


Brief Announcement: Multi-Valued Connected Consensus: A New Perspective on Crusader Agreement and Adopt-Commit

Hagit Attiya ✉ 

Department of Computer Science, Technion, Haifa, Israel

Jennifer L. Welch ✉ 

Department of Computer Science and Engineering, Texas A&M University, College Station, TX, USA

Abstract

Algorithms to solve fault-tolerant consensus in asynchronous systems often rely on primitives such as crusader agreement, adopt-commit, and graded broadcast, which provide weaker agreement properties than consensus. Although these primitives have a similar flavor, they have been defined and implemented separately in ad hoc ways. We propose a new problem called *connected consensus* that has as special cases crusader agreement, adopt-commit, and graded broadcast, and generalizes them to handle multi-valued (non-binary) inputs. The generalization is accomplished by relating the problem to approximate agreement on graphs.

We present three algorithms for multi-valued connected consensus in asynchronous message-passing systems, one tolerating crash failures and two tolerating malicious (unauthenticated Byzantine) failures. We extend the definition of *binding*, a desirable property recently identified as supporting binary consensus algorithms that are correct against adaptive adversaries, to the multi-valued input case and show that all our algorithms satisfy the property. Our crash-resilient algorithm has failure-resilience and time complexity that we show are optimal. When restricted to the case of binary inputs, the algorithm has improved time complexity over prior algorithms. Our two algorithms for malicious failures trade off failure resilience and time complexity. The first algorithm has time complexity that we prove is optimal but worse failure-resilience, while the second has failure-resilience that we prove is optimal but worse time complexity. When restricted to the case of binary inputs, the time complexity (as well as resilience) of the second algorithm matches that of prior algorithms.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases graded broadcast, gradecast, binding, approximate agreement

Digital Object Identifier 10.4230/LIPIcs.DISC.2023.36

Related Version *Full Version*: <https://arxiv.org/abs/2308.04646>

Funding *Hagit Attiya*: partially supported by the Israel Science Foundation (grants 380/18 and 22/1425).

1 Introduction

One way to address the impossibility of solving consensus in asynchronous systems is to employ unreliable *failure detectors* [6]. Several algorithms in this class (e.g., [4, 14]) combine a failure detector with a mechanism for detecting whether processes have reached unanimity, in the form of an *adopt-commit* protocol [21]. In such a protocol, each process starts with a binary input value and returns a pair (v, g) where v is one of the input values and g is either 1 or 2. The process is said to pick v as its output value; furthermore, if $g = 2$, then it *commits* to v , and if $g = 1$, then it *adopts* v . In addition to the standard validity property



© Hagit Attiya and Jennifer L. Welch;

licensed under Creative Commons License CC-BY 4.0

37th International Symposium on Distributed Computing (DISC 2023).

Editor: Rotem Oshman; Article No. 36; pp. 36:1–36:7

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

that the output value is the input of some correct process, an adopt-commit protocol ensures that processes commit to at most one value, and if any process commits to a value, then no process adopts the other value.

Another way to address the impossibility of consensus is to use randomization and provide only probabilistic termination. Some algorithms in this class (e.g., [20]) rely on a mechanism called *crusader agreement* [9]: Roughly, if all processes start with the same value v , they must decide on this value, and otherwise, they may pick an *undecided* value, denoted \perp . Other algorithms in this class (e.g., [7]) rely on *graded broadcast* [12], also called *graded crusader agreement*, *graded consensus*, or just *gradedcast*. In a sense, graded broadcast is a combination of adopt-commit with crusader agreement: the decisions are either (v, g) , where v is a binary value and g is either 1 or 2, or \perp (also denoted $(\perp, 0)$). As in adopt-commit, the requirement is that processes commit to at most one value, but in addition, if any process *adopts* a value, then no process adopts the other value. In a sense, the \perp value allows a separation between adopting one value and adopting a different value.

