robots and edge $(v_1 v_2)$ exists in the graph iff the robot corresponding to v_1 can communicate directly to the robot corresponding to v_2 (i.e. v_1 and v_2 are in communication range of each other). A formal definition of robot graph follows.
Definition 4. Robot graph *$RG(V, E)$ is a graph, where its vertices (V) are the robots and $(v_1, v_2) \in E$ iff corresponding robots to v_1 is a neighbor of corresponding robot to v_2. Size of V (i.e. number of robots in the multirobot team) is called n in this paper.*

Assumption 1 *Robots are aware of the maximum number of robots in the system, which can be considerably higher than the actual number of robots. The maximum number is called N throughout the paper.*

Assumption 2 *Robots have identical communication capabilities.*

As a result of the above assumption, the neighbor property is symmetric, and the robot graph is undirected.
Definition 5. Connected. *We say robot R_1 and robot R_2 are connected, when in the corresponding robot graph, there is a path between R_1 and R_2.*

Assumption 3 *Each robot has a unique and ordered ID. For robot X its ID is called X.id.*

Next, the definition and assumption regarding the communication between robots is provided.

Definition 6. Message, stamped message. *For our purposes, a* message*, which is used for robot communication, is a string in format* $(T, (S))$*, where* T *indicates the type of the message, and* S *is a list of robot* stamps*. Message* $(T, (S))$ *is said to be* stamped *by robot R iff* $R.id \in (S)$*. Robot* R *stamps message* $(T, (S))$ *by generating new message* $(T, (S, R.id))$.

Assumption 4 *When called for by the protocol, robots relay messages to one another. Robots start processing received messages, as soon as they get them. The maximum period from the time that robot* R_1 *receives message* X*, until its neighbor robot* R_2 *receives the processed (possibly stamped) version of message* X *from* R_1 *is c seconds.*

1.3 Algorithms to Check Biconnectivity

As mentioned in Section 1.1, checking for biconnectivity is the first step towards the overall goal of achieving and maintaining a biconnected multirobot structure. It is an important step, because the robots must be able to detect if they are not biconnected, so that they can take remedial actions to restore *biconnectivity* before they lose *connectivity*. The remedial actions could be as simple as all robots moving back to a base and dispersing from there.

Note that the biconnected property is a global property of the multirobot system: robots cannot determine whether or not it holds from purely local information. For example see Figure 1.1, where the graph is not biconnected, and the robots associated with the nodes on the right side of the graph need global information about the nodes on the left side to know that the whole structure is not biconnected.

In our approach, each robot R, maintains two lists:

- CR_R *(connected robots)*: the list of robots that are connected to R.
- DCR_R *(doubly-connected robots)*: the robots doubly-connected to R.

Each robot R first fills the CR_R list, then using that, the DCR_R list is computed. Finally with the help of the DCR_R list, it detects if the robot graph is biconnected.

In the rest of this section, we first provide an algorithm (CR-FILL) for filling CR_R in Section 1.3.1, then another algorithm (DCR-FILL) is presented in Section 1.3.2 which fills the DCR_R lists with the help of the already computed CR_R lists. Afterwards in Section 1.3.3, an algorithm which checks the biconnectivity with the help of the computed DCR_R lists is provided. All these algorithms are distributed and each robot runs them independently. Finally an analysis of the presented algorithms is provided in Section 1.3.4.

1.3.1 CR-FILL

In this subsection, we provide an algorithm for filling the CR_R list. That is for robot R, it finds the robots that are connected to it.

The basic idea is for the robots to stamp and pass messages in the system. In this way, if there is a path of $r_0 \rightarrow r_1 \rightarrow r_2 \rightarrow R$, between r_0 and R, robot R will receive a message that is stamped by r_0, r_1 and r_2. Thus it will know that it is connected to those robots, and will add them to the CR_R list.

