Adaptability and the Usefulness of Hints*

Piotr Berman[†] Juan A. Garay[‡]

Abstract

In this paper we study the problem of designing algorithms in situations where there is some information concerning the typical conditions that are encountered when the respective problem is solved. The basic goal is to assure efficient performance in the typical case, while satisfying the correctness requirements in every case. We introduce *adaptability*, a new measure for the quality of an algorithm, which generalizes competitive analysis and related frameworks. This new notion applies to sequential, parallel and distributed algorithms alike.

In a nutshell, a "hint" function conveys certain information about the environment in which the algorithm operates. Adaptability compares the performance of the algorithm against the "specialist"—an algorithm specifically tuned to the particular hint value. From this perspective, finding that no single algorithm can adapt to all possible hint values is not necessarily a negative result, provided that a family of specialists can be constructed.

Our case studies provide examples of both kinds. We first consider the "ancient" problem of on-line scheduling of jobs in m identical processors, and show that no algorithm can fully adapt to the natural hint of largest job size. In particular, for the cases $m = 2, 3$, we present specialists that beat the general lower bound for on-line makespan.

In the domain of distributed computing, we analyze the Distributed Consensus problem under several hint functions. To fulfill the requirements of one of the cases we consider, we present the first consensus algorithm that is simultaneously optimal in number of processors, early-stopping property (that is, it runs in time proportional to the actual number of faults), and total number of communicated bits. In our new parlance, the algorithm adapts to the number of faults hint with both running time and communication.

Key words: Adaptability, on-line algorithms, job scheduling, distributed algorithms, Byzantine agreement.

---

*A preliminary version of this paper appeared in *Proc. 6th Annual European Symp. on Algorithms–ESA '98*, Venice, Italy, August 1998.

[†] The Pennsylvania State University, Computer Science Department, Pond Lab, University Park 16802, USA. E-mail: `berman@cse.psu.edu`.

[‡] Information Sciences Research Center, Bell Laboratories, 600 Mountain Ave, Murray Hill, NJ 07974, USA. E-mail: `garay@research.bell-labs.com`. URL: `//www.bell-labs.com/user/garay/`.

1 Introduction

In this paper we study the problem of designing algorithms in situations where there is some information concerning the **typical** conditions which are encountered when the respective problem is solved. The basic goal is to assure a correct behavior in *every* case, and to guarantee efficient performance in the *typical* case. We introduce *adaptability*, a new measure for the quality of an algorithm, which generalizes previous ones. In particular, the notion extends the "competitive throughput" formulation for distributed algorithms of Aspnes and Waarts [4]. As illustrated by our case studies, the notion applies to sequential algorithms as well, where it measures how well a given algorithm takes advantage of the special cases which a designer or user may regard as important.

While the formulation of adaptability is somewhat more complex, we feel that it provides a useful tool for algorithm design and evaluation in situations where the traditional worst case analysis is inadequate. In particular, we want to construct a measure of quality that would evaluate as optimal the algorithm which performs well in practice, while pin-pointing the situations where no generally optimal algorithm exists (and consequently that a *specialized* algorithm should be chosen).

The earliest rigourous study of this phenomenon is due to Sleator and Tarjan [23] for the case of search trees. If we build a search tree for n objects, we cannot guarantee to anwer the queries in time better than $\log n$, while a perfectly balanced tree achieves this bound. However, if we knew the distribution of the queries in advance, we could build an optimum tree for that particular distribution, and in extremal cases achieve $O(1)$ average time per query. Therefore, a static search tree, even if optimal with respect to the worst-case performance, may be $\log n$ times worse than a specialized competitor. The question studied by Sleator and Tarjan was: Can a single algorithm compete with any specialized search tree without knowing the distribution of the queries? As we all know, they answered this question in the positive.

In the domain of distributed computing, a similar question of competitiveness arose for consensus protocols [21]. A protocol that tolerates the presence of t faults must, in the worst case, run for $t+1$ rounds [14]. While in ordinary applications very few faults are encountered most of the time, the system designer may need to set the bound t on the maximum tolerated number of faults high as a form of assurance. Therefore, the question was whether we could have the best possible assurance against faults ($\lfloor (n-1)/3 \rfloor$ out of n), and yet have very fast runs in the average cases. Again, algorithms were found that have a maximum tolerance theshold and use $\min(f+2, t+1)$ rounds, where $f \leq t$ is the actual number of faults that occur in a run (e.g., [12, 8]). Such algorithms have been called *early-stopping*.

Both examples above have the following common thread: the *usual* conditions experienced by the algorithm in a given application have some common characteristic. Given a *hint*, i.e., the value of this characteristic, one may provide an algorithm that performs optimally when the hint is true, and which under any conditions performs correctly. We may say that this algorithm is a *specialist*. Given a problem, the question is whether we need a specialist, or whether there exists an optimal *generalist*, i.e., an algorithm that is able to match (up to a constant factor) the specialists' performance for every value of the hint. In the former case, we should investigate what the value of the hint is in order to choose the specialist properly; in the latter, we can rely on the generalist to *adapt* to the conditions that the hint would describe without any additional help.

Before we present our quality measure—the *adaptability* of an algorithm—more formally, we explain why we feel that the existing measures do not serve our stated goals. It is well known that worst-case performance does not capture the adaptive properties of algorithms at all. The "classical" *competitive analysis* [23, 19] is essentially worst-case analysis applied to the case of on-line computations. On the other hand, average-case analysis requires to make a guess, sometimes very questionable, concerning the probability distribution of the request sequence.

