Certification of Distributed Algorithms Solving Problems with Optimal Substructure

Kim Völlinger([✉]) and Wolfgang Reisig

Humboldt-Universität Zu, Berlin, Germany
{kim.voellinger,reisig}@informatik.hu-berlin.de

Abstract. We report work-in-progress on applying the concept of a certifying algorithm to distributed algorithms. A *certifying algorithm* produces not only a result, but also a *witness* that verifies the result's correctness. Certifying variants of numerous (sequential) algorithms have been developed. However, *distributed algorithms* behave differently from sequential algorithms. Consequently, it is challenging to make them certifying. Our *local approach* is to make the distributed algorithm compute many *local* witnesses that together verify the result's correctness. We identified problems for which this approach is applicable. Particularly, we hypothesize that for problems with *optimal substructure* (i.e., an optimal solution can be constructed from optimal solutions of its subproblems) it is often easy to apply the local approach. As an example, we give a *certifying* distributed algorithm for the shortest path problem.

Keywords: Distributed algorithms · Certifying algorithms · Optimal substructure · Shortest path problem

1 Introduction

A major problem in software engineering is assuring the quality of software. Well-known methods are testing and formal verification. However, testing does not cover all inputs and formal verification is often infeasible. Moreover, both methods are not fault-tolerant: they are completed before the program is delivered; hence, they cannot deal with failures occurring after delivery. Certifying algorithms are an alternative: we adapt the underlying algorithm of a program to protect a user of this program against a faulty algorithm, implementation and execution. Thus, certifying algorithms are a formal, fault-tolerant method. Numerous certifying *sequential* algorithms have been developed. We report work-in-progress on applying the concept of a certifying algorithm to *distributed* algorithms.

1.1 Certifying Sequential Algorithms

As an example, we consider the problem of deciding if a graph is bipartite, i.e. if its vertices can be divided in two classes so that each edge has its vertices in

R. Calinescu and B. Rumpe (Eds.): SEFM 2015, LNCS 9276, pp. 190–195, 2015.
DOI: 10.1007/978-3-319-22969-0_14

both classes. Assume an algorithm that decides a given graph G is not bipartite. How can a user of this algorithm be convinced of the result's correctness? An odd cycle in G convinces the user: it implies that G is not bipartite. Hence, it *witnesses* the result's correctness.

A *certifying algorithm* produces a *witness* for each result, i.e. an artifact implying the result's correctness. This implication is the *witness property*. For example, a certifying algorithm deciding bipartiteness produces an odd cycle as a witness if the graph is not bipartite and a bipartition if the graph is. In general, the user of a certifying algorithm has (1) to understand the witness property, and (2) to check if the witness is correct, as the witness is computed by an untrusted algorithm. A certifying algorithm can be accompanied by a *certifier* to help the user with (1), and a *checker* to help the user with (2). A *certifier* is a proof checker containing a proof for each witness property. In case of the bipartiteness example, a certifier contains a proof of the witness property of an odd cycle and of a bipartition. Note that the certifier can check these proofs at design time. The *checker* is an algorithm that checks at runtime if the computed witness is correct. In case of the bipartiteness example, depending on the result, it checks if the computed witness is a subgraph and an odd cycle, or a bipartition. Now, the user has to trust the checker. The rationale is that checking is easier than constructing. Figure 1 sums up the idea of a certifying algorithm. The certifier or checker could also reject if the witness does not imply the result's correctness, or if the computed witness is not correct.

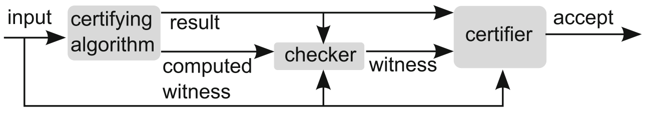

Fig. 1. A certifying algorithm accompanied by its checker and certifier.

When developing a certifying algorithm, the challenge is to find a witness whose proof of the witness property is easy and whose checking is simple. There is always a witness for a correct result, for instance, its computation: in general, however, its proof of the witness property is difficult and therefore, it is not a desirable witness.

1.2 Distributed Algorithms

A *network* is formed by *interactive components* that are connected by message-passing channels. Distributing a computation over a network yields specific problems, such as coordination, communication or synchronization. Distributed algorithms solve these problems. A *distributed algorithm* assigns an algorithm to each component describing the component's computing and communication. For instance, there are distributed algorithms to elect a leader, find a consensus or identify a substructure of the network [7]. A distributed algorithm is designed for a specific network class. We assume an asynchronous model, i.e. no global clock exists. The distributed setting is more complex than the sequential one [6] Sect. 1.3. Thus, it is worth investigating certification of distributed algorithms.