The relation between crusader agreement, adopt-commit and graded broadcast becomes apparent when they are pictorially represented, as in Figure 1, with the possible decisions represented by vertices on a path. The different “convergence” requirements all boil down to ensuring that processes decide on *the same or adjacent vertices* on the path.

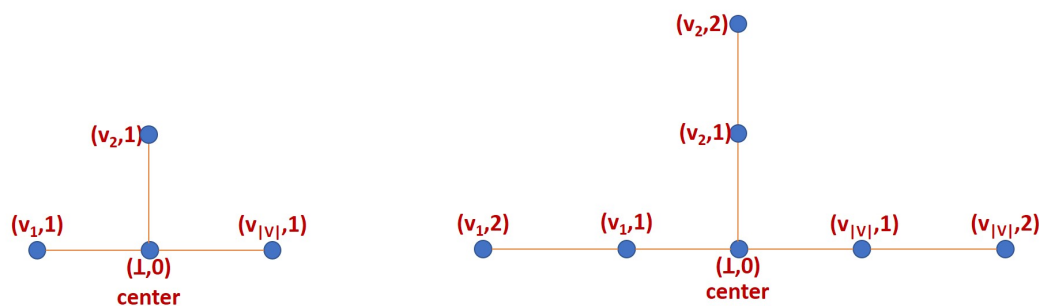
With binary inputs, this description of the problems resembles *approximate agreement on the $[0, 1]$ real interval with parameter ϵ* [10]: processes start at the two extreme points of the interval, 0 or 1, and must decide on values that are at most ϵ apart from each other. Decisions must also be valid, i.e., contained in the interval of the inputs.

Indeed, crusader agreement reduces to approximate agreement with $\epsilon = \frac{1}{2}$: Run approximate agreement with your input (0 or 1) to get some output y , then choose the value in $\{0, \frac{1}{2}, 1\}$ that is closest to y (taking the smaller one if there are two such values, e.g., for $y = \frac{1}{4}$). Finally, return \perp if $\frac{1}{2}$ is chosen. (A similar observation is noted in [11, 16].) Likewise, adopt-commit reduces to approximate agreement with $\epsilon = \frac{1}{3}$, and graded consensus to taking $\epsilon = \frac{1}{4}$. This connection makes it clear why *binary* crusader agreement, adopt-commit and graded broadcast can be solved in an asynchronous message-passing system, in the presence of crash and malicious (unauthenticated Byzantine) failures, within a small number of communication rounds.

In some circumstances, agreement must be reached on a *non-binary* value, e.g., the identity of a leader, or the next operation to apply in state machine replication. To handle *multi-valued* inputs, where processes can start with an input from some set V with $|V| \geq 2$, we define a new problem, *connected consensus*. Connected consensus elegantly unifies seemingly-diverse problems, including crusader agreement, graded broadcast, and adopt-commit, and generalizes them to accept multi-valued inputs. The definition takes inspiration from *approximate agreement on graphs* [5], in which each process starts with a vertex of a graph as its input and must decide on a vertex such that all decisions are within distance one of each other and within the convex hull of the inputs.



■ **Figure 1** Left: crusader agreement. Center: adopt-commit. Right: graded broadcast.



■ **Figure 2** Spider graphs: $R = 1$ (left) and $R = 2$ (right).

2 Connected Consensus and Related Problems

Connected consensus can be viewed as approximate agreement on a restricted class of graphs, called *spider graphs* [15]. These graphs consist of a central clique (which could be a single vertex) to which are attached $|V|$ paths (“branches”) of length R , the *refinement parameter*.

More formally, let V be a finite, totally-ordered set of values; assume $\perp \notin V$. Given a positive integer R , let $G_S(V, R)$ be the “spider” graph consisting of a central vertex labeled $(\perp, 0)$ that has $|V|$ paths extending from it, with one path (“branch”) associated with each $v \in V$. The path for each v has R vertices on it, not counting $(\perp, 0)$, labeled $(v, 1)$ through (v, R) , with (v, R) being the leaf. (See Figure 2.) Given a subset V' of V , we denote by $T(V, R, V')$ the minimal subtree of $G_S(V, R)$ that connects the set of leaves $\{(v, R) | v \in V'\}$; note that when V' is a singleton set $\{v\}$ then $T(V, R, \{v\})$ is the single (leaf) vertex (v, R) .