This is particularly true in the area of distributed computation, where the enviroment in which an algorithm operates has numerous characterstics, such as the pattern of delays (in asynchronous settings), the pattern of faults, the schedule of the outside inputs, etc. Typically, the task of a distributed algorithm is conceptually simple, and the whole point is to be able to cope with the unpredictability of its enviroment. This unpredictability makes it difficult to define the notion of "average" in a satisfying manner. Therefore, rather than trying to characterize the average behavior, it is more natural to compare the algorithm being evaluated to possible competitors which somehow "know" certain characteristics of the enviroment. There were already several papers analyzing on-line and distributed algorithms in this fashion. In this paper we want to continue this approach and to offer a new concept that would make it easier to compare and to discuss distinct results.

The definition of *competitive latency* of a distributed algorithm of Ajtai *et al.* [2] looks rather close to our approach, but it is unclear whether it yields any more information than worst-case analysis. In particular, they found that their algorithm for the *cooperative collect* problem is better than previous ones by a factor $n^{1/2}/\log^2 n$ using the competitive latency measure, and by the same factor if they use the worst-case work measure. On the other hand, the later work on the same topic by Aspnes and Waarts [4] shows the advantages of the evaluated algorithm that would be missed by worst-case analysis. In terms of our framework, part of the reason is in the different choice of the hint function. One of our objections to their framework is that it delegates to fine print one of the most important aspects of the analysis, thus making the comparison between different results more difficult.

Our definition follows more closely the competitive analysis of paging done by Borodin *et al.* [9]. Their conceptual framework clearly helped in obtaining illuminating results about paging algorithms that could not be obtained through worst-case analysis. In our framework, we would say that they analyzed the role of the hint provided by the *access graph*, in which the pages that can be requested constitute the nodes and the pairs of pages that can be requested consecutively are the edges. The access graph helps to capture the notion of locality of reference. Our objection to this framework is the choice of the optimality measure, which leads to a cumbersome formulation of the results. Borodin *et al.* clearly study the *ratio* of the performance of the evaluated algorithm to the performance of benchmark algorithms (which we call the "specialists"), and they express all their results and conjectures as ratios of two measures. Therefore, we feel that it would be more appropriate to use this ratio as the measure.

Another related work is that of Azar, Broder and Manasse [3], who investigated metrical tasks systems with the set of all possible inputs being partitioned into subsets, each with a specialized algorithm. They proposed a general method for switching between specialized algorithms on the fly, and used the specialized algorithms to define a "generic" algorithm with a competitive ratio that depends on the quality of the specialized algorithms and their number.

Our results. We now summarize the contributions of this paper.

- We introduce *adaptability*, a new notion for evaluating the quality of algorithms which generalizes previous ones. We present the framework formally in Section 2, where we also express the examples mentioned in this section in terms of adaptability to show that they can be presented in an uniform manner. We also show-case the notion in two different domains:
- We consider the "ancient" problem of scheduling jobs in m identical machines [16]. The hint function we consider is the largest size of a job that will be submitted, and we show that no generalist can *fully* adapt (i.e., match the performance of the specialists up to the constant

factors). The proof is constructive, and consists of presenting two specialists for the cases $m = 2, 3$ that beat the general lower bound. These results are presented in Section 3.

- We analyze the Distributed Consensus problem [21] under several natural hint functions. In particular, for the function f—the *actual* number of faults that occur in a run of the algorithm—we provide the first algorithm (generalist) that adapts to f with both running time and communication. Another interesting hint we consider is δ, the *discrepancy* in the initial values of the correct processors. We show that no consensus algorithm can adap to δ with rounds, and that it suffices to choose one of just two specialized algorithms. These results are presented in Section 4.

2 Adaptability

In general, we consider the problem of achieving some goal in enviroments from a set $\mathcal{E}$.[1] An algorithm is *valid* if it achieves this goal in every enviroment $E \in \mathcal{E}$. A *hint* function h maps $\mathcal{E}$ into some arbitrary set H. For every valid algorithm A and enviroment $E \in \mathcal{E}$, a run of A in E has some cost, denoted $\mathrm{cost}(A, E)$. Our basic objective is to minimize this cost.[2]

Given a hint value v, we define the worst-case cost of the algorithm restricted to v as $\mathrm{cost}_v(A) \stackrel{\mathrm{def}}{=} \max_{E \in h^{-1}(v)} \mathrm{cost}(A, E)$. Then the *adaptability of A to a hint value $h(E) = v$* is

$$a_A(h = v) \stackrel{\mathrm{def}}{=} \max_{\text{valid algorithm } B} \frac{\mathrm{cost}_v(A)}{\mathrm{cost}_v(B)} .$$

A *specialist* for hint value v is an algorithm $S_{h=v}$ that is "uniquely adapted to v," i.e., $a_{S_v}(h = v) = O(1)$. The *adaptability of A to hint h* is

$$a_A(h) \stackrel{\mathrm{def}}{=} \max_{v \in H} a_A(h = v) .$$

Intuitively, the hint h characterizes enviroments that are either typical, or critical (for example, we may need to minimize the response time of an algorithm only in critical cases). Knowing the hint v, we might be able to find a specialized algorithm S_v which minimizes the worst-case cost in the set of enviroments indicated by v. (Note that because the specialized algorithm must be correct even when $h(E) \neq v$, the hint is not an input to the algorithm in the conventional sense.) We evaluate how algorithm A compares against such specialists. If the adaptability of A with respect to h is $O(1)$, we say that *A adapts to h.* (Sometimes we will refer to such an algorithm as the *generalist.*) We now express the examples mentioned in the Introduction in terms of adaptability, to show that they can be presented in a uniform manner.

