k-set consensus; uniform k-set consensus; optimality; unbeatability; topology; knowledge

1. INTRODUCTION

The k-set consensus problem [7], which allows processes to decide on up to k distinct values, has played an important role in the study of distributed systems for more than two decades. Its analysis provided deep insights into concurrency and solvability of tasks in fault-tolerant settings, both synchronous and asynchronous (e.g. [1, 2, 8, 14, 15, 22, 28]). Whereas combinatorial techniques sufficed for the study of the traditional (1-set) consensus problem [13], establishing lower bounds and impossibility results for the more general k-set consensus has proven to be more challenging. Tackling these has given rise to the introduction of topological techniques to the theory of distributed systems, and these have become a central tool (see [19] for a detailed treatment of the subject).

In the synchronous message passing model, the literature distinguishes between uniform and nonuniform variants of the classic consensus problem. In nonuniform consensus only the correct processes (the ones that do not crash) are required to decide on the same value, while values decided on by processes who crash are allowed to deviate from the value decided by correct processes. In uniform consensus, however, all decisions must be the same. In asynchronous models the two variants coincide, since there is never a guarantee that a silent process has crashed. Perhaps due to the fact that k-set consensus was first studied in the asynchronous setting, only its uniform variant (in which the total number of distinct values decided by correct and faulty processes is no greater than k) has been considered in the literature.

This paper is concerned with k-set consensus in the synchronous message-passing model. In this setting, there are well-known bounds relating the degree of coordination that can be achieved (captured by the parameter k), the number of process crashes that occur in a given execution (typically denoted by f), and the time required for decision, which is \( \lfloor f/k \rfloor + 2 \) in the uniform case (see [1, 14, 15]). Given an \textbf{a priori} bound of \( t \) on the total number of crashes in a run, the worst-case lower bound for decision is \( \lfloor f/k \rfloor + 1 \) [8].

We consider both uniform and nonuniform variants of k-set consensus. For the nonuniform case, we present a protocol that we call \( \text{Opt}_{\min[k]} \), which is unbeatable, in the sense of [4]. Unbeatability is a very strong form of optimality: A k-set consensus protocol \( P \) that always decides at least as soon as \( \text{Opt}_{\min[k]} \) cannot have even one process ever decide strictly earlier than when that process decides in \( \text{Opt}_{\min[k]} \). For the uniform case, we present a protocol that we call \( \text{Opt}_{\min[k]} \), which is built using methods similar to those used for \( \text{Opt}_{\min[k]} \), and which strictly beats all known
protocols, often by a large margin. In many cases, \( u-P_{\text{min}}[k] \) decides in 2 rounds against an adversary for which the best known early-deciding protocols would decide in \( \lceil \frac{k}{r} \rceil + 1 \) rounds. Whether \( u-P_{\text{min}}[k] \) is unbeatable remains an open problem.

The first unbeatable protocols for consensus (in both uniform and nonuniform variants) were presented in [4]. The analysis of those unbeatable consensus protocols shows that the notion of a hidden path is central to the inabilty to decide in consensus. Roughly speaking, a hidden path w.r.t. a process \( i \) at the end of round \( m \) is a sequence of processes that crash one after the other and could inform some process \( j \) at time \( m \) of an initial value unknown to \( i \).

Our improved \( k \)-set consensus protocols are based on an observation that the time required for decision does not depend simply on the number of processes that crash in each round. Only failures that occur in a very specific pattern, maintaining what we call a hidden capacity of \( k \), can prevent processes from being able to decide. The hidden capacity is a generalization of the notion of a hidden path, which, as mentioned above, was shown in [4] to play an important role in consensus.

The main contributions of this paper are:

- We provide new solutions to \( k \)-set consensus in the synchronous message passing model with crash failures. For the nonuniform case, we present the first unbeatable protocol for \( k \)-set consensus, called \( \text{Opt}_{\text{min}}[k] \): No protocol can beat the decision times of \( \text{Opt}_{\text{min}}[k] \). For uniform \( k \)-set consensus, we present a protocol \( u-P_{\text{min}}[k] \) that strictly beats all known early-deciding solutions in the literature [1, 8, 14, 15, 26], in some cases beating (all of) them by a large margin.

- We identify a quantity called the hidden capacity of a given execution w.r.t. a process \( i \) at time \( m \), which plays a major role in determining the decision times in runs of \( k \)-set consensus. Roughly speaking, once its hidden capacity drops below \( k \), process \( i \) can decide. Maintaining a hidden capacity of \( k \) requires at least \( k \) processes to crash in every round. However, they must crash in a very particular fashion. In previous solutions to \( k \)-set consensus in this model (see, e.g., [14]), a process that observes \( k \) or more new failures per round will not decide. In our protocols, decision can be delayed only as long as failures maintain a hidden capacity of \( k \) or more.

- Proving unbeatability has the flavor of a lower bound or impossibility proof. Unsurprisingly, proving the unbeatability of \( \text{Opt}_{\text{min}}[k] \) is extremely subtle. We present two different proofs of its unbeatability. One is a completely constructive, combinatorial proof, while the other is a nonconstructive, topological proof based on Sperner’s lemma. The latter is a new style of topological proof, since it addresses the local question of when an individual process can decide, rather than the more global question of when the last decision is made. This sheds light on the open problem and challenge posed by Guerraoui and Pochon in [16] regarding how topological reasoning can be used to obtain bounds on local decisions. Topological lower bounds for \( k \)-set consensus (e.g. [8, 15]) have established that \( k - 1 \) connectivity (in addition to a Sperner coloring) precludes the possibility that all processes decide. Our analysis provides further insight into the topological analysis for local decisions, illustrating that hidden capacity of \( k \) implies \( k - 1 \) connectivity of a subcomplex of the protocol complex in a given round. The hidden capacity thus explains the source of topological connectivity that underlies the lower-bound proofs for \( k \)-set consensus. To the best of our knowledge, this is the first time knowledge and topological techniques directly interact to obtain a topological characterization.

The full paper [5] contains the detailed technical analysis that supports the claims in this extended abstract. This includes full proofs of all technical claims. In particular, both the full combinatorial proof of unbeatability and the corresponding topological proof are presented there.