In the *connected consensus problem for V and R* , each process has an input from V . The requirements are:

Termination: Each correct process must decide on a vertex of $G_S(V, R)$, namely, an element of $\{(v, r) | v \in V, 1 \leq r \leq R\} \cup \{(\perp, 0)\}$.

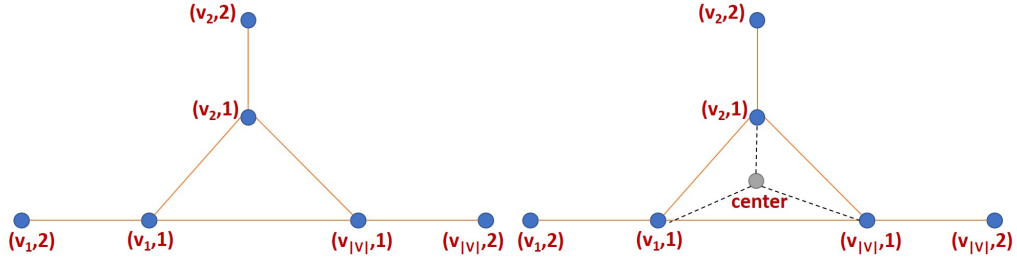
Validity: Let $I = \{(v, R) | v \text{ is the input of a (correct)}^1 \text{ process}\}$. The output of each (correct) process must be a vertex in $T(V, R, I)$. In particular, if all (correct) processes start with the same input v , then (v, R) must be decided.

Agreement: The distance between the vertices labeled by the decisions of all (correct) processes is at most one.

Setting $R = 1$ gives *crusader agreement* [9]. Setting $R = 2$ gives *graded broadcast* [13], also called *adopt-commit-abort* [8].

Recently, the definition of binary (graded) crusader agreement was extended to include a *binding* property [1]: “before the first non-faulty party terminates, there is a value $v \in \{0, 1\}$ such that no non-faulty party can output the value v in any continuation of the execution.” That paper demonstrates that this property facilitates the modular design of randomized consensus algorithms that tolerate an *adaptive* adversary. We refer to [1] for an excellent description of the usage, and its pitfalls, of (graded) crusader agreement, together with common coin protocols, in randomized consensus; they show how faster (graded) crusader agreement algorithms lead to faster randomized consensus algorithms.

¹ When “correct” is in parentheses, it only applies for the case of malicious failures.



■ **Figure 3** Centerless spider graph with $R = 2$ (left) and its reduction to a (centered) graph (right).

We generalize the binding property to hold for multi-valued inputs: once the first process decides, one value is “locked”, so that in all possible extensions, the decisions are on the same branch of the spider graph. Formally:

Binding: In every execution prefix that ends with the first (correct) process deciding, one value is “locked”, meaning that in every extension of the execution prefix, the decision of every (correct) process must be on the same branch of the spider graph.

If the first decision is not $(\perp, 0)$, then this condition follows from Agreement. More interestingly, if the first decision is $(\perp, 0)$, then there are many choices as to which branch is locked but the choice must be the same in every extension. Note that when $|V| = 2$, our definition is equivalent to the original one [1], but for larger V , our definition is stronger – the original definition only excludes one value, leaving $|V| - 1$ possible decision values, while ours excludes $|V| - 1$ values, leaving only one possible decision value.

When $R = 1$, there are only two vertices on any given branch of the spider graph, $(v, 1)$ and $(\perp, 0)$. Thus, the Binding property implies the Agreement property. If $R = 2$, though, the Binding property only restricts the branch of the spider graph on which decisions can be made; both $(\perp, 0)$ and $(v, 2)$ are on the same branch, but Agreement does not permit them to both be decided.