2.1 The expressive power of adaptability

We start with the results and conjectures of Borodin *et al.* [9] regarding paging in a two-level storage system. We define $\mathcal{E}_{k,N}$ as the sequences of page requests that can be served by an off-line algorithm with k pages of fast memory and N faults. The hint is the *access graph*, where nodes represent the pages of virtual memory and edges join the pages that are requested one after another somewhere

[1] We do not make any distinctions between the input to a problem, or request sequence, and the remaining conditions that might affect a run of the algorithm (for example, the schedule of processes in an asynchronous distributed computing application—cf. [2, 4]). An environment $E \in \mathcal{E}$ describes everything that affects the run.

[2] The notion of adaptability can be readily extended to cases where the objective is to maximize the profit accrued by the algorithm, rather than to minimize the cost it incurs. This is done by replacing the maxima by minima, and reversing the fraction. In this paper we focus on cost measures only.

in the sequence. Borodin *et al.* define two functions, $c_{A,k}(G)$, the competitiveness of algorithm A restricted to traversals of graph G, and $c_k(G)$, the competitiveness of the specialist. In those terms, $a_A(G) = \frac{c_{A,k}(G)}{c_k(G)}$ and $a_A(\text{access graph}) = \max_G \ a_A(G)$. All of their results and conjectures have an explicit formulation in our language. For example, they prove that $a_{LRU,4}(\text{access graph}) = 1.5$; and they pose as an open problem whether "there is a paging algorithm that adapts to the access graph."

Adaptability can also be used in conjunction with the probabilistic analysis of algorithms. Karlin *et al.* [20] studied the paging problem under the assumption that the pages in the request sequence form the states of a Markov chain M (to capture the locality of reference more precisely than Borodin *et al.*). In our framework, M would form the environment and the hint value, and $\text{cost}(A, M) = \lim_{T\to\infty} S_{R,T}/T$, where $S_{R,T}$ is the expected number of page faults that algorithm A incurs during the first T page requests generated by M. The negative results of Karlin *et al.* are that almost all "natural" algorithms (e.g., any marking algorithm) do not adapt to this hint function. The positive results show a constructive family of specialists, one for each Markov chain; because Markov chains can be learned in the limit [22], this leads to the existence of a generalist algorithm that adapts to the initially-unknown Markov chain.

In the distributed computing message-passing domain, an early-stopping consensus protocol is an algorithm that adapts to the actual number of faults with (respect to) the number of rounds (i.e., the number of rounds is the cost function, while the actual number of faults is the hint). In shared-memory systems, the work of Ajtai *et al.* [2] deals with the adaptability of algorithms for the cooperative collect problem, with the cost measured by the total number of reads to the shared registers, and the hint being the delay schedule of the processors. They found that the adaptability of their algorithm is $O(n^{1/2}\log^2 n)$. The subsequent work of Aspnes and Waarts [4] discussed the same problem, but using instead the profit function of *throughput* and the hint being the "input." The adaptability of the same algorithm was found to be $O(n^{1/4}\log^2 n)$.

3 Adaptability of an Ancient Scheduling Problem

One of the oldest problems studied in the area of scheduling is the case when a sequence of jobs $(1, \ldots, n)$ arrive to m identical machines, where job j has size s_j. As soon as a job arrives, the scheduler assigns it to a machine (later, a job is never reassigned to another machine). The goal is to minimize the *makespan*, i.e., the maximum sum of job sizes assigned to a single machine (we say that such a sum is the *load* of a machine). A very simple algorithm called *List*—always schedule on the currently least loaded machine—achieves a competitive ratio of $2 - 1/m$ [16]. Because Graham published this result in 1966, in recent papers the problem was dubbed "an ancient scheduling problem," which we will abbreviate as ASP. Although in recent years this algorithm was improved several times, [6, 18, 1], *List* is provably optimal for $m = 2, 3$[13].

The hint function that we will consider is the *largest size of a job* that will be submitted. In real-life situations we often have at least an approximated knowledge of this nature, which makes this hint quite natural. Because even the simple algorithm achieves a constant competitive ratio, the question of adaptability is trivial if we discuss it in the asymptotic sense. However, one may ask whether a generalist algorithm can *fully* adapt, i.e., match the performance of specialist algorithms up to the constant factors. This question is answered negatively by the following theorem.

Theorem 1 *Let the cost of a run of an ASP algorithm be the* makespan, *let M be the maximum job size that is submitted in a given run, and let* m *be the number of machines. Then for every*

ASP algorithm A

$$a_A(M) \geq \frac{3/2}{4/3} \quad \textit{for } m = 2 \quad \textit{and} \quad a_A(M) \geq \frac{5/3}{1.475} \quad \textit{for } m = 3.$$

The M-hint (maximum job size) will be used by our specialized algorithms in the following manner. First, we can rescale the job sizes so that $M = 1$. Next, because we know that a job of size 1 will eventually arrive, we may schedule such a job as the first step of our algorithm. Later, when the first job of size 1 arrives, we can place it on the machine on which it is already scheduled. Thus we can analyze our algorithm under the assumption that the first job has size 1 and no larger job will ever arrive.