1.3 Structure of this Paper

In Sect. 2, we investigate the challenges of making a distributed algorithm certifying and suggest an approach with local witnesses that together verify the result's correctness. We hypothesize that a distributed algorithm can easily be made certifying if the problem to be solved has optimal substructure. As an example, we give a certifying distributed algorithm for the shortest path problem. As a challenge, we describe the minimum spanning tree example. We discuss related work in Sect. 3 and draw conclusions in Sect. 4.

2 Making Distributed Algorithms Certifying

While non-termination is considered a fault in sequential algorithms, some distributed algorithms should run continuously, e.g. those that deal with failures. Certification of non-terminating algorithms is challenging. However, in this paper, we focus on *terminating* distributed algorithms. After termination, the computed global result is distributed over the network such that each component holds its local result. The result's distribution leads to questions such as should there be a witness for each local result or one witness for the global result; should there be one checker or several; where is a checker located in the network. Making a sequential algorithm certifying is challenging, and even more so for a distributed algorithm.

2.1 Local Approach

Here, our approach is to make a distributed algorithm certifying by making it compute witnesses that together prove the global result's correctness. A witness is *local* to a component if it only contains information from a bounded area in the vicinity of this component. Likewise, a checker is *local* to a component if it has only knowledge about the topology for a bounded area in the vicinity of this component. We consider our approach to be *local* if witnesses and checkers are local, and if the local witnesses together imply the global result's correctness. A (global) certifier holds a proof of this implication – the witness property. Hence, witnesses are computed and checked distributively at runtime. In contrast, the proof of the witness property is checked sequentially by the certifier. This is justified, since it is done once at design time.

We do *not* expect that every distributed algorithm has a localized certifying variant. However, we aim to characterize problems for which the local approach is applicable. So far, problems for which our local approach is applicable include deciding bipartiteness, the echo algorithm, spanning tree construction, maximal independent set construction and shortest path construction. We hypothesize that a distributed algorithm can easily be made certifying if the problem to be solved has *optimal substructure*, i.e. an optimal solution to a problem is constructed from optimal solutions of its subproblems. Every problem that can be solved by dynamic programming has optimal substructure [2].

2.2 Example: Shortest Path Problem

We assume an undirected graph with weighted edges. The *length* of a path is the sum of the weights of its edges. We assume one special vertex, the *source* s. Computing a shortest path from the source to each vertex is the (single-source) *shortest path problem*. The length of a shortest path from the source to a vertex v is called the *distance* of v. A function D is a *distance function* iff $D(v)$ equals the distance from s to v. In networks, the shortest path problem appears in distance-vector routing. We model a network as a graph by representing each component as a vertex and each channel as a weighted edge. Each component computing its distance from the source is the shortest path problem in networks.

Distance Properties. We characterize a distance function by three properties that use the problem's optimal substructure, i.e. a shortest path from s to a vertex v contains a shortest path from s to one of v's neighbors. The distance of v depends on the distances of its neighbors. Let $G = (V, E, s)$ be an undirected, connected graph with a source s. Let $weight : E \to \mathbb{R}_{>0}$ be a function that assigns each edge a weight. We use the following properties for our certifying distributed algorithm. A function $D : V \to \mathbb{R}_{\geq 0}$ is a distance function iff [5]:

$$D(s) = 0 \tag{1}$$

$$\text{for each } (u, v) \in E : D(v) \leq D(u) + weight(u, v) \tag{2}$$

$$\text{for each } v \in V, v \neq s \text{ there exists } (u, v) \in E : D(v) = D(u) + weight(u, v) \tag{3}$$

Certifying Variant of the Distributed Bellman-Ford Algorithm. The distributed Bellman-Ford Algorithm [7] solves the shortest path problem in a network. We assume an undirected, connected network graph whose edges have each a positive weight. Each component i computes its distance i_{D} to the source. In addition, each component i computes a *local witness* i_{w} containing the computed distances of its neighbors. As neighbors send their distances to each other while running the distributed Bellman-Ford algorithm, a component collects the distances to all its neighbors. In addition, we assign each component i a *local checker* that knows the neighbors of i, the weights of i's adjacent edges, and whether i is the source. The local checker of i can check the properties (1)–(3) for i by help of i's result i_{D} and witness i_{w}. In addition, it has to check if the witness i_{w} is consistent with i's neighborhood, i.e. the witness holds the computed distances. The certifier holds a proof of the witness property, i.e. together the local witnesses imply the global result's correctness.[1] To this end, the certifier

[1] We aim to formalize this proof with the proof assistant Coq.

holds a proof for the following implications: if the properties (1)–(3) are fulfilled for each component, they are fulfilled for the network graph; if the properties are fulfilled for the network, the computed distance of each component is correct.

