Deletion Graph Problems Based on Deadlock Resolution

Alan Diêgo Aurélio Carneiro, Fábio Protti, Uéverton S. Santos

[1]Programa de Pós-Graduação em Computação - IC UFF, Niterói, Brasil

{aaurelio,usouza,fabio}@ic.uff.br

Abstract. *A deadlock occurs in a distributed computing when a group of processes wait indefinitely for resources from each other. Such a property is stable, that is, once occurs into a global state it also will exist in any subsequent global state. Distributed computations are usually represented by wait-for graphs, where the behavior of processes is determined by a deadlock model. In this paper we consider the scenario where deadlock was detected in a system and then some deadlock-breaking set must be found and removed. Hence, given a "snapshot" G of a deadlocked distributed computation which operates according to a deadlock model $\mathbb{M}$, we investigate the complexity of vertex deletion and arc deletion problems whose goal is to obtain the minimum number of removals in order to turn G free of graph structures that characterize deadlocks. The complexity of such problems depends on the deadlock model which governs the computation as well as the type of structure to be removed. The results of this paper are NP-completeness proofs and polynomial algorithms for general and particular graph classes. In special, we show that the arc deletion problem on the OR Model can be solved in polynomial time, and the vertex deletion problem on the OR Model remains NP-Complete even on graphs with $\Delta(G) = 4$, but it is solvable in polynomial time on graphs with $\Delta(G) \leq 3$.*

1. Introdution

A set of processes belongs to a deadlock if each process of this set is blocked, waiting response from another process of this same set; that is, the processes can not proceed their execution, because of a necessary event or response that only another process in the same set can send. Deadlock is a common phenomenon to resource sharing.

A wait-for graph $G = (V, E)$ is an useful abstraction to analyse deadlock situations. The set of vertices V represents processes in a distributed computation and the set of directed edges E represents wait conditions [1]. An edge exists in E directed away from $v_i \in V$ towards $v_j \in V$ if v_i is blocked, waiting a signal from v_j. The graph G changes dynamically according to the dependency model as the computation progresses. The dependency models are known as deadlock models and, in essence, they determine the development of G. More precisely, a deadlock model specifies rules for vertices that are not *sinks* in G to become sinks [2]. (A sink is a vertex with out-degree zero.)

The main deadlock models that have been investigated so far in the literature are presented below.

AND model – A process v_i can only become a sink when v_i receives a signal from all of the processes in O_i.

OR model – a process v_i becomes a sink when v_i receives a signal from one of the processes in O_i.

X-Out-Of-Y model – There are two integers, x_i and y_i, associated with a process v_i. In order to be relieved from its wait state, it suffices for v_i to receive a signal from any x_i of those y_i processes.

AND-OR model – There are $t_i \geq 1$ subsets of O_i associated with process v_i. These subsets are denoted by $O_i^1, \ldots, O_i^{t_i}$ and must be such that $O_i = O_i^1 \cup \cdots \cup O_i^{t_i}$. It suffices for a process v_i to become a sink to receive a signal from all processes in at least one of $O_i^1, \ldots, O_i^{t_i}$.

Once a deadlock is detected, only a external intervention may break it. The contributions of this work are to provide an analysis of the computational complexity of optimization problems, so-called deletion problems, related to deadlock resolution. Given a deadlocked distributed computation G which operates according to a deadlock model $\mathbb{M} \in \{\text{AND, OR, X-OUT-OF-Y, AND-OR}\}$, we investigate vertex-deletion and arc-deletion problems whose goal is to obtain the minimum number of removals in order to turn G free of graph structures that characterize deadlocks. The complexity of such problems depends on the deadlock model which governs the computation as well as the type of structure to be removed. To the best of our knowledge, this computational complexity mapping considering the particular combination of deletion operations and deadlock models for deadlock resolution is novel.

Due to space constraints, all proofs will be omitted.

2. Deletion Problems

We define λ–DELETION($\mathbb{M}$) as a generic optimization problem for deadlock resolution, where λ indicates the type of deletion operation to be used in order to break all the deadlocks of the input graph, and $\mathbb{M} \in \{\text{AND, OR, X-OUT-OF-Y, AND-OR}\}$ is the deadlock model of the wait-for graph G.

The types of deletion operations considered in this work are explained below:

1. **Arc:** The intervention is given by arc removal. The removal of an arc can be seen as the preemption of a resource.
2. **Vertex:** The intervention is given by vertex removal. The removal of a vertex can be seen as the abortion of one process.
3. **Outputs:** The intervention is given by removing all out-edges of a vertex.The removal of all out-edges of a vertex can be interpreted as an immediate transformation of a blocked process into an executable process by preempting all its required resources.

3. Computational Complexities

As deadlock detection can be done in polynomial time for any model $\mathbb{M} \in \{\text{AND, OR, X-OUT-OF-Y, AND-OR}\}$, it remains to show NP-hardness proofs or polynomial algorithms for our problems. The complexities are proved in the theorems below.

Theorem 1.

(a) ARC–DELETION(AND) *is NP-Hard.*

(b) VERTEX–DELETION(AND) *is NP-Hard.*

(c) OUTPUTS–DELETION(AND) *is NP-Hard.*

Theorem 2.

(a) ARC–DELETION(OR) *can be solved in polynomial time.*

(b) OUTPUTS–DELETION(OR) *can be solved in linear time.*

Theorem 3.