Before we proceed we will formulate a lemma which will be used several times and which holds for $m = 2, 3$.

Lemma 1 *Assume that at some point in time the scheduling algorithm has the maximum load of a machine equal to $l \geq 1$. Assume also that from this point onwards the algorithm behaves exactly like algorithm List. Then starting from the first time when the makespan increases again, the ratio of the obtained makespan to the optimum one is at most $m(l+1)/(ml+1)$.*

The proof of the lemma is entirely based on the observation that the optimum makespan is at least as large as the average of the machine loads. The details will be provided in the full version.

The algorithm for two machines is to simply run *List*. Right at the beginning the obtained makespan equals 1 and as long as it stays unchanged, it is optimal because it is equal to the largest job size. After it increases, by Lemma 1 the ratio of the obtained makespan to the optimal one is at most $2(1{+}1)/(2{+}1) = 4/3$. Therefore we can use the modified *List* as the optimum algorithm B in the definition of adaptability function $a_A(M)$. Note that the ratio of 4/3 is optimum, because we can face the scheduler with the choice of the following two sequences of the job sizes: (1, 1/2, 1/2) and (1, 1/2, 1/2, 1). Either the scheduler will obtain the makespan of 3/2 when 1 is possible, or 2 when 3/2 is possible.

The same algorithm and argument can be applied to three machines where it yields the ratio of $3(1{+}1)/(3{+}1) = 3/2$. However, this ratio is not optimal (but already better than the ratio of 5/3 that is achievable without the hint). One can obtain a ratio of $r = (\sqrt{97}-1)/6 \approx 1.475$ using a somewhat more complicated algorithm and much more complicated argument. This algorithm has two phases. In the first, which we may call first fit, machines have fixed numbers 1, 2 and 3. When a new job of size s arrives, we schedule it on the first machine that had load l satisfying $l + s \leq r$. Once it is not possible anymore, we enter the second phase, when we simply run *List*.

To prove the ratio we start with two observations. In the first fit phase ratio r is assured, as the makespan is bounded by r and the largest job offers the lower bound of 1. Suppose that the ratio of obtained makespan to the optimum is at most r after scheduling the first job in the *List* phase. Then we obtain the highest load of at least 1.47; by Lemma 1 this assures that after the change of the makespan the ratio will never exceed 1.24. Therefore it suffices to consider what happens when the first job is scheduled after the first fit phase.

For that part of the reasoning we define $l_0 = (\sqrt{97}+3)/12$. We consider two cases according to the highest machine load machine that occurs during the first fit phase. If this load is higher (or equal) than l_0, the claim about ratio r will follow from Lemma 1, because $3(l_0+1)/(3l_0+1) = r$. If on the other hand this load is lower than l_0, than the first makespan in the *List* phase is less than $l_0 + 1$. On the other hand, all jobs on machines 2 and 3 exceed $r - l_0$; in an optimal schedule one of these jobs must be scheduled on the same machine as a job of size 1; this leads to a lower bound of $1 + r - l_0$ and the ratio estimate $(2 + l_0)/(1 + r - l_0) = r$. Our choice of the values of r and l_0 comes from solving the two equations that were used in this proof.

4 Adaptability of Distributed Consensus

As another case study of the notion of adaptability, we also analyze the Distributed Consensus problem (a.k.a. Byzantine agreement, reliable broadcast, consensus) [21] under several natural hint functions. In particular, for the function f—the actual number of faults that occur in a run of the algorithm—we provide an algorithm that adapts with both running time and communication. Giving better algorithms for fundamental primitives is bound to immediately have an impact on other algorithms that build on them; in the case of Distributed Consensus: atomic broadcast, multicast, consistency maintenance, etc. We first define the problem formally. Given are n processors, at most t of which might be arbitrarily faulty. Each processor i has an initial value $v_i \in \{0,1\}$. Required is an algorithm with the following properties:

Decision: Every non-faulty processor eventually decides on a value from $\{0,1\}$.

Agreement: The non-faulty processors all decide on the same value.

Validity: If the initial values v_i of all nonfaulty processors are identical, then every nonfaulty processor decides on v_i.

We assume the standard model for this problem, in which processors may fail in arbitrarily malicious ways [21]. Each processor can communicate directly with every other processor via a reliable point-to-point channel. The processors are synchronous, and their communication proceeds in synchronous rounds: in a round, a processor can send one message to each of the other processors, and all messages are received in the round in which they are sent. Finally, it must be the case that $n > 3t$ for the problem to have a solution [21].

An environment in which a consensus protocol runs has several characteristics, including, in particular, the sets of correct and faulty processors—P_C and P_F, respectively—and the sets of correct processors with initial values 0 and 1—P_0 and P_1, respectively. We consider the following hint functions for this problem, which are of practical relevance:

- The actual number of faults that occur in a run of the algorithm, $f \stackrel{\text{def}}{=} \#P_F$;
- the *discrepancy* in the initial values $\delta \stackrel{\text{def}}{=} \#P_1 - \#P_0$;
- the absolute value of the discrepancy, $|\delta|$;
- a subset of processors containing correct processors, i.e., $S \subseteq P$ such that $\#S = k$ and $S \cap P_C \neq \emptyset$.