2. PRELIMINARY DEFINITIONS

2.1 Computation and Communication Model

Our model of computation is a synchronous, message-passing model with benign crash failures. A system has \( n \geq 2 \) processes denoted by \( \text{Procs} = \{1, 2, \ldots, n\} \). Each pair of processes is connected by a two-way communication link, and each message is tagged with the identity of the sender. Processes share a discrete global clock that starts at time 0 and advances by increments of one. Communication in the system proceeds in a sequence of rounds, with round \( m + 1 \) taking place between time \( m \) and time \( m + 1 \). Each process starts in some initial state at time 0, usually with an input value of some kind. In every round, each process first performs a local computation, and performs local actions, then it sends a set of messages to other processes, and finally receives messages sent to it by other processes during the same round. We consider the local computations and sending actions of round \( m + 1 \) as being performed at time \( m \), and the messages are received at time \( m + 1 \).

A faulty process fails by crashing in some round \( m \geq 1 \). It behaves correctly in the first \( m - 1 \) rounds and sends no messages from round \( m + 1 \) on. During its crashing round \( m \), the process may succeed in sending messages on an arbitrary subset of its links. At most \( t \leq n - 1 \) processes fail in any given execution.

It is convenient to consider the state and behavior of processes at different (process-time) nodes, where a node is a pair \((i, m)\) referring to process \( i \) at time \( m \). A failure pattern describes how processes fail in an execution. It is a layered graph \( F \) whose vertices are all nodes \((i, m)\) for \( i \in \text{Procs} \) and \( m \geq 0 \). An edge has the form \((\langle i, m - 1 \rangle, \langle j, m \rangle)\) and it denotes the fact that a message sent by \( i \) to \( j \) in round \( m \) would be delivered successfully. Let \( \text{Crash}(t) \) denote the set of failure patterns in which all failures are crash failures, and at most \( t \) crash failures can occur. An input vector describes what input the processes receive in an execution. The only inputs we consider are initial values that processes obtain at time 0. An input vector is thus a tuple \( \vec{v} = (v_1, \ldots, v_n) \) where \( v_j \) is the input to process \( j \). We think of the input vector and the failure pattern as being determined by an external scheduler, and thus a pair \( \alpha = (\vec{v}, F) \) is called an adversary.

A protocol describes what messages a process sends and what decisions it takes, as a deterministic function of its local state at the start of the round. Messages received during a
round affect the local state at the start of the next round. We assume that a protocol $P$ has access to the number of processes $n$ and to the bound $t$, typically passed to $P$ as parameters.

A **run** is a description of an infinite behavior of the system. Given a run $r$ and a time $m$, the **local state** of process $i$ at time $m$ in $r$ is denoted by $r_i(m)$, and the **global state** at time $m$ is defined to be $r(m) = (r_1(m), r_2(m), \ldots, r_n(m))$. A protocol $P$ and an adversary $\alpha$ uniquely determine a run, and we write $r = P^{[\alpha]}$.

Since we restrict attention to benign failure models and focus on decision times and solvability in this paper, Conant showed that it is sufficient to consider full-information protocols (fp's for short), defined below [9]. There is a convenient way to consider such protocols in our setting. With an adversary $\alpha = (i, \mathcal{F})$ we associate a communication graph $\mathcal{G}_\alpha$, consisting of the graph $\mathcal{F}$ extended by labeling the initial nodes $(j,0)$ with the initial states $v_j$ according to $\alpha$. With every node $(i,m)$ we associate a subgraph $\mathcal{G}_\alpha(i,m)$ of $\mathcal{G}_\alpha$, which we think of as $i$’s **view** at $(i,m)$. Intuitively, this graph represents all nodes $(j,\ell)$ from which $(i,m)$ has heard, and the initial values it has seen. Formally, $\mathcal{G}_\alpha(i,m)$ is defined by induction on $m$. $\mathcal{G}_\alpha(i,0)$ consists of the node $(i,0)$, labeled by the initial value $v_i$. Assume that $\mathcal{G}_\alpha(1,m), \ldots, \mathcal{G}_\alpha(n,m)$ have been defined, and let $J \subseteq \text{Procs}$ be the set of processes $j$ such that $j = i$ or $e_j = ((j,m),(i,m+1))$ is an edge of $\mathcal{F}$. Then $\mathcal{G}_\alpha(i,m+1)$ consists of the node $(i,m+1)$, the union of all graphs $\mathcal{G}_\alpha(j,m)$ with $j \in J$, and the edges $e_j = ((j,m),(i,m+1))$ for all $j \in J$. We say that $(j,\ell)$ is **seen** by $(i,m)$ if $(j,\ell)$ is a node of $\mathcal{G}_\alpha(i,m)$. Note that this occurs exactly if $\mathcal{F}$ allows a (Lamport) message chain from $(j,\ell)$ to $(i,m)$.

A full-information protocol $P$ is one in which at every node $(i,m)$ of a run $r = P^{[\alpha]}$ the process $i$ constructs $\mathcal{G}_\alpha(i,m)$ after receiving its round $m$ nodes, and sends $\mathcal{G}_\alpha(i,m)$ to all other processes in round $m+1$. In addition, $P$ specifies what decisions $i$ should take at $(i,m)$ based on $\mathcal{G}_\alpha(i,m)$. Full-information protocols thus differ only in the decisions taken at the nodes. Finally, in a run $r = P^{[\alpha]}$, we define the local state $r_i(m)$ of a process $i$ at time $m$ to be the pair $\langle \beta, \mathcal{G}_\alpha(i,m) \rangle$, where $\beta = \bot$ if $i$ is undecided at time $m$, and if $\beta = v$ in case $i$ has decided $v$ at or before time $m$.

For ease of exposition and analysis, all of our protocols are fp's. However, in fact, they can all be implemented in such a way that any process sends any other process a total of $O(n \log n)$ bits throughout any execution (see the full paper [5] for more details).