Recall that in *adopt-commit* [14, 21], processes return a pair (v, g) where v is one of the input values and g is either 1 (adopt) or 2 (commit). Thus, there is no analog of the “center” vertex. We model this with a *centerless* spider graph (see left side of Figure 3). Here, $G_S(V, R)$ is the graph consisting of a clique on the vertices $(v, 1)$ for all $v \in V$, each with a path extending from it, with $R - 1$ vertices on it, not counting $(\perp, 0)$, labeled $(v, 2)$ through (v, R) , with (v, R) being the leaf. Decisions must satisfy Termination, Validity and Agreement as specified for the variant with a center. Since the graph has no center, binding cannot be defined; indeed, when a process returns $(v, 1)$, other processes might return $(v', 1)$, for $v \neq v'$.

The centerless problem can be reduced to the centered problem with the same refinement parameter: Call the algorithm for the centered problem with your input u . If the return value is (v, g) with $g > 0$, then decide this value for the centerless problem; when the return value is $(\perp, 0)$, decide $(u, 1)$ for the centerless problem. (See right side of Figure 3.)

In the *vacillate-adopt-commit* (VAC) problem [2], the possible output values are (v, commit) , (v, adopt) , and $(v, \text{vacillate})$, where v is any value. If any output is (v, commit) , then every other output is either (v, commit) or (v, adopt) , for the same v . Furthermore, if there is no commit output and there is at least one (v, adopt) output, then every other output is either (v, adopt) , with the same value v , or $(w, \text{vacillate})$, where w can be any value. VAC corresponds to a centerless spider graph with refinement parameter $R = 3$. However, a closer look at the usage of VAC suggests that the return value of vacillate is irrelevant and the problem could be represented as a centered spider graph with $R = 2$.

3 New Algorithms for Connected Consensus

With these definitions at hand, we turn to designing algorithms for connected consensus in asynchronous message-passing systems that tolerate crash or malicious failures. There is an algorithm for approximate agreement on general graphs in the presence of malicious failures [19]. However, it requires exponential local computation and does not satisfy the Binding property. We are interested in special-case spider graphs, as described above; furthermore, we focus on the cases when the refinement parameter R equals either 1 or 2, which captures the applications of interest. Thus we exploit opportunities for optimizations to obtain better algorithms.

For *communication complexity*, we count the maximum, over all executions, of the number of messages sent by all the (correct) processes. We adopt the definition in [3] for *time complexity* in an asynchronous message-passing system. We start by defining a timed execution as an execution in which nondecreasing nonnegative integers (“times”) are assigned to the events, with no two events by the same process having the same time. For each timed execution, we consider the prefix ending when the last correct process decides, and then scale the times so that the *maximum time* that elapses between the sending and receipt of any message between correct processes is 1. We define the time complexity as the maximum, over all such scaled timed execution prefixes, of the time assigned to the last event. (For simplicity, we assume all processes start at time 0.) This definition of time complexity is analogous to that in [17, 18], which measures the length of the longest sequence of causally related messages.

We present an algorithm for $R = 1$ and $R = 2$ with the Binding property that tolerates crash failures assuming $n > 2f$, where n is the number of processes and f is the maximum number of faulty processes, which is optimal. Its time complexity is R and its message complexity is $O(n^2)$. The message complexity is optimal and the time complexity is optimal for reasonable resiliencies. The best previous algorithms, in [1], have slightly worse time complexity: 2 for $R = 1$ (crusader agreement) and 3 for $R = 2$ (graded crusader agreement). Furthermore, both of these previous algorithms are for the binary case ($|V| = 2$) only.