Two helper algorithms must run continuously on all the robots to help the CR-FILL algorithm. The first helper algorithm dictates how messages should be passed around. In the second, if robot r has not sent a message for a while and another robot is running a CR-FILL algorithm and needs a stamped message initiated from r, it will send a stamped message. After introducing these two helper algorithms, the CR-FILL algorithm itself will be presented.

Using the first helper algorithm, all robots continually stamp and pass biconnected type messages that they receive. Any robot r, which receives $(\text{“}CR\text{”}, (S))$, checks the content of S, and if $r.id \notin S$ it stamps the message and send it out. That is, it sends message $(\text{“}CR\text{”}, (S, r.id))$. If $r.id \in S$, it does not send any message because stamping and sending it would lead to a duplicate ID in $(S, R.id)$. For an overview of this algorithm see Algorithm 1.

Algorithm 1 Message passing algorithm which robots continually run

1: **upon** receiving a message of form $(\text{“}CR\text{”}, (S))$ **do**
2: **if** $R.id \notin (S)$ **then** robot R broadcast message $(\text{“}CR\text{”}, (S, R.id))$.
3: **end upon**

Any robot, upon receiving a message of format $(\text{“}CR\text{”}, S)$, if it has not sent out a $(\text{“}CR\text{”}, (R.id))$ message in the last Nc seconds (Recall from Assumption 1, that N is the maximum number of robots in the system), it sends out message $(\text{“}CR\text{”}, (R.id))$ (see Algorithm 2).

Algorithm 2 The condition for initiating a “CR” message.

1: **upon** receiving a message of form $(\text{“}CR\text{”}, (S))$ **do**
2: **if** has not sent out a $(\text{“}CR\text{”}, (R.id))$ message in the last Nc seconds **then**
3: broadcast message $(\text{“}CR\text{”}, (R.id))$
4: **end if**
5: **end upon**

When calling the main CR-FILL algorithm, robot R starts by initializing the CR_R list to empty. Afterwards each time it receives message $(\text{“}CR\text{”}, (S))$, it adds all the IDs in S to CR_R. While still adding IDs to the CR_R, at time Nc, robot R sends out a stamped message $(\text{“}CR\text{”}, (R.id))$. The pseudocode of this algorithm is available in Algorithm 3.

Algorithm 3 Pseudocode for the CR-FILL algorithm for robot R

1: Time 0 (start of the algorithm): initialize the CR_R to empty.
2: Time Nc: broadcast message $(\text{“}CR\text{”}, (R.id))$
3: **if** message of form $(\text{“}CR\text{”}, (S))$ is received **then** add IDs in (S) to CR_R.

Since the length of the longest path in the graph is less than N, the maximum time for a message to reach robot r_2 from r_1 is Nc seconds. We now show any robot r that is connected to R will be added to CR_R within $3Nc$ seconds. Any robot r that is connected to R, receives the stamped message from robot R within $2Nc$ seconds

(note that the first message is sent at time Nc). If it has sent a message in the last Nc seconds, robot R has gotten that message. Otherwise, it will send out a message which will be heard by R in at most Nc seconds. Thus after $3Nc$ seconds, CR_R represents the correct list of robots that are connected to R. This analysis is based on the assumption that the robots do not change connectivity in the $3Nc$ seconds that CR-FILL runs. Note that after $3Nc$ seconds, no message is left in the system. Because message $(\text{“}CR\text{”}, (S))$ can only survive if it is received by robots such that their ID is not in (S), $2Nc$ seconds after sending the first message any remaining message in the system has been stamped by all robots, and it cannot survive any longer. Also note that c is ideally on the order of milliseconds, though in practice it may be difficult to guarantee such small bounded transmission times. In such cases, the algorithms as is may become impractical for large teams of fast-moving (so that connectivity changes quickly) robots.

1.3.2 DCR-FILL

In this subsection, DCR-FILL, an algorithm to fill the DCR lists is presented. It is assumed the message passing algorithm (Algorithm 1) is running continually by all robots.