### 2.2 Domination and Unbeatability

A protocol $P$ is a worst-case optimal solution to a decision problem $S$ in a given model if it solves $S$, and decisions in $P$ are always taken no later than the worst-case lower bound for decisions in this problem, in a given model of computation. However, this protocol can be strictly improved upon by **early stopping** protocols, which are also worst-case optimal, but can often decide much faster than the original ones. In this paper, we are interested in protocols that are efficient in a much stronger sense.

Consider a context $\gamma = (\mathcal{V}, F)$, where $\mathcal{V}$ is a set of initial vectors. A decision protocol $Q$ dominates a protocol $P$ in $\gamma$, denoted by $Q \preceq_\gamma P$, if, for all adversaries $\alpha$ and every process $i$, if $i$ decides in $P^{[\alpha]}$ at time $m_i$, then $i$ decides in $Q^{[\alpha]}$ at some time $m'_i \leq m_i$. Moreover, we say that $Q$ strictly dominates $P$ if $Q \preceq_\gamma P$ and $P \not\preceq_\gamma Q$. I.e., if it dominates $P$ and for some $\alpha \in \gamma$ there exists a process $i$ that decides in $Q^{[\alpha]}$ strictly before it does so in $P^{[\alpha]}$.

Following [20], a protocol $P$ is said to be an all-case optimal solution to a decision task $S$ in a context $\gamma$ if it solves $S$ and, moreover, $P$ dominates every protocol $P'$ that solves $S$ in $\gamma$. For the standard (eventual) variant of consensus, in which decisions are not required to occur simultaneously, Moses and Tuttle showed that no all-case optimal solution exists [25]. Consequently, Halpern, Moses, and Waarts in [18] initiated the study of a natural notion of optimality that is achievable by eventual consensus protocols:

**Definition 1.** [18] A protocol $P$ is an **unbeatable** solution to a decision task $S$ in a context $\gamma$ if $P$ solves $S$ in $\gamma$ and no protocol $Q$ solving $S$ in $\gamma$ strictly dominates $P$.

Thus, $P$ is unbeatable if for all protocols $Q$ that solve $S$, if there exist an adversary $\alpha$ and process $i$ such that $i$ decides in $Q^{[\alpha]}$ strictly earlier than it does in $P^{[\alpha]}$, then there must exist some adversary $\beta$ and process $j$ such that $j$ decides strictly earlier in $P^{[\beta]}$ than it does in $Q^{[\beta]}$. An unbeatable solution for $S$ is $\preceq_\gamma$-minimal among the solutions of $S$.

### 2.3 Set Consensus

In the $k$-set consensus problem, each process $i$ starts out with an initial value $v_i \in \{0, 1, \ldots, k\}$. Denote by $\exists v$ the fact that at least one of the processes started out with initial value $v$. In a protocol for (nonuniform) $k$-set consensus, the following properties must hold in every run $r$:

- **k-Agreement:** The set of values that correct processes decide on has cardinality at most $k$.
- **Decision:** Every correct process must decide on some value, and
- **Validity:** For every value $v$, a decision on $v$ is allowed only if $\exists v$ holds.

In uniform $k$-set consensus [6, 11, 17, 23, 27, 29], the $k$-Agreement property is replaced by

- **Uniform $k$-Agreement:** The set of values decided on has cardinality at most $k$.

In uniform $k$-set consensus, values decided on by failing processes (before they have failed) are counted, whereas in the nonuniform case they are not counted. The two notions coincide in asynchronous settings, since it is never possible to distinguish at a finite point in time between a crashed process and a very slow one. Uniformity may be desirable when elements outside the system can observe decisions, as in distributed databases when decisions correspond to commitments to values.

---

1 Unbeatable protocols were called optimal in [18]. Following [4], we prefer the term unbeatable because “optimal” is used very broadly, and inconsistently, in the literature.

2 The set of values in $k$-set consensus is often assumed to contain more than $k + 1$ values. We choose this set for ease of exposition. Our results apply equally well with minor modifications if a larger set of values is assumed. See Footnote 4.
3. UNBEATABLE (1-SET) CONSENSUS

Before introducing our protocols for \(k\)-set consensus, we briefly review the analysis and the unbeatable protocol for nonuniform (1-set) consensus given in [4]. In this version of the problem, all processes start with binary initial values \(v_i \in \{0, 1\}\), decisions must satisfy the validity condition, and all correct processes decide on the same value. Recall that we are assuming that the processes follow a full-information protocol, so a protocol can be specified simply by giving the rules by which a process decides on value \(v\) for \(v = 0, 1\).

By Validity, \(\exists v\) is a necessary condition for deciding on a value \(v \in \{0, 1\}\). Consequently, a process cannot decide \(v\) unless it knows that some process had initial value \(v\). Clearly, a process knows \(\exists v\) if it sees a value of \(v\) either as its own initial value or reported in a message that the process receives. [4] consider the design of a protocol that will decide on 0 as soon as possible, namely at the first point at which a process knows \(\exists 0\). They proceed to consider when a process can decide 1, given that all processes decide 0 if they ever come to know that 30. Clearly, we should consider the possibility of deciding 1 only for processes that do not know that 30. By the (1-)Agreement property, correct processes must decide on the same value. Thus, a process cannot decide 1 if another process is deciding 0. Let us consider when a process can know that nobody is deciding 0.

Our analyses will make use of the different types of information that a process \(i\) at time \(m\) can know about the state of a process \(j\) at time \(\ell\), in runs of an \(\text{fip}\). (We denote such process-time pairs by \((i, m), (j, \ell), \text{etc.}\).) We say that \((j, \ell)\) is seen by \((i, m)\) if \(i\) has received a message by time \(m\) containing the state at \((j, \ell)\). We say that \((j, \ell)\) is guaranteed crashed at \((i, m)\) if \(i\) has proof at time \(m\) that \(j\) crashed before time \(\ell\) (\(i\) heard from someone who did not hear from \(j\) in some round \(\leq \ell\)). Finally, we say that \((j, \ell)\) is hidden from \((i, m)\) if it is neither seen by \((i, m)\) nor guaranteed crashed there. As far as \(i\) is concerned, \(j\) may have sent messages in round \(\ell + 1\), and since \(i\) does not see \((j, \ell)\), it may not know at \(m\) what information \(j\)'s messages contained.