We also consider two cost functions: the number of communication *rounds*, and the total number of *bits* transmitted. Regarding hint f, "early stopping" techniques (see, e.g., [12, 8]) enable the design of consensus algorithms with a running time which is proportional to f, (i.e., algorithms that adapt to f with the number of rounds). In Section 4.1 we will present the first consensus algorithm for arbitrary failures that adapts to f with number of rounds and bits *simultaneously.*

The remaining hints are treated in Sections 4.2 and 4.3; here we provide a brief explanation of those results. Hint δ captures situations, common in practice, in which either one input value is distinctly more frequent, or when the performance is critical only if one of the two values predominates (typically, in such situations one decision value amounts to performing an action, such as "abort," and the other to refrain from it). We show that a consensus algorithm cannot adapt to δ with rounds, and that it suffices to choose one of just two specialized algorithms. Hint $|\delta|$ captures situations when in a typical case correct processors start with almost unanimous preferences. We show that if the total number of processors exceeds $4t$, then adaptability with rounds is possible;

in other cases, the problem remains open. Even then this hint is valuable, in the sense that certain design choices do not affect the worst case, but preclude adaptability to $|\delta|$. The hint corresponds to the cases where some processors are more reliable than others. We show that adaptibility is not possible, but that one can construct a single family of specialists.

4.1 The actual number of faults hint

In this section we consider a hint function that specifies the actual number of faults f ($f \leq t$) that occur in a run of the algorithm, and present an algorithm that adapts to f with cost measures number of rounds and total bit transfer simultaneously.

Recall that a lower bound of $t+1$ rounds of communication has been established for this problem in the worst case [14]. Thus, overcoming potential faults is expensive, even though, in practice, they are rarely observed. This led researchers to explore ways in which the price of fault tolerance may be reduced. Dolev, Reischuk, and Strong [12] demonstrated that by permitting processors to stop at different times (instead of simultaneously), early-stopping consensus algorithms are possible that run in time proportional to f (specifically, $\min(f+2, t+1)$ rounds). That is, in our new parlance,

Theorem 2 [12, 8] *Let the cost of a run of a consensus algorithm be the* number of rounds, *and f the actual number of faulty processors. Then there exists an algorithm A such that $a_A(f) = O(1)$ (i.e., A adapts to f).*

Hadzilacos and Halpern [17] concentrated on runs that are failure-free, and presented consensus algorithms with optimal number of messages for these runs. Which brings us to the other cost measure: the communication costs of such protocols, given by the total number of transmitted bits. Dolev and Reischuk [11] showed a lower bound of $\Omega(nt)$ for this measure, a bound that holds for *every* run of a consensus algorithm. Moreover, algorithms have been developed that match this bound [7, 10]. Intuitively, f is useless for this measure: any bit-optimal algorithm adapts to this hint—in the trivial sense. This immediately yields

Theorem 3 *Let the cost measure of a run of a consensus algorithm be the* total number of transmitted bits, *and f be the actual number of faulty processors. Then there exists an algorithm A such that $a_A(f) = O(1)$.*

Given Theorems 2 and 3, it is natural to ask whether a consensus algorithm can adapt to f with both cost measures simultaneously. In the next section we answer this question in the positive.

Simultaneous adaptability. As mentioned above, existing early-stopping consensus algorithms adapt to f with rounds, but not with communication, as a total of $\Omega(n^2t)$ bits are transmitted. On the other hand, there exist algorithms with $O(nt)$ communication, which satisfy Theorem 3 [7, 10]. The running time of such algorithms, however, is proportional to t, the maximum number of tolerable faults. We now present a new consensus algorithm that is able to adapt to f with both cost measures, i.e., it runs for $O(f)$ rounds while communicating a total of $O(nt)$ bits. We call this algorithm SA, for *Simultaneously-Adaptive.*

The new algorithm is based on the *phase king* paradigm for consensus of [8]. In such a paradigm, the computation consists of a series of phases, each consisting of possibly several rounds; in each phase, a distinguished processor(s) takes charge of the computation. SA is defined recursively.[3] For simplicity, we assume in this section that the total number of processors $n = 3t + 1$.

[3]Indeed, given the communication bound that we are trying to achieve, message exchanges must be restricted to hierarchically defined subsets of processors. However, we remark that special precaution must be taken to prevent a single faulty processor from delaying termination for a number of rounds equal to the depth of the recursion.

Recursive instance $\mathsf{SA}(k, P_k, t_k)$ is executed by a set P_k of at least $3t_k + 1$ processors, which learn of their initial "preferences" (initial values for this instance) before communication round k; the top instance is $\mathsf{SA}(1, P, t)$, where $P = P_1$ is the original set of processors. Assuming that at most t_k of these processors are faulty, the protocol assures that before round $k + r(t_k)$ all the correct processors will have the same final preference for this instance. Here r is a fixed function, $r(m) = O(m)$, that gives the (worst-case) number of rounds of the protocol. Additionally, if the correct processors are unanimous in their initial preferences, their final preferences will be the same.

