Journal of Artificial Intelligence Research 47 (2013) 649-695 Submitted 02/13; published 08/13

Protecting Privacy through Distributed Computation in Multi-agent Decision Making

Online Appendix 2: DFS Tree Generation Algorithm

Thomas Léauté THOMAS.LEAUTE@A3.EPFL.CH
Boi Faltings BOI.FALTINGS@EPFL.CH

Algorithm 1 Pseudo-tree generation algorithm for variable x

Require: a root variable
1: **if** x has at least one neighbor **then**
2: **if** x is the root **then**
3: $open_x \leftarrow$ all neighbors of x
4: Remove a random neighbor y_0 from $open_x$ and add it to $children_x$
5: Send a CHILD message to y_0
6: **loop**
7: Wait for an incoming message of type *type* from a neighbor y_i
8: **if** $open_x = \emptyset$ **then** // first time x is visited
9: $open_x \leftarrow$ all neighbors of x except y_i
10: $parent_x \leftarrow y_i$
11: **else if** *type* = CHILD **and** $y_i \in open_x$ **then**
12: Remove y_i from $open_x$ and add it to $pseudo\_children_x$
13: Send PSEUDO message to y_i
14: **next**
15: **else if** *type* = PSEUDO **then**
16: Remove y_i from $children_x$ and add it to $pseudo\_parents_x$
17: // Forward the CHILD message to the next *open* neighbor:
18: Choose a random $y_j \in open_x$
19: **if** there exists such a y_j **then**
20: Remove y_i from $open_x$ and add it to $children_x$
21: Send a CHILD message to y_j
22: **else**
23: **if** x is not the elected root **then** // backtrack
24: Send a CHILD message to $parent_x$
25: **return**

Given a root variable, a DFS-tree ordering of the decision variables rooted at that root can be produced using Algorithm 1, originally proposed by Cheung (1983). This algorithm performs a distributed, depth-first traversal of the constraint graph, during which each variable x maintains the following fields:

- $parent_x$ is x's parent in the pseudo-tree (if x is not the elected root);
- $children_x$ is the list of x's children;
- $pseudo\_parents_x$ is the list of x's pseudo-parents;
- $pseudo\_children_x$ is the list of x's pseudo-children;
- $open_x$ is the list of x's neighboring variables that remain to be visited.

Variable x heuristically chooses a neighbor y_0 as its first child (line 4) and sends it a CHILD message (line 5). The choice of the child is guided by a heuristic, which, in the case of the P-DPOP$^+$ algorithm, selects a child at random so as not to leak out private information.

When variable x receives a message from a neighbor y_i (line 7), if it is the first message received, then the sender y_i is identified as the parent of x (lines 8 to 10). Else, if it is a CHILD message received from an *open* neighbor, then y_i is identified as a pseudo-child, which variable x reveals to y_i by responding with a PSEUDO message (lines 11 to 16). The CHILD message is then recursively passed from *open* neighbor to *open* neighbor, until no *open* neighbor remains (lines 18 to 22).

When variable x has no more open neighbors, it initiates a traversal backtrack by returning a CHILD message to its parent (lines 23 to 24). The algorithm terminates when the root variable receives a CHILD message from a child and has no more *open* neighbors, which happens after the exchange of exactly $2n_e$ sequential messages, where n_e is the number of edges in the pseudo-tree: two CHILD messages down and up each tree-edge, plus one CHILD message up and one PSEUDO message down each back-edge.

Theorem 1. *Algorithm 1 guarantees full agent and topology privacy.*

Proof. After running the algorithm, each variable's knowledge of the chosen pseudo-tree is limited to the following:

- The variable knows its parent variable, which is necessarily a neighbor in the constraint graph, therefore this does not violate topology privacy.
- The variable knows its pseudo-parents, which are also neighbors. For each such pseudo-parent, the variable discovers the existence of a cycle in the constraint graph, involving itself, its parent, and the given pseudo-parent. This is tolerated by the definition of topology privacy, since the variable is involved in this cycle.

  Since there exists a treepath in the pseudo-tree from the root, through all its pseudo-parents, to itself, and there exists a back-edge between itself and the highest of its pseudo-parents, the variable also discovers the existence of a long cycle involving all these variables. This is also tolerated by our definition of topology privacy, since the variable is also involved in this cycle. Furthermore, notice that the variable does not know the order in which its pseudo-parents are arranged in the cycle.

  Notice also that the variable *does not* discover the existence of possible ancestors in the pseudo-tree other than its neighbors.

- The variable knows its children, which are also neighbors. Because each child is in a different branch, the variable can infer that there does not exist any constraint between any of its children. This is tolerated by the definition of topology privacy.

- The variable knows its pseudo-children, which are also neighbors. Furthermore, it knows below which of its children each pseudo-child is situated in the pseudo-tree. Therefore, for each pseudo-child below any given child, the variable discovers the existence of a cycle involving itself, the child and the pseudo-child. This is tolerated since the variable is involved in this cycle.

  Moreover, like for its parent and pseudo-parents, for each child, the variable discovers the existence of a long cycle involving itself, the given child, and all pseudo-children below that child. And, unlike for pseudo-parents, the variable can infer the order of its pseudo-children in the cycle, based on the order in which the CHILD token has been received from the pseudo-children. This is also tolerated by the definition of topology privacy, since the variable is involved in that cycle.

  Finally, like for children, the variable discovers the absence of constraints between any two (pseudo-)children located below two different children, which is tolerated. The variable *does not* discover the existence of other possible descendants.

□

References

Cheung, T.-Y. (1983). Graph traversal techniques and the maximum flow problem in distributed computation. *IEEE Transactions on Software Engineering*, *9*(4), 504–512.