Fig. 1 illustrates a case in which process \(i\) does not know \(\exists 0\) at time 2, while process \(i_3\) decides 0 at time 2. This is possible only if there is a hidden path with respect to \((i, 2)\), in the terminology of [4]: at each time \(\ell\) from 0 up to the current time \(m=2\), there is a node \((j, \ell)\) that is hidden from \((i, 2)\).

In general, if there is a hidden path with respect to \((i, m)\) in a given execution, and process \(i\) does not know \(\exists 0\) at time \(m\), then \(i\) cannot be guaranteed that no correct process is currently deciding 0. It thus cannot decide 1. If no such path exists, i.e., if there is some time \(k \leq m\) that contains no hidden node w.r.t. \((i, m)\), then \(i\) knows that nobody is deciding 0. Moreover, in that case \(i\) knows that no value of 0 is known to any active process, and so nobody will ever decide 0. Based on this analysis, [4] proposes the following protocol:

| Protocol Opt0 (for an undecided process \(i\) at time \(m\)) [4]: |
|---------------------------------------------------------------|
| \textbf{if} \ seen 0 \ then \ decide(0) \ |
| \textbf{elseif} \ some \ time \(k \leq m\) \ contains \ no \ hidden \ node \ then \ decide(1) |

As shown in [4], Opt0 is an unbeatable protocol for nonuniform consensus. Moreover, it strictly dominates all previously known early stopping protocols for consensus, in some cases deciding in 3 rounds when the best previously known protocol would decide in \(t + 1\) rounds.

Hidden paths, first defined in [4], are implicit in many lower-bound proofs for consensus in the crash failure model [10, 12]. They play a crucial role in the correctness and unbeatability proof of Opt0. In this paper, we extend the notion of a hidden path, and use it to provide an unbeatable protocol for set consensus, as well as a protocol for uniform set consensus that beats all previously known protocols.

4. UNBEATABLE \(k\)-SET CONSENSUS

The design and especially the analysis of protocols for \(k\)-set consensus are typically considerably more subtle than that of protocols for (1-set) consensus. Indeed, since processes may decide on different values, their decisions no longer depend on hearing about concrete values. Nevertheless, as we now show, it is possible to extend the above notions to obtain a simple protocol for nonuniform \(k\)-set consensus. While the protocol is natural to derive and simple to state, it is unbeatable. Moreover, its proof of unbeatability is highly nontrivial and extremely subtle.

4.1 Protocol Description and Correctness

Recall that the set of possible initial values is assumed to be \(\{0, 1, \ldots, k\}\). As in the case of 1-set consensus, our goal is to define the rules by which a process will decide on \(v\), for every value \(v\) in this set. Define by \(Min(i, m)\) the minimal value that process \(i\) has seen by time \(m\) (i.e., the minimal value \(v\) s.t. \(i\) knows that \(\exists v\)) in a given run of the \(\text{fip}\). Moreover, we say that process \(i\) is low at time \(m\) if \(Min(i, m) < k\). We set out to design a protocol in which each process \(i\) that decides at time \(m\) decides on \(Min(i, m)\).

Suppose that we choose to allow a process that becomes low to immediately decide on \(Min(i, m)\). This is analogous to allowing a process that sees 0 to immediately decide on 0 in 1-set consensus. When should a process be able to decide on the value \(k\)? As already specified, every low process decides on a low value, i.e., a value smaller than \(k\) as soon as possible; therefore, we are only concerned about when high processes (processes that are not low) should decide on a high value, i.e., on \(k\). Of course, a process \(i\) should be able to decide on \(k\) if it knows that doing so will not violate the properties of \(k\)-set consensus, and in particular, the \(k\)-Agreement property, which disallows the correct processes to decide on more than \(k\) values. I.e., if \(i\) considers it to be possible that there is a correct process deciding on each of the values 0, \ldots, \(k - 1\), then \(i\) should not decide on \(k\).

In other words, \(i\) can decide on \(k\) only when it knows that at most \(k - 1\) of the low values will be decided on. Notice that this is not the same as there being only \(k - 1\) possible low initial values in the run, but rather that at most \(k - 1\) values can serve as \(Min(j, m)\) for any process \(j\) that decides at time \(m\). Consider the following generalization of the notion of a hidden path:
Figure 1: A hidden path at time $m = 2$ indicates that a value unknown to $i$ may exist in the system.

**Definition 2.** Fix a run $r$. We define the **hidden capacity** of process $i$ at time $m$, denoted by $HC(i, m)$, to be the maximum number $c$ such that for every $\ell \leq m$, there exist $c$ distinct nodes $(i, \ell), \ldots, (i, \ell)$ at time $\ell$ that are hidden from $(i, m)$. The nodes $(i, \ell)$ are said to be **witnesses to the hidden capacity** of $i$ at $m$.

Put another way, the hidden capacity of $(i, m)$ is at least $c$ if at each time $\ell \leq m$ there are at least $c$ nodes that are hidden from $(i, m)$. In particular, a hidden path implies that the hidden capacity is at least $1$. Recall that $(i, m)$ does not see what happens at nodes $(j, \ell)$ that are hidden from it, and does not know who they heard from in round $\ell$ and whether they communicated to others in round $\ell + 1$.

As we now show, the hidden capacity is very closely related to the ability of a process to decide $k$ at a given time. If the hidden capacity of $(i, m)$ is $c$, then there could be up to $c$ disjoint hidden paths, where each failing process that belongs to one of the hidden paths sends in its crashing round a message solely to its successor in that path. Fig. 2(a) illustrates the node $(i, 2)$ with hidden capacity 3, while Fig. 2(b) illustrates three disjoint hidden paths. If such disjoint paths begin in nodes with distinct initial values, they can end at $c$ nodes, each of which sees a distinct minimal value. It follows that a node $(i, m)$ with hidden capacity $c$ must consider it possible that there are $k$ nodes $(j_0, m), \ldots, (j_{k-1}, m)$ with $Min(j_b, m) = b$ for every $b = 0, \ldots, k-1$. Therefore, $i$ is not be able to decide on the value $k$ at time $m$ without risking violating **$k$-Agreement**.