Moreover, the protocol has the following early-stopping property: If the number of faulty processors in P_k is $f_k \leq t_k$, then the correct processors will know their final preferences by the end of round $k + r_{es}(f_k)$, where $r_{es}(m) = O(m)$ is a fixed function that estimates the number of rounds when an *early* decision (and *stopping*) can be guaranteed.

Before we turn to a more detailed description of the algorithm, we announce the net result:

Theorem 4 *Consensus algorithm* SA *adapts to the actual number of faults f with number of communication rounds and total communicated bits, while tolerating the maximum possible number of faults.*

Description of algorithm SA. A sketch of the "code" is given in Figure 1 in Appendix A.

We assume variable V contains a tuple $< v_p, \pi >$, where $v_p \in \{0, 1\}$ is processor p's current preference (initially, v_p is p's initial value), and predicate $\pi \in \{0, 1, 2\}$ represents the strenght of that preference (initially, $\pi = 0$). In every case, to define protocol instance $\mathsf{SA}(k, P_k, t_k)$ we select some $p_k \in P_k$ to be the *king of P_k*.

If $t_k = 0$, then we may assume that $P_k = \{p_k\}$. The conditional correctness assumes that p_k is correct. Obviously, a satisfactory protocol consists of p_k computing its final preference as equal to the initial one. Therefore $r(0) = 0$. If $t_k = 1$, then we may assume that P_k has four elements; in this case two rounds of the "Exponential Information Gathering" (EIG) algorithm of [5][4] define a satistactory protocol. As a result, $r(1) = 2$ and $r_{es}(0) \geq 2$.

For $t_k \geq 2$ we define the protocol recursively. Let a and b be equal or almost equal integers such that $t_k = a + b + 2$. Let P_{k+5} and $P_{k+r(a)+8}$ be two disjoint subsets of $P_k - \{p_k\}$ with $3a + 1$ and $3b + 1$ members, respectively; we will refer to these sets as the *first committe* and the *second committee.* The protocol consists of **three phases**: the king's phase and the phases of the two committees, and describes the actions in nine *logical* rounds. The actions of a single logical round would take place in a single communication round if the processors did not apply early stopping rules; because they do, different processors of P_k can execute them at different times. Indeed, the execution of the protocol effectively becomes asynchronous, as at a given point different processors may be executing different levels of the recursion. We show below that this can be handled without any message indexing, and thus asynchrony does not add to the communication overhead.

The first three logical rounds form the king's phase. In round 1, processors communicate their initial preference. It is possible, however, that a processor has announced its initial preference as part of some earlier message; this assumes that this processor concluded that the correct processors of P_k would be unanimous in their initial preferences. In any case, this action takes place in communication round k at the latest.

As soon as a processor knows that in round 1 $2t_k + 1$ processors of P_k have announced their preferences to be v, it sets its next preference to v and sends $(v, 1)$ as its message of round 2; the meaning of this message is "I **strongly** prefer v". Otherwise, if communication round k was

[4] EIG is basically a full information protocol in which each processor broadcasts its value, and then every processor echoes the values they received in the previous round.

completed and this condition is still not satisfied, the processor sends $(0,0)$, meaning "I *weakly* prefer 0". At the latest, this action takes place in communication round $k+1$.

As soon as a processor knows that in round 2 $2t_k+1$ processors of P_k have announced that they strongly prefer v, it sets its next preference to v and sends $(v,1)$ as its message of round 3. Otherwise, if communication round $k+1$ is completed, the processor checks if it received at least t_k+1 strong preferences for some value v. If yes, it sends $(v,0)$ as a message or round 3, otherwise it sends $(0,0)$. At the latest, this action takes place in communication round $k+2$.

As soon as a processor knows that in round 3 $2t_k+1$ processors of P_k have announced that they strongly prefer v, it sends $(v,2)$ as its message of round 4. This message means "I **very strongly** prefer v"; in other words: "I will prefer v, with the maximum strength possible, through the duration of this protocol and all its recursive calls; I will send no more messages in this protocol." Otherwise it computes its initial preference for the phase of the first committee when communication round $k+2$ is completed. If it learns that t_k+1 processors strongly prefer v, it will prefer v as well. If this holds neither for $v=0$ nor for $v=1$, then the processor takes its preference from the message received from the king, i.e., from p_k.

The phase of the first committee is similar in spirit, but there are several differences. The logical rounds 4, 5, and 6 take place at the latest in communication rounds $k+3$, $k+4$, and $k+5$, respectively. The role of the king is played by the first committee running instance $\mathsf{SA}(k+5, P_{k+5}, t_{k+5}=a)$. The initial preferences for this run are the preferences contained in the messages of round 5. The subsequent communications of this run take place within P_{k+5}, and as soon as a member of P_{k+5} knows its final preference for this run, it sends it to the members of $P_k - P_{k+5}$. Members of P_{k+5} know the preference of the first committee as soon as this run concludes, i.e., before communication round $k+5+r(t_k+5)$; members of $P_k - P_{k+5}$ know this preference when they receive $t_{k+5}+1$ announcements from P_{k+5} about the same value. This should happen at the latest when communication round $k+5+r(a)$ is completed.

The phase of the second committee is almost identical to that of the first committee. The logical rounds 7, 8, and 9 take place at the latest in communication rounds $k+6+r(a)$, $k+7+r(a)$ and $k+8+r(a)$, respectively. The final preferences are computed at the latest when communication round $k+8+r(a)+r(b)$ is completed. Because this is the last phase, processors do not send messages about their very strong preferences (these messages are used to execute an early stop, so they are useless at the very end).