The basic idea for filling DCR_R for robot R is to find the robots that are in a common loop with R. When the robot graph is undirected (Assumption 2), there is a loop including both R and R' iff two internally vertex-disjoint paths (Definition 1) exist between R and R'. In this case, R and R' are doubly-connected (Definition 3). According to Algorithm 1, robots pass stamped messages around. When robot R receives a message that has been stamped by itself (i.e. R), it knows the robots that have stamped the message after the R stamps are in a common loop with R, and should be added to DCR_R.

Robot (r) starts by broadcasting message $(\text{“}DCR\text{”}, (r.id))$, which will be heard by all of its neighbor robots. Upon receiving message $(\text{“}DCR\text{”}, (S))$, if this is the first time to receive a “DCR” message, it resets DCR_r to empty (initializing), afterwards it checks the content of (S). If its own ID *is* in the stamp part of the message (S), it can represent (S) as $(S_1, r.id, S_2)$. If S_2 includes more than one vertex, it means that there is a loop and the robot adds all the IDs in S_2 to DCR_r. If S_2 includes only one vertex, it means that the robot has got back a message from a robot that it has just sent a message to, and should be ignored. Algorithm 4 presents the pseudocode of this algorithm.

We now show that for robot R, the DCR-FILL algorithm sets the correct DCR_R list within nc seconds.

Theorem 2. *For any robot R, the* DCR-FILL *algorithm finds the full list of doubly-connected robots (DCR_R) within nc seconds.*

Proof. Consider the robot graph $RG(V, E)$ for the robots, and also $v_1 \in V$ represents robot R. To prove this theorem, we need to show for any vertex $v_2 \in V$, if v_1 is doubly-connected to v_2, then $v_2 \in DCR_{v_1}$, and if ($v_2 \in DCR_{v_1}$) then v_2 and v_1 are doubly-connected. Also we need to show DCR-FILL *is completed (i.e. there is no message in the system) after nc seconds.*

We start with the first part, and assume v_1 and v_2 are doubly-connected, so there are two internally vertex-disjoint paths (a loop) between them. The starting message

Algorithm 4 Pseudocode for DCR-FILL algorithm

```
1: Time 0: robot R broadcasts ("DCR", (R.id)).
2: upon receiving a message of form ("DCR", (S)) do
3:    if this is the first time to receive a "DCR" message then
4:       reset DCR_R to empty
5:    end if
6:    if R.id ∈ (S) then
7:       split (S) to (S_1, R.id, S_2)
8:       if size(S_2) > 1 then add the IDs in S_2 to DCR_R
9:    end if
10: end upon
```

from v_1 will go through the loop, and vertex v_1 will get back the message that it stamped earlier, which is now also stamped by v_2. Thus, v_2 will be added to DCR_{v_1}.

Now we have to prove the other part, assuming $v_2 \in DCR_{v_1}$. Based on the algorithm, the only way that v_2 is added to DCR_{v_1} is when v_1 receives a message ("DCR", $(S_i, v_1.id, S_j, v_2.id, S_k)$). Notice that based on the condition in the algorithm ($size(S_2) > 1$ (Algorithm 4), if both S_j and S_k are empty, the IDs will not be added to DCR_R, also recall that there is no duplicate IDs in messages, because no robot stamps a message that it has previously stamped. The two internally vertex-disjoint paths between v_1 and v_2 are $v_1 S_j v_2$ and $v_1 S_k v_2$. Thus v_1 and v_2 are doubly-connected.

Similar to the argument at the end of Section 1.3.1, after nc seconds no message remains in the system, and the algorithm terminates.