Interestingly, when the hidden capacity drops below $k$, it becomes possible to decide on $k$. Suppose that the hidden capacity of $(i, m)$ is smaller than $k$. This means that there must be some time $\ell < m$ such that some $c < k$ nodes at time $\ell$ are hidden from $(i, m)$. Each of these nodes has a single minimal value. Assuming that $Min(i, m) = k$, we have that for time $\ell$ nodes seen by $(i, m)$, the minimal value is $k$. Since this is the $fp$, every active process after time $\ell$ will have one of these $c + 1 \leq k$ minimal values. It follows that $i$ can safely decide on $k$ at time $m$ if its hidden capacity is lower than $k$.

The above discussion suggests the following protocol for nonuniform $k$-set consensus:

---

**Protocol $Opt_{\text{min}[k]}$** (for an undecided process $i$ at time $m$):

- if $i$ is low or $i$ has hidden capacity $< k$
  - then decide($Min(i, m)$)

Our protocol $Opt_{\text{min}[k]}$ directly generalizes the unbeatable consensus protocol $Opt_0$. Being low in this case corresponds to seeing 0, while $HC(i, m) < k = 1$ corresponds to there being no hidden path. Recall that the decision rules of $Opt_0$ may be thought of as follows: a process $i$ decides on the value 0 as soon as it knows that some node had initial value 0, and it decides on the value 1 as soon as it knows that no correct process will ever decide on the value 0 (hence **Agreement** is not violated). In $Opt_{\text{min}[k]}$, a process $i$ decides on a low value $v$ (i.e., a value $v \in \{0, \ldots, k-1\}$) as soon as it knows that some node had initial value $v$, and decides on the high value $v = k$ (indeed, for a high node $(i, m)$, we have $Min(i, m) = k$) as soon as it knows that at most $k-1$ values smaller than $v$ will ever be decided on by correct processes (thus satisfying **$k$-Agreement**).

Based on the above analysis, we obtain:

**Proposition 1.** $Opt_{\text{min}[k]}$ solves $k$-set consensus, and all processes decide by time $\lceil k/k \rceil + 1$.

---

### 4.2 Unbeatability

Proving that $Opt_{\text{min}[k]}$ is unbeatable is nontrivial. The main technical challenge along the way is, roughly speaking, showing that, e.g., in the scenario depicted in Fig. 2, each of the hidden processes at time $m = 2$ (i.e., if for $b = 1, 2, 3$ decides on the unique low value $(v_b)$ known to it, not merely in $Opt_{\text{min}[k]}$, but in any protocol $P$ that dominates $Opt_{\text{min}[k]}$).

We note that Proposition 1, as well as all other the results that we present, including Theorems 1–3 below, as well as all proofs in the full paper [5], hold verbatim even if the set of possible initial values is $\{0, \ldots, d\}$, for some $d \geq k$. (In this case, all values in $\{k, k + 1, \ldots, d\}$ are considered high.) In particular, the definition for this case of all protocols, including $Opt_{\text{min}[k]}$, is unchanged.

In particular, the challenges involved are significantly greater than those in involved in proving that $Opt_0$ is unbeatable for (1-set) consensus (as is often the case with problems regarding $k$-set consensus vs. their (1-set) consensus counterparts).
Indeed, in the scenario depicted in Fig. 2, one could imagine a hypothetical protocol in which all active nodes decide at time 2 on some high value, say the initial value of process 1, thus guaranteeing that 1 will not violate \textit{k-Agreement} by deciding on this value immediately as well. The claim that such a hypothetical protocol does not exist is made precise in Lemma 1 (see also a discussion in Section 4.3 below).

**Lemma 1.** Let \( P \) be a protocol solving nonuniform \( k \)-set consensus. Assume that in \( P \), every process \( i \) that is low at any time \( m \) must decide by time \( m \) at the latest. Let \( r \) be a run of \( P \), let \( i \) be a process and let \( m \) be a time. If the following conditions hold in \( r \):

1. \( i \) is low at \( m \) for the first time,
2. \( i \) has seen a single low value \( v \) by time \( m \),
3. \( HC(i, m) \geq k - 1 \), and
4. there exist \( k \) distinct processes \( j_1, \ldots, j_k \) s.t. \( (j_b, m-1) \) is high and \( (j_b, m) \) is hidden from \( (i, m) \), for all \( b = 1, \ldots, k \).

then \( i \) decides in \( r \) on its unique low value \( v \) at time \( m \).

We note that for \( k = 3 \), the nodes \( i_j^2 \) in Fig. 2 indeed meet the requirements of Lemma 1 at time \( m = 2 \). (E.g., for \( i_j^2 \), we may take the processes called \( i_j^2, i_j^1, i_j^3 \) in Fig. 2 to serve as the processes \( j_1, \ldots, j_k \) in the statement of the lemma.) Given Lemma 1, using reasoning similar to our discussion above regarding the decision rules in \( \text{Opt}_{\text{min}[k]} \), we conclude that a high process with hidden capacity at least \( k \) (such as process \( i \) at time \( m = 2 \) in Fig. 2, for \( k = 3 \)) cannot decide in any protocol \( P \) that dominates \( \text{Opt}_{\text{min}[k]} \) without risking violating \textit{k-Agreement}, from which the unbeatability of \( \text{Opt}_{\text{min}[k]} \) follows (indeed, all undecided nodes in \( \text{Opt}_{\text{min}[k]} \) are high and have hidden capacity at least \( k \), and are therefore undecided under \( P \) as well).

**Theorem 1.** \( \text{Opt}_{\text{min}[k]} \) is an unbeatable protocol for nonuniform \( k \)-set consensus in the crash failure model.

Theorem 2. \( \text{Opt}_{\text{min}[k]} \) is last-decider unbeatable for nonuniform \( k \)-set consensus in the crash failure model.