Analysis and correctness of algorithm SA. We first evaluate the communication cost incurred by algorithm SA.

Lemma 2 *In a run of* SA*, a total of $O(nt)$ bits are communicated.*

The lemma follows since the total number of bits sent in SA obviously satisfies the recurrence $B(n) < O(n^2) + 2B(\frac{n}{2})$; hence it is $O(nt)$ (since $n = O(t)$).

We now show its (conditional) correctness and the early stopping property. If P_k contains more than t_k faulty processors, the conditional correctness is vacuously true. Therefore we assume that there are at most t_k such processors. We first prove several lemmas under this assumption.

Lemma 3 *Assume that in a run of* $\mathsf{SA}(k, P_k, t_k)$ *all correct processors in P_k have the same initial preference, say $v \in \{0,1\}$. Then, regardless of the behavior of the king in phase 1, they exit the algorithm in logical round 4 with a very strong preference for v.*

Proof: Under the assumption, all correct processors of P_k end logical round 1 having a strong preference for v (at least $2t_k+1$ copies of v collected), and logical round 2 results in all of them having a strong preference for v again ($2t_k+1$ copies of $< v, 1 >$ collected). Therefore in the third logical round they collect $2t_k+1$ copies of $< v, 1 >$ again, and hence send $< v, 2 >$ in round 4 (to annouce their very strong preference) and quit without reading the messages sent in this round. $\square$

Lemma 4 *Assume that in a run of* $\mathsf{SA}(k, P_k, t_k)$ *all correct processors in* P_k *have the same preference, say* $v \in \{0,1\}$*, at the beginning of the first committee's phase. Then, whithout waiting for the messages from the first committee, they exit the algorithm in logical round* 7 *with a very strong preference for* v*. This happens at the latest in communication round* $k+6$*.*

The reasoning is strictly analogous to that of Lemma 3. Moreover, because in logical rounds 4, 5, and 6 the next preference is computed by a "$2t_k+1$" rule, the correct processors do not wait for the respective communication rounds. The proof of the following Lemma is also analogous to that of Lemma 3.

Lemma 5 *Assume that in a run of* $\mathsf{SA}(k, P_k, t_k)$ *all correct processors in* P_k *have the same preference, say* $v \in \{0,1\}$*, at the beginning of the second committee's phase. Then, whithout waiting for the messages from the second committee, they exit the algorithm in logical round* 9 *with a final preference for* v*.*

Lemma 6 *Assume that in a run of* $\mathsf{SA}(k, P_k, t_k)$ p_k*, the king of* P_k *is correct. Then the correct processors are unanimous at the beginning of the first committee's phase and exit in round* 7 *at the latest.*

Proof: Assume first that some correct processor has a strong preference for $v \in \{0,1\}$ after logical round 2. This means that in round 2 it collected at least $2t_k+1$ copies of $< v, 1 >$. Therefore, since $|P_k| \geq 3t_k+1$, every correct processor collected at least t_k+1 copies of $< v, 1 >$, and therefore it has at least a weak preference for v. Note that it is impossible in round 2 to collect t_k+1 copies of both $< 0, 1 >$ and $< 1, 1 >$. Assuming that t_k+1 copies of $< 0, 1 >$ were collected, we know that at least one was sent by a correct processor, which means it has seen at most t_k copies of 1 among messages of logical round 1, which in turn means that every correct processor has seen at most $2t_k$ copies of 1. Hence, no correct processor could have a strong preference for 1 after round 1, and so $< 1, 1 >$ messages are sent in round 2 by faulty processors only. Thus, the "t_k+1" rule works unambiguously. As a result, all correct processors have at least a weak preference for v—the king included—and in round 3 they either choose v as the next preference without checking the value of the king, or they adopt it from the king.

The remaining case is that no correct processor has a strong preference after logical round 2. Then all of them will adopt the preference from the king.

The final conclusion is implied by Lemma 4. $\square$

Lemma 7 *Assume that in a run of* $\mathsf{SA}(k, P_k, t_k)$ $f_k \leq t_k$ *of the processors are faulty. Then all the correct processors of* P_k *will reach the same final decision, equal to their initial prefence if they started unanimously, before communication round* $k+r(t_k)+1$ *and before round* $k+6f_k+1$*.*

Proof: If $t_k < 2$, the claim follows trivially. If the correct procesors start the run unanimously, the claim follows from Lemma 3. If p_k, the king of P_k is correct, the claim follows from Lemma 6. In the remainder of the proof we will assume that the king is faulty.

Recall that P_k contains the king and two committees with $3a+1$ and $3b+1$ elements, so that $t_k = 2+a+b$. Let the number of faults in these committees (denoted P_{k+5} and $P_{k+r(a)+8}$) be f_{k+5} and $f_{k+r(a)+8}$, respectively, where $f_k = 1 + f_{k+5} + f_{k+r(a)+8}$. Assume first that $f_{k+5} \leq a$. Then the first committee delivers a correct message by round $k+5+6f_{k+5}+7+1$ at the latest, and Lemma 5 guarantees that three communication rounds later all correct processors will exit with the same preference. Because $f_k \leq 1 + f_{k+5}$, this exit time satisfies the requirement that it is $k+6f_k+7$ at the latest.