For malicious failures, we first present a simple algorithm with Binding for $R = 1$ and $R = 2$, that assumes $n > 5f$. Like the crash-tolerant algorithm, its time complexity is R and its message complexity is $O(n^2)$. The message complexity is optimal and the time complexity is optimal for reasonable resiliencies. Both this algorithm and our crash-tolerant one derive the Binding property from the inputs of the processes. That is, the assignment of input values to the processes uniquely determines which non- \perp value, if any, can be decided in any execution with that input assignment. The fact that Binding is determined solely by the inputs is conducive to the development of simple and efficient algorithms. However, we show that in the presence of malicious failures Binding cannot be determined solely by the inputs when $n < 5f$, even if faulty processes do not equivocate.

Our main algorithmic contribution is a connected consensus algorithm for $R = 1$ and $R = 2$ with Binding that tolerates f malicious failures, where $n > 3f$. A simple proof shows that this is the optimal resilience. Its time complexity is 5 for $R = 1$ and 7 for $R = 2$, and its message complexity is $O(|V| \cdot n^2)$, where V is the set of input values. The message complexity can be reduced to $O(n^2)$, at the cost of increasing the time complexity by 2, using techniques of [18].

The upper bounds of 5 and 7 on the time complexity are tight for our algorithm, as shown by giving a concrete execution. The execution uses $V = \{0, 1\}$ and it is also an execution of the crusader agreement algorithm in [1], implying that the tight time complexity of the

■ **Table 1** Summary of connected consensus algorithms for $R = 1$ (crusader agreement) and $R = 2$ (graded broadcast) with input set V ; all algorithms satisfy Binding.

failure type	crash		malicious			
	this paper	[1] ($ V = 2$)	this paper	this paper	this paper + [18]	[1] ($ V = 2$)
resilience	$n > 2f$	$n > 2f$	$n > 5f$	$n > 3f$	$n > 3f$	$n > 3f$
messages	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(V \cdot n^2)$	$O(n^2)$	$O(n^2)$
time $R = 1$	1	2	1	5	7	5
time $R = 2$	2	3	2	7	9	7

latter algorithm is also 5, and that of the graded broadcast algorithm in [1] is 7. This is in contrast to the *round complexities* of 4 and 6 calculated in [1] for their algorithms. The round complexity counts the number of broadcasts performed by an algorithm. However, in their algorithms (as well as in ours), waiting conditions are imposed before performing the next broadcast. If the condition is simply to receive enough messages from the previous broadcast, then at most one time unit elapses per broadcast. But when there is an additional condition, then the condition may take more than one time unit to become true.

4 Discussion

This paper presents the *connected consensus* problem. A numeric *refinement* parameter, R , allows connected consensus to generalize a number of primitives used to solve consensus, including crusader agreement, graded broadcast, and adopt-commit. The problem can be reduced to real-valued approximate agreement when the input set is binary and to approximate agreement on a specific class of *spider* graphs for multi-valued input sets (with two or more inputs). We define the *Binding property* for the multi-valued case, which previously was only defined for the binary case.

We design efficient message-passing algorithms for connected consensus when R is 1 (corresponding to crusader agreement) or 2 (corresponding to graded broadcast), in the presence of crash and malicious failures, for arbitrarily large input sets. The algorithms are modular in that the $R = 2$ case is obtained by appending more communication exchanges to the $R = 1$ case. (Table 1 summarizes our algorithms and relates them to prior work.)

Our algorithm for crash failures has optimal resilience and message complexity. Its time complexity is optimal for reasonable resiliencies and improves on the best previously known algorithms, which only handled binary inputs. We provide two algorithms for malicious failures: One algorithm has time complexity 1 or 2 (for $R = 1$ or $R = 2$) and sends $O(n^2)$ messages, but requires $n > 5f$. The other algorithm only requires $n > 3f$, but has time complexity 5 or 7 (for $R = 1$ or $R = 2$) and sends $O(|V| \cdot n^2)$ messages. This is the same performance as the algorithms in [1] which are only for the case when $|V| = 2$.

An intriguing open question is whether there is some measure, perhaps time, in which solving connected consensus without Binding is more efficient than solving it with Binding?