1.3.3 Biconnectivity Check

After running CR-FILL and DCR-FILL consecutively, the CR and DCR lists will be accurate. Notice that both algorithms for filling the CR and DCR lists finish within a known time limit. Thus the robots should wait $3Nc$ seconds, and afterwards CR and DCR lists will be accurate. For robot r if CR_r and DCR_r are equal, it means that r is doubly-connected to all the robots that it is connected to. By Theorem 1, we know if there are two robots r_1 and r_2 that are doubly-connected to all other robots, then the robot graph is biconnected. Also, we know by Lemma 1 that if there is a robot that is not doubly-connected to all other robots, the robot graph is not biconnected. Thus if the robot and one of its neighbors is doubly-connected to all other robots, the robot knows that the robot graph is biconnected. Also if the robot or one of its neighbors is not doubly-connected to all other robots, it will know that the robot graph is not biconnected.

The overview of the biconnectivity check algorithm is shown in Algorithm 5. The *initiator* robot (which can be any robot who wants to check biconnectivity) starts by sending a ("CHECK-REQUEST",()) message to its neighbors to ask them to check if they are doubly-connected to other robots or not. Upon receiving a ("CHECK-REQUEST",()) the other robots run biconnectivity check (unless they are already running it) as non-initiators (skipping line 3 of Algorithm 5). Note that multiple robots can run the biconnectivity check algorithm in parallel.

If the robot is doubly-connected to all other robots, it sends the message ("DC-TRUE", ()) to all its neighbor robots, and a ("DC-FALSE",()) message otherwise. If the robot is doubly-connected to all other robots and receives a ("DC-TRUE", ()) message, it knows that the robot graph is biconnected. Otherwise (if it is not biconnected to all other robots, or receives a ("DC-FALSE", ()) message) it knows that the robot graph is not biconnected. Since the initiator and its neighbors should run the biconnectivity check, the total time needed for the biconnectivity check to complete is $6Nc + 2c$ seconds.

Algorithm 5 Pseudocode for biconnectivity check algorithm. It returns *true* if the robot graph is biconnected, and *false* otherwise.

```
1: run CR-FILL and DCR-FILL in parallel, and wait 3Nc seconds for them to be finished.
2: if (initiator) then send message ("CHECK-REQUEST",())
3: if size(DCR_R) = size(CR_R) then
4:     send message ("DC-TRUE",())
5: else
6:     send message ("DC-FALSE",())
7:     return false;
8: end if
9: if a message of form ("DC-FALSE", ()) is received then return false;
10: if a message of form ("DC-TRUE", ()) is received then
11:     if size(DCR_R) = size(CR_R) then return true;
12: end if
```

1.3.4 Algorithms' Analysis

In this section we analyze both the time and communication complexity of the CR-FILL and DCR-FILL algorithms.

CR-FILL, DCR-FILL and biconnectivity check algorithms use $3Nc$, nc, and $6Nc+2c$ seconds to complete respectively.

For the analysis of the number of messages, we assume that the time needed for the message sent by a robot to reach its neighbor is constant (c). This assumption does not change the total number of messages, but possibly can change the maximum number of messages at any point in time. The worst case for the number of messages in the multirobot system happens when the robot graph is fully connected, that is every two robots are neighbors. For both DCR-FILL and CR-FILL algorithms, the maximum number of messages in the system is $n!$, because when robot R receives message $(T, (S))$, where S has size of i, the message only survives if robot R sends it to the robots that have not already stamped the message, and $n - i - 1$ such robots exist. Thus from the messages that have started from robot R, $(n-1)!$ can exist in the system, and since each of the n robots starts one message of its own, at any point in time, the maximum number of messages in the system for DCR-FILL or CR-FILL algorithm is $n!$.

Note that the times provided above are the worst case, and especially when there are many robots, the robot graph is most likely not fully connected. An example of a still densely connected robot graph with 35 robots is given in Figure 1.2, there in each time period, on average each robot deals with 1037 messages, which is a manageable number. But 1037 messages is still a lot to process in each time step.