4.2.1 Last-decider unbeatability

In [4] the authors also consider a variation on the notion of unbeatability, called \textit{last-decider unbeatability}, which compares runs in terms of the time at which the last correct process decides. This notion neither implies, nor is implied by, unbeatability as defined above. Interestingly, \( \text{Opt}_{\text{min}[k]} \) is unbeatable in this sense as well:
4.3 A Constructive Combinatorial Approach vs. A Nonconstructive Topological Approach

In the full paper [5], we provide two proofs for Lemma 1. One is combinatorial and completely constructive, devoid of any topological arguments, while the other is nonconstructive and topological, based on Sperner’s lemma. Our topological proof reasons in a novel way about subcomplexes of the protocol complex. Both proofs of Lemma 1 are by induction. In both proofs, the induction hypothesis and the (proof of the) base case are the same. The proofs differ only in the induction step.

In the terminology of Lemma 1, both proofs start by showing that there exists a run that \((i, m)\) finds possible in which each of the \(k – 1\) nodes at time \(m – 1\) that are hidden from \((i, m)\) holds a distinct low value other than \(v\); therefore, by the induction hypothesis, had these \(k – 1\) nodes not failed, they would have each decided on its unique low value. Hence, each of the nodes \(j_1, \ldots, j_k\) must consider it possible that all of the hidden nodes that it sees from time \(m – 1\) have actually decided and are correct. The challenge is to show, without any information about \(P\) except for the initial assumption that it dominates \(\text{OrT}_{\min[k]}\), that there must exist a run \(r’\) of \(P\) that \((i, m)\) finds possible (i.e., a way to adjust the messages received by \(j_1, \ldots, j_k\) at time \(m\)) in which \(j_1, \ldots, j_k\) collectively decide on all low values (including \(v\)) at time \(m\); see Fig. 3. Therefore, \(i\) cannot decide on a high value without violating \(k\)-Agreement (since it must decide, it must do so on a low value, and therefore on the only low value that is has seen, namely \(v\)).

Our combinatorial proof for the induction step constructively builds such a possible run \(r’\) as required (providing a “recipe” for how to adjust the messages received by \(j_1, \ldots, j_k\) at time \(m\) so that they collectively decide on all low values). In contrast, our topological proof uses Sperner’s lemma to show that if process \((i, m)\) does not decide on a low value, then there must exist a run of the protocol that violates \(k\)-Agreement (i.e., a run \(r’\) as described above). It is interesting that the topological proof essentially shows that \((i, m)\) is forced to decide on a low value in the run \(r\) because the star complex, denoted \(\mathcal{S}(i, m, P_m)\), of the node \((i, m)\) in the protocol complex \(P_m\) of the protocol \(P\) at time \(m\), is \((k – 1)\)-connected. Intuitively, \(\mathcal{S}(v, P_m)\) is the “part” of \(P_m\) containing all executions that are indistinguishable to \((i, m)\). That \(\mathcal{S}(i, m, P_m)\) is \((k – 1)\)-connected is the reason why the proof can map a subdivision of \(k\)-simplexes to process states; indeed, the subdivision is mapped to a subcomplex of \(\mathcal{S}(i, m, P_m)\). Therefore, \((i, m)\) has no other choice than to decide on a low value, because if it does not do so, then its decision induces a Sperner coloring, which ultimately (together with the connectivity of \(\mathcal{S}(i, m, P_m)\)) implies that the \(k\)-Agreement property is violated.

It is worth noticing that in this topological analysis we only care about the connectivity of a proper subcomplex of the protocol complex, contrary to all known time-complexity lower-bound proofs [15, 21] for \(k\)-set consensus, which care about the connectivity of the whole protocol complex in a given round. While connectivity of the whole protocol complex is the “right” thing to consider for lower-bound proofs about when all processes can decide, we show that for proving unbeatability, i.e., when concerned with the time at which a single process can decide, the “right” thing to consider is the connectivity of just a subcomplex (the star complex of a given process state).

This analysis sheds light on the open question posed by Guerraoui and Pochon in [16] asking for extensions to previous topology techniques that deal with optimality of protocols. In summary, while all-decide lower bounds have to do with the whole protocol complex (e.g., [21]), optimal-single-decision lower bounds have to do with just subcomplexes of the protocol complex. Our topological proof of unbeatability is the first proof that we are aware of that makes this distinction.

Finally, we emphasize that the connectivity properties of the star complex \(\mathcal{S}(i, m, P_m)\) are due to the hidden capacity of \((i, m)\) in the hypothesis of Lemma 1. Indeed, one can formally relate the connectivity of \(\mathcal{S}(i, m, P_m)\) to the hidden capacity of \((i, m)\), as we now show.

**Proposition 2.** Let \(P_m\) be the \(m\)-round protocol complex containing all \(m\)-round executions of a flip \(P\). If \((i, m)\) is a vertex of \(P_m\) whose view corresponds to a local state with hidden capacity at least \(k\) in each of the \(m\) rounds, then the star complex \(\mathcal{S}(i, m, P_m)\) of \((i, m)\) in \(P_m\) is \((k – 1)\)-connected.

Proposition 2 speaks about a local property in protocol complexes, which turns out to be important for optimality analysis. It is unknown whether the converse of this lemma is true, namely, whether \((k – 1)\)-connectivity of the star complex implies hidden capacity at least \(k\) in every round.

5. UNIFORM SET CONSENSUS

We now turn to consider uniform \(k\)-set consensus. In [4], the concepts of hidden paths and hidden nodes are used to present an unbeatable protocol \(u\)-Opt\(_{\mathcal{T}^0}\) for \((1\)-set\) uniform consensus. In this section, we present a protocol called \(u\)-Opt\(_{\min[k]}\) that generalizes the unbeatable \(u\)-Opt\(_{\mathcal{T}^0}\) to \(k\) values (i.e., for \(k = 1\), it behaves exactly like \(u\)-Opt\(_{\mathcal{T}^0}\)). As in the nonuniform case, the analysis of the case \(k > 1\) is significantly more subtle and challenging; in fact, generalizing the protocol statement in the uniform case is considerably more involved than in the nonuniform case.