References

- 1 Ittai Abraham, Naama Ben-David, and Sravya Yandamuri. Efficient and adaptively secure asynchronous binary agreement via binding crusader agreement. In *41st ACM Symposium on Principles of Distributed Computing*, pages 381–391, 2022.
- 2 Yehuda Afek, James Aspnes, Edo Cohen, and Danny Vainstein. Brief announcement: Object oriented consensus. In *36th ACM Symposium on Principles of Distributed Computing*, pages 367–369, 2017. Full version in <https://www.cs.yale.edu/homes/aspnes/papers/vac-abstract.html>.
- 3 Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. McGraw-Hill Publishing Company, 1st edition, 1998.
- 4 Zohir Bouzid, Achour Mostefaoui, and Michel Raynal. Minimal synchrony for Byzantine consensus. In *34th ACM Symposium on Principles of Distributed Computing*, pages 461–470, 2015.
- 5 Armando Castañeda, Sergio Rajsbaum, and Matthieu Roy. Convergence and covering on graphs for wait-free robots. *Journal of the Brazilian Computer Society*, 24:1–15, 2018.
- 6 Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996.
- 7 Giovanni Deligios, Martin Hirt, and Chen-Da Liu-Zhang. Round-efficient Byzantine agreement and multi-party computation with asynchronous fallback. In *19th International Conference on Theory of Cryptography, TCC*, pages 623–653, 2021.
- 8 Carole Delporte-Gallet, Hugues Fauconnier, and Michel Raynal. On the weakest information on failures to solve mutual exclusion and consensus in asynchronous crash-prone read/write systems. *Journal of Parallel and Distributed Computing*, 153:110–118, 2021.
- 9 Danny Dolev. The Byzantine generals strike again. *Journal of Algorithms*, 3(1):14–30, 1982.
- 10 Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. Reaching approximate agreement in the presence of faults. *J. ACM*, 33(3):499–516, 1986.
- 11 Alan David Fekete. Asymptotically optimal algorithms for approximate agreement. *Distributed Computing*, 4:9–29, 1990.
- 12 Paul Feldman and Silvio Micali. Optimal algorithms for Byzantine agreement. In *12th Annual ACM Symposium on Theory of Computing*, pages 148–161, 1988.
- 13 Pease Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous Byzantine agreement. *SIAM J. Comput.*, 26(4):873–933, 1997.
- 14 Eli Gafni. Round-by-round fault detectors: unifying synchrony and asynchrony. In *17th ACM Symposium on Principles of Distributed Computing*, pages 143–152, 1998.
- 15 Manfred Koebe. On a new class of intersection graphs. In *Annals of Discrete Mathematics*, volume 51, pages 141–143. Elsevier, 1992.
- 16 Stephen R Mahaney and Fred B Schneider. Inexact agreement: Accuracy, precision, and graceful degradation. In *4th ACM Symposium on Principles of Distributed Computing*, pages 237–249, 1985.
- 17 Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. Signature-free asynchronous binary Byzantine consensus with $t < n/3$, $O(n^2)$ messages, and $O(1)$ expected time. *J. ACM*, 62(4):31:1–31:21, 2015.
- 18 Achour Mostéfaoui and Michel Raynal. Signature-free asynchronous Byzantine systems: from multivalued to binary consensus with $t < n/3$, $O(n^2)$ messages, and constant time. *Acta Informatica*, 54(5):501–520, 2017.
- 19 Thomas Nowak and Joel Rybicki. Byzantine approximate agreement on graphs. In *33rd International Symposium on Distributed Computing*, pages 29:1–29:17, 2019.
- 20 Sam Toueg. Randomized Byzantine agreements. In *3rd ACM Symposium on Principles of Distributed Computing*, pages 163–178, 1984.
- 21 Jiong Yang, Gil Neiger, and Eli Gafni. Structured derivations of consensus algorithms for failure detectors. In *17th ACM Symposium on Principles of Distributed Computing*, pages 297–306, 1998.