As an example, we discuss witnesses and their checking for the network graph shown in Fig. 2 with a as source. The local checker is a trusted part of its component. Every component holds its local witness after running the certifying distributed Bellman-Ford algorithm. For instance, the local witness of component e contains the computed distances b_{D} and c_{D}. The local checker of e knows that b and c are the neighbors of e, that their associated edge weights are 8 and 1 and that e is not the source. For checking, e's checker gets e's result and witness. It confirms that b and c agree on the computed distances contained in e's witness. For property (1), e's checker has nothing to check since e is not the source. Due to property (2), it has to check if $e_{\mathrm{D}} \leq b_{\mathrm{D}} + 8$ and $e_{\mathrm{D}} \leq c_{\mathrm{D}} + 1$. For property (3), it has to check whether one of these two inequalities is fulfilled as an equality, and, indeed, $e_{\mathrm{D}} = c_{\mathrm{D}} + 1$.

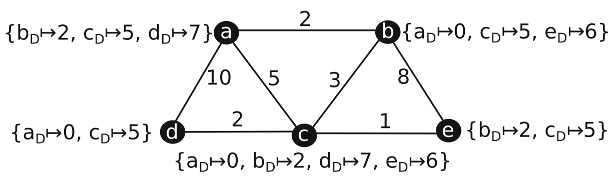

Fig. 2. Network in which every component holds its local witness after running the certifying distributed Bellman-Ford algorithm with a as source.

2.3 Challenge: Minimum Spanning Tree Problem

The minimum spanning tree (MST) problem has optimal substructure. The algorithm of Gallager, Humblet and Spira (GHS) [3] is a well-known, difficult distributed algorithm that computes an MST for an injectively weighted network graph. We aim to find a certifying variant of the GHS. We expect the certifying GHS to be different from the certifying Bellman-Ford algorithm: not every component should compute a witness; instead all the components belonging to an already computed minimum spanning subtree should compute one witness. However, it is not yet clear if we can apply our local approach. Kor et al. show in [4] that a distributed verification of an MST by its result and *without* witnesses is not or not much easier than the distributed construction of an MST.

3 Related Work

Literature offers more than 100 certifying algorithms; several examples are described in [5]. However, none of them is a distributed algorithm. Some techniques for making a distributed algorithm self-stabilizing share similarities to our local approach. The idea of *self-stabilization* is that a system in a faulty state stabilizes itself to a correct state. To this end, the components of a system have to detect that the system's state is faulty whereby local detection is desired [1]. In contrast, we separate the checking from the computation, rely on witnesses, and integrate the proofs of the witness properties.

4 Conclusion and Future Work

A certifying distributed algorithm protects its user against a faulty algorithm, implementation and execution. Therefore, it should be considered as a method for engineering distributed software systems. It is suggested to combine certifying algorithms with other formal methods, such as proving the witness property with a proof assistant, or verifying the checker program. We demonstrated an approach that yields a localized certification in a distributed system, i.e. local witnesses that prove the global result's correctness. We presented a certifying variant of a distributed algorithm solving the shortest path problem for which we used the optimal substructure of this problem.

With our local approach, a component can only be certain of its result's correctness, if *all* checkers accept. However, some local results may be correct even if the global result is not. We aim to allow a component to check its result's correctness more independently. Furthermore, certification of non-terminating algorithms poses new questions such as what the result is; when to produce a witness; or when to check a witness.

References

1. Awerbuch, B., Patt-Shamir, B., Varghese, G., Dolev, S.: Self-stabilization by local checking and global reset (Extended abstract). In: Tel, Gerard, Vitányi, Paul M.B. (eds.) WDAG 1994. LNCS, vol. 857, pp. 326–339. Springer, Heidelberg (1994)
2. Cormen, T.H., Stein, C., Rivest, R.L., Leiserson, C.E.: Introduction to Algorithms, 2nd edn. McGraw-Hill Higher Education, Cambridge (2001)
3. Gallager, R.G., Humblet, P.A., Spira, P.M.: A distributed algorithm for minimum-weight spanning trees. ACM Trans. Program. Lang. Syst. **5**(1), 66–77 (1983)
4. Kor, L., Korman, A., Peleg, D.: Tight bounds for distributed MST verification. In: Schwentick, T., Dürr, C. (eds.) 28th International Symposium on Theoretical Aspects of Computer Science (STACS 2011). Leibniz International Proceedings in Informatics (LIPIcs), vol. 9, pp. 69–80. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2011)
5. McConnell, R.M., Mehlhorn, K., Näher, S., Schweitzer, P.: Certifying Algorithms. Comput. Sci. Rev. **5**, 119–161 (2011)
6. Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. Society for Industrial and Applied Mathematics, Philadelphia (2000)
7. Raynal, M.: Distributed Algorithms for Message-Passing Systems. Springer, Berlin (2013)