While in the protocol \(\text{OrT}_{\min[k]}\) (which is defined in Section 4 for nonuniform consensus) an undecided process \(i\) decides on its minimal value if and only if \(i\) is low or has hidden capacity \(< k\), in \(u\)-Opt\(_{\min[k]}\) we have to be more careful. Indeed, we must ensure that a value (even a low one) that process \(i\) decides upon will not “fade away”. This could happen if \(i\) is the only one knowing the value, and if \(i\) crashes without successfully communicating it to active processes.

The case analysis here is also significantly more subtle than in the case of \((1\)-set\) uniform consensus. To phrase the exact conditions for decision, we begin with a definition; recall that \(t\) is an upper bound on the number of faulty nodes in any given run, and is available to all processes; while curiously the knowledge of \(t\) cannot be used to speed up \(\text{OrT}_{\min[k]}\), it is indeed useful for speeding up decisions in the uniform case.
Definition 3. [4] Let \( r \) be a run and assume that \( i \) knows of \( d \) failures at time \( m \) in run \( r \). We say that \( i \) knows that the value \( v \) will persist at time \( m \) if (at least) one of the following holds.

- \( m > 0 \), and \( i \) both is active at time \( m \) and has seen the value \( v \) by time \( m - 1 \), or
- \( (i, m) \) sees at least \( t - d \) distinct nodes \( (j, m - 1) \) of time \( m - 1 \) that have seen the value \( v \).

As shown in [4], if \( i \) knows at time \( m \) that \( v \) will persist, then all active nodes at time \( m + 1 \) will know \( v \). Everyone’s minimal value will be no larger than \( v \) from that point on.

In \( u-P_{\text{min}[k]} \), an undecided process \( i \) decides on a value \( v \) if and only if \( v \) is the minimal value s.t. \( i \) knows that both

- \( v \) was at some stage the min value known to a process that was low or had hidden capacity \( < k \), and
- \( v \) will be known to all processes deciding strictly after \( i \).

As mentioned above, designing \( u-P_{\text{min}[k]} \) to check that these conditions hold requires a careful statement of the protocol, which we now present.

Protocol \( u-P_{\text{min}[k]} \) (for an undecided process \( i \) at time \( m \)):

- if \( i \) is low or HC\((i, m) < k \) and \( i \) knows that Min\((i, m)\) will persist
  then decide\(\text{Min}(i, m)\)
- else if \( m > 0 \) and \( (i, m - 1) \) was low or HC\((i, m - 1) < k \)
  then decide\(\text{Min}(i, m - 1)\)
- else if \( m = \lceil t/k \rceil + 1 \) then decide\(\text{Min}(i, m)\)

The correctness and worst-case complexity of \( u-P_{\text{min}[k]} \) are stated in Theorem 3. The reader is referred to its proof in the full paper [5] for a precise analysis of the decision conditions. We remark that, roughly speaking, the second condition decides on \( \text{Min}(i, m - 1) \) and not on \( \text{Min}(i, m) \) because the latter value is not guaranteed to persist, while the former value is.

**Theorem 3.** \( u-P_{\text{min}[k]} \) solves uniform \( k \)-set consensus in the crash failure model, and all processes decide by time \( \min\{\lceil t/k \rceil + 1, \lceil t/k \rceil + 2\} \).

As shown by Theorem 3, the protocol \( u-P_{\text{min}[k]} \) meets the worst-case lower bound for uniform \( k \)-set consensus from [15, 1]. We emphasize that \( u-P_{\text{min}[k]} \) strictly dominates all existing uniform \( k \)-set consensus protocols in the literature [8, 14, 16, 26]. Essentially, in each of these protocols, a process remains undecided as long as it discovers at least \( k \) new failures in every round. The fact that \( u-P_{\text{min}[k]} \) is based on hidden capacity and hidden paths, rather than on the number of failures seen, allows runs with much faster stopping times. In particular, there exist runs in which all previous processes decide after \( \lceil t/k \rceil + 1 \) rounds, and in \( u-P_{\text{min}[k]} \) all processes decide by time 2; see Fig. 4 for an example.

At this point, however, we have been unable to resolve the following.

**Conjecture 1.** \( u-P_{\text{min}[k]} \) is an unbeatable uniform \( k \)-set consensus protocol in the crash failure model.

6 We emphasize that the notion of “improvement” captured by the notion of unbeatability studied in this paper is defined in terms of the times at which processes perform their decisions. This is distinct from their halting times, for example (although a process can safely halt at most one round after it decides). Optimizing decision times can come at a cost.
saying that a protocol is worst-case optimal. Our second result is a protocol, $\text{u-P}_{\min[k]}$, for uniform $k$-set consensus that strictly beats all known protocols in the literature [8, 14, 16, 26]; notably, in some executions, processes in our protocol can decide much faster than in those protocols. Whether our uniform $k$-set consensus protocol is unbeatable remains an open problem. Both protocols are efficiently implementable.

We have presented two distinct proofs for the unbeatability of our nonuniform $k$-set consensus protocol $\text{Opt}_{\min[k]}$. Each proof gives a different perspective of the unbeatability of the protocol. The first proof is fully constructive and combinatorial, while the second relies on Sperner’s lemma and is nonconstructive and topological. The topological proof of Lemma 1 is more than just a “trick” to prove the lemma. In a precise sense, the proof shows what a topological analysis of unbeatable protocols is about.

The construction and analysis of both protocols, as well as the unbeatability of $\text{Opt}_{\min[k]}$, crucially depends on a new notion hidden capacity. This notion is a generalization of the notion hidden path introduced in [4], which plays a similar role in the case of (1-set) consensus. Derived from

in communication, for example. Finally, of course, one can compare protocols in terms of more global properties such as average decision times (w.r.t. appropriate distributions etc.). Another point to notice when considering unbeatability is that it is based on comparing the performance of different protocols on the same behaviors of the adversary. While we find this a reasonable thing to do in benign failure models such as crash and omission failures, it may be rather tricky in the presence of malicious (Byzantine) failures.

7. ACKNOWLEDGMENTS

Figs. 1–4 include visual elements designed by Freepik.