(a) ARC–DELETION(AND-OR), VERTEX–DELETION(AND-OR) *and* OUTPUTS–DELETION(AND-OR) *are NP-Hard.*

(b) ARC–DELETION(X-OUT-OF-Y), VERTEX–DELETION(X-OUT-OF-Y) *and* OUTPUTS–DELETION(X-OUT-OF-Y) *are NP-Hard.*

The Table 1 presents the computational complexities of the problems presented so far. The complexity analysis of VERTEX–DELETION(OR) is presented in the next section.

λ–DELETION($\mathbb{M}$)				
	E	OR	AND-OR	X-Out-Of-Y
Arc	NP-H	P	NP-H	NP-H
Vertex	NP-H	?	NP-H	NP-H
Sink	NP-H	P	NP-H	NP-H

Table 1. Partial scenario of λ–DELETION($\mathbb{M}$) complexity.

4. Vertex–Deletion(OR)

We explore different classes of graphs in order to discover features that make the problem VERTEX–DELETION(OR) NP-hard or solvable in polynomial time. We show that VERTEX–DELETION(OR) is NP-hard via a 3-SAT reduction.

Lemma 4. VERTEX–DELETION(OR) *is NP-Hard.*

In general, a wait-for graph on the OR model can be seen as a conglomerate of several strongly connected components. The problems that can be solved in polynomial time has one characteristic in common: it suffices to simply solve every knot in G because no other SCC will turn into a knot after these removals. The next result handles with the natural question "Can VERTEX–DELETION(OR) be solved in polynomial time when the input graph is strongly connected (i. e., G is a single knot)?".

Corollary 5. VERTEX–DELETION(OR) *remains NP-Hard even when G is strongly connected.*

Now we consider properties of the underlying undirected graph of G.

Since one of the most used architectures in distributed computation is the user/server achitecture, a intuitively interesting graph class for distributed computation purposes are bipartite graphs.

Theorem 6. VERTEX–DELETION(OR) *remains NP-Hard even when the underlying undirected graph of G is bipartite, planar and have maximum degree equal to* 4.

We proved that even for graphs with $\Delta(G) = 4$ the problem remains NP-Hard. Furthermore, the problem for graphs with $\Delta(G) \leq 2$ is trivial [4]. Thus, we explore the complexity of VERTEX–DELETION(OR) when the underlying undirected subcubic graph of G (graph with maximum degree three); in this case we obtain aa $O(m\sqrt{n})$ algorithm.

When we study the vertices characteristics, based on the in- and out-degrees, we are able to phase out unnecessary vertices such as sources and sinks. After that, all vertices

of G are in deadlock. The next step is to continuously analyse graph aspects in order to establish rules and procedures that may define specific vertices that are part of an optimum solution.

By classifying the remaining vertices into types, we were able to identify cases where a knot can be solved with only one removal without spreading to others SCCs. By analyzing topological characteristics, we can obtain some reductions presented below, that provide a partial solution and a smaller and more restricted graph to consider.

Observation 1. *Any SCC with more than one output to a knot can be disregarded.*

Lemma 7. *Any SCC C^1 that reaches any SCC C^2 that is not a knot can be disregarded. Furthermore, if C^1 is at distance 1 of some knot, a reduction is applicable to Q.*

From the reductions presented, we have that the graph to be analyzed, without loss of generality, has only knots that are directed cycles where each vertex in those knots have an input edge coming from another SCC. Also, all SCCs (which are not a knot) only reach knots at a distance one. Then, we show that the remaining problem, i.e., freeing the graph of deadlock reduces to finding a (f, g)-semi-matching in SCCs after applying some rules of reduction.

Lemma 8. *[3] Given a bipartite graph $G' = (K \cup C,\ E)$ and two functions $f : k \to \mathbb{N}$ and $g : c \to \mathbb{N}$, where $k \in K$ and $c \in C$, find a maximum (f, g)-semi-matching of G' can be done in polynomial time.*

Finally, we prove that the following theorem holds.

Theorem 9. VERTEX-DELETION(OR) *restricted to subcubic graphs can be solved in $O(m\sqrt{n})$ time.*

Such results are summarized in the following table.

VERTEX–DELETION(OR)

Instance	Complexity
Weakly connected	NP-Hard
Strongly connected	NP-Hard
Planar, bipartite, $\Delta(G) \geq 4$ and $\Delta(G)^+ = 2$	NP-Hard
$\Delta(G) = 3$	Polynomial

Table 2. Complexity of VERTEX–DELETION(OR) for some graph classes.

References

[1] V. C. Barbosa. The Combinatorics of Resource Sharing. In *Models for Parallel and Distributed Computation*, pages 27–52. Springer, 2002.

[2] V. C. Barbosa, A. D. A. Carneiro, F. Protti, and U. S. Souza. Deadlock models in distributed computation: foundations, design, and computational complexity. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pages 538–541. ACM, 2016.

[3] P. Kolman and J. Kratochvíl. *Graph-Theoretic Concepts in Computer Science: 37th International Workshop, WG 2011, Teplá Monastery, Czech Republic, June 21-24, 2011, Revised Papers*, volume 6986. Springer Science & Business Media, 2011.

[4] D. P. Mitchell and M. J. Merritt. A distributed algorithm for deadlock detection and resolution. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 282–284. ACM, 1984.