Our analysis is based on the assumption that CR-FILL must generate special-purpose messages. When messages are being sent for other purposes, the stamps required can simply be appended to those, thus eliminating the need for *many* extra messages. Though if each message does not normally generate a broadcast responses, *some* extra messages may be needed in order to keep the time complexity the same. In principle, DCR-FILL only needs to be started by 2 robots, thus reducing the number of messages required by a factor of $\frac{2}{n}$. When those two robots have computed their DCR_R, they can let the others know if the robot graph is biconnected. The technical details of how to determine which two robots send the starting messages is beyond the scope of this paper. However in essence, it is similar to maintaing team leaders, which is a common practice in multirobot systems (e.g. [9]).

If CR-FILL uses existing messages and DCR-FILL is run by only 2 robots, the total number of messages is $2(n-1)!$ in the worse case, and for the graph of Figure 1.2, on average each robot deals with approximately 69 messages, which is an easily manageable number in most realistic scenarios.

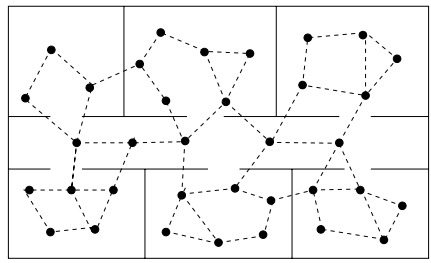

Fig. 1.2. An example of a common robot graph with 35 robots.

All the presented algorithms only store CR and DCR lists, which have a maximum size of n. Thus all algorithms use $O(n)$ memory space.

The time complexity discussed here is for each received message. That is, we assume the decisions are made when messages arrive. Both the CR-FILL and DCR-FILL algorithms have time complexity of $O(n)$ because they only traverse a list of IDs, which has size of at most n. The time complexity of biconnectivity check, which include both CR-FILL and DCR-FILL is of $O(2n)$.

1.4 Conclusion and Future Work

In this paper, we defined and argued the need for biconnected multirobot structures. A distributed algorithm for checking biconnectivity is presented, proven correct, and analyzed theoretically.

In future work, we aim to provide optimality bounds for the provided checking biconnected algorithms. The assumption that robots have identical communication capabilities should be relaxed, which will result in the robot graph being directed. Also, our algorithms for maintaining biconnectivity and constructing biconnected structure from scratch remains as future work.

Acknowledgments

We would like to thank Kurt Dresner and Nick Jong for their valuable comments on an earlier version of this paper. This research was supported in part by NSF CAREER award IIS-0237699 and ONR YIP award N00014-04-1-0545.

References

1. Parker, L.E.: Distributed algorithms for multi-robot observation of multiple moving targets. Autonomous Robots **12** (2002) 231–255

2. Ahmadi, M., Stone, P.: A multi-robot system for continuous area sweeping tasks. In: Proceedings of International Conference on Robotics and Automation (ICRA), to appear. (2006)
3. Tarjan, R., Vishkin, U.: Finding biconnected componemts and computing tree functions in logarithmic parallel time. In: 25th Annual Symposium on Foundations of Computer Science, 1984. (1984) 12–20
4. Westbrook, J., Tarjan, R.E.: Maintaining bridge-connected and biconnected components on-line. Algorithmica (Historical Archive) **7** (1992) 433–464
5. Galil, Z., Italiano, G.F.: Maintaining biconnected components of dynamic planar graphs. In: Proceedings of the 18th International Colloquium on Automata, Languages and Programming, London, UK, Springer-Verlag (1991) 339–350
6. Swaminathan, B., Goldman, K.J.: An incremental distributed algorithm for computing biconnected components (extended abstract). In: Proceedings of the 8th International Workshop on Distributed Algorithms, London, UK (1994)
7. Ahuja, M., Zhu, Y.: An efficient distributed algorithm for finding articulation points, bridges, and biconnected components in asynchronous networks. In: Proceedings of the Ninth Conference on Foundations of Software Technology and Theoretical Computer Science, London, UK, Springer-Verlag (1989) 99–108
8. Diestel, R.: Graph Theory. Springer, New York (1997)
9. *R. Alur et al.*: A framework and architecture for multirobot coordination. In: Seventh International Symposium on Experimental Robotics. (2001)