Armando Castañeda is supported partially by a project PAPIT-UNAM 1A101015. This research was partially done while Armando Castañeda was at the Department of Computer Science of the Technion, supported by an Aly Kaufman post-doctoral fellowship.

Yanai Gonczarowski is supported by the Adams Fellowship Program of the Israel Academy of Sciences and Humanities; his work is supported by the European Research Council under the European Community’s Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no. [249159], by ISF grants 230/10 and 1435/14 administered by the Israeli Academy of Sciences, and by Israel-USA Binational Science Foundation (BSF) grant number 2014389.

Yoram Moses is the Israel Pollak chair at the Technion; his work was supported in part by ISF grant 1520/11 administered by the Israeli Academy of Sciences.
8. REFERENCES

[1] D. Alistarh, S. Gilbert, R. Guerraoui, and C. Travers. Of choices, failures and asynchrony: The many faces of set agreement. *Algorithmica*, 62(1-2):595–629, 2012.

[2] E. Borowsky and E. Gafni. Generalized FLP impossibility result for t-resilient asynchronous computations. In *Proc. 25th ACM Symp. on Theory of Computing*, pages 91–100, 1993.

[3] A. Castañeda, Y. A. Gonczarowski, and Y. Moses. Brief announcement: Pareto-optimal solutions to consensus and set consensus. In *Proc. 32nd ACM Symp. on Principles of Distributed Computing*, pages 113–115, 2013.

[4] A. Castañeda, Y. A. Gonczarowski, and Y. Moses. Unbeatable consensus. In *Proc. 28th International Symp. on Distributed Computing*, pages 91–106, 2014. Full version available on arXiv.

[5] A. Castañeda, Y. A. Gonczarowski, and Y. Moses. Unbeatable set consensus via topological and combinatorial reasoning. Available on arXiv.org, 2016.

[6] B. Charron-Bost and A. Schiper. Uniform consensus is harder than consensus. *Journal of Algorithms*, 51(1):15–37, 2004.

[7] S. Chaudhuri. Agreement is harder than consensus: Set consensus problems in totally asynchronous systems. In *Proc. 9th ACM Symp. on Principles of Distributed Computing*, pages 311–324, 1990.

[8] S. Chaudhuri, M. Herlihy, N. A. Lynch, and M. R. Tuttle. Tight bounds for k-set agreement. *Journal of the ACM*, 47(5):912–943, 2000.

[9] B. Coan. A communication-efficient canonical form for fault-tolerant distributed protocols. In *Proc. 5th ACM Symp. on Principles of Distributed Computing*, pages 63–72, 1986.

[10] D. Dolev, R. Reischuk, and H. R. Strong. Early stopping in Byzantine agreement. *Journal of the ACM*, 34(7):720–741, 1990.

[11] P. Dutta, R. Guerraoui, and B. Pochon. Tight bounds on early local decisions in uniform consensus. In *Proc. 17th International Symp. on Distributed Computing*, pages 264–278, 2003.

[12] C. Dwork and Y. Moses. Knowledge and common knowledge in a Byzantine environment: crash failures. *Information and Computation*, 88(2):156–186, 1990.

[13] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty processor. *Journal of the ACM*, 32(2):374–382, 1985.

[14] E. Gafni, R. Guerraoui, and B. Pochon. The complexity of early deciding set agreement. *SIAM Journal on Computing*, 40(1):63–78, 2011.

[15] R. Guerraoui, M. Herlihy, and B. Pochon. A topological treatment of early-deciding set-agreement. *Theoretical Computer Science*, 410(6-7):570–580, 2009.

[16] R. Guerraoui and B. Pochon. The complexity of early deciding set agreement: How can topology help? *Electr. Notes Theor. Comput. Sci.*, 230:71–78, 2009.

[17] V. Hadzilacos. On the relationship between the atomic commitment and consensus problems. In *Fault-Tolerant Distributed Computing*, pages 201–208, 1986.

[18] J. Y. Halpern, Y. Moses, and O. Waarts. A characterization of eventual Byzantine agreement. *SIAM Journal on Computing*, 31(3):838–865, 2001.

[19] M. Herlihy, D. N. Kozlov, and S. Rajsbaum. *Distributed Computing Through Combinatorial Topology*. Morgan Kaufmann, 2013.

[20] M. Herlihy, Y. Moses, and M. R. Tuttle. Transforming worst-case optimal solutions for simultaneous tasks into all-case optimal solutions. In *Proc. 30th ACM Symp. on Principles of Distributed Computing*, pages 231–238, 2011.

[21] M. Herlihy, S. Rajsbaum, and M. R. Tuttle. Unifying synchronous and asynchronous message-passing models. In *Proc. 17th ACM Symp. on Principles of Distributed Computing*, pages 133–142, 1998.

[22] M. Herlihy and N. Shavit. The topological structure of asynchronous computability. *Journal of the ACM*, 46(6):858–923, Nov. 1999.

[23] I. Keidar and S. Rajsbaum. A simple proof of the uniform consensus synchronous lower bound. *Information Processing Letters*, 85(1):47–52, 2003.

[24] Y. Moses. Relating knowledge and coordinated action: The knowledge of preconditions principle. In *Proc. 15th Conference on Theoretical Aspects of Rationality and Knowledge*, pages 207–216, 2015.

[25] Y. Moses and M. R. Tuttle. Programming simultaneous actions using common knowledge. *Algorithmica*, 3:121–169, 1988.

[26] P. R. Parvédy, M. Raynal, and C. Travers. Early-stopping k-set agreement in synchronous systems prone to any number of process crashes. In *Proc. 8th International Conference on Parallel Computing Technologies*, pages 49–58, 2005.

[27] M. Raynal. Optimal early stopping uniform consensus in synchronous systems with process omission failures. In *Proc. 16th Annual ACM Symp. on Parallelism in Algorithms and Architectures*, pages 302–310. ACM Press, 2004.

[28] M. E. Saks and F. Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. *SIAM Journal on Computing*, 29(5):1449–1483, 2000.

[29] X. Wang, Y. M. Teo, and J. Cao. A bivalency proof of the lower bound for uniform consensus. *Information Processing Letters*, 96(5):167–174, 2005.