It remains to consider the case when p_k is faulty and $f_{k+5} > a$. Then we know that $f_{k+r(a)+8} \leq b$, so that the second committee is correct, and that the second committee starts its recusive copy of the algorithm by communication round $k + r(a) + 8$ at the latest. The result follows from a reasoning very similar to the one above. □

We remark that we can improve the algorithm and analysis to obtain smaller constant factors in the number of bits and the number of rounds; we choose not to for ease of presentation's sake. Lemmas 2 and 7 yield

Theorem 4 *Consensus algorithm* SA *adapts to the actual number of faults with number of communication rounds and total communicated bits, while tolerating the maximum possible number of faults.*

4.2 The discrepancy hint

The δ hint—the discrepancy between the number of initial preferences for 0 and 1—provides the simplest natural situation in which no consensus algorithm can adapt and consequently, if the hint is available, a respective specialist should be used. We start with the following

Observation 1 *Any early stopping consensus algorithm* ES *terminates in a constant number of rounds if* $\#P_0 = 0$ *or* $\#P_1 = 0$.

Consider now an algorithm derived from ES by preceeding it with the following round of communication (code for processor i):

> **broadcast** v_i
> **if** more than t 0's received **then** $v_i \leftarrow 0$

According to the definition of δ, if $\delta \leq 0$ then $\#P_0 \geq \#P_1$, and therefore $\#P_0 > t$ (recall that $n > 3t$), which implies that the initial round of the new algorithm results in $\#P_1 = 0$. Consequently, we have converted ES into specialist $S_{\delta \leq 0}$, an algorithm that by Observation 1 terminates after a constant number of rounds for every value of $\delta \leq 0$. Similarly, by replacing 0's with 1's, we can obtain $S_{\delta \geq 0}$, a specialist which terminates after a constant number of rounds whenever $\delta \geq 0$. Thus, for every value of δ we can select a specialist that terminates in a constant number of rounds.

By the same token, no single consensus algorithm can adapt to this hint function. Otherwise, such a generalist would terminate in every environment in $O(1)$ rounds, violating the well-known $t+1$ lower bound for consensus of [14]. We thus obtain

Theorem 5 *Let* $\mathrm{cost}(A, E)$ *of consensus algorithm* A *be the number of rounds. Then* $a_A(\delta) = \Omega(t)$, *and for every hint value* v, *a respective specialist* $S_{\delta=v}$ *can be selected from* $\{S_{\delta \leq 0}, S_{\delta \geq 0}\}$.

The situation with hint $|\delta|$, the absolute value of the discrepancy, is more complicated. For the case $n > 4t$, two specialists with similar properties to the ones above exist, thus making it impossible for a single generalist to adapt. On the other hand, for $n \leq 4t$ we conjecture that such a generalist algorithm exist, and we show a necessary condition which such a generalist must satisfy. More specifically, consider a consensus algorithm derived from ES by preceeding it with the following round:

broadcast v_i
if more than $n/2$ 0's received **then** $v_i \leftarrow 0$ **else** $v_i \leftarrow 1$

One can see that if $|\delta| > t$, then this initial round of the new algorithm leads the correct processors to unanimity, because either $\#P_0 > n/2$ or $\#P_1 > n/2$. Thus, Observation 1 applies, and this algorithm can be used as $S_{|\delta|>t}$. On the other hand, if $|\delta| \le n - 3t$, then we can conclude that both $\#P_0 > t$ and $\#P_1 > t$ hold true, and algorithm $S_{\delta \ge 0}$ defined above for hint δ would terminate in a constant number of rounds. Therefore, $S_{\delta \ge 0}$ can be used as $S_{|\delta| \le n-2t}$.

If $n > 4t$, then for every value of $|\delta|$ one of these two specialists can assure termination in a constant number of rounds. On the other hand, by the same argument as before, no single generalist can exist.

Theorem 6 *Let* $\mathrm{cost}(A, E)$ *of consensus algorithm* A *be the number of rounds and* $n > 4t$. *Then* $a_A(|\delta|) = \Omega(t)$, *and for every hint value* v, *a respective specialist* $S_{|\delta|=v}$ *can be selected from* $\{S_{|\delta|>t}, S_{|\delta| \le n-3t}\}$.

When $3t < n \le 4t$, hint $|\delta|$ can lead to three situations: We may select $S_{|\delta|>t}$ and be sure of termination in a constant number of rounds; we may select $S_{|\delta| \le n-2t}$ with the same result; or none of these specialists may be helpful. We conjecture that in the latter case, no algorithm can adapt to $|\delta|$. Specifically:

Conjecture 1 *Theorem 6 holds for* $n > 3t$. *For* $n - 3t \le |\delta| = v \le t$ *the specialist* $S_{|\delta|=v}$ *can be any round-optimal consensus algorithm (i.e., these values of hint* $|\delta|$ *are "useless").*

In fact, we go further and propose that the lower bound proof of [14] can be appropriately extended to show that no specialist can improve upon $S_{|\delta|>t}$, regardless of the actual value of $|\delta|$, namely

Conjecture 2 $a_{S_{|\delta|>t}}(|\delta|) = O(1)$.

4.3 Correct processors

Suppose that we are given the hint that a processor, say, p, is the most reliable. We obtain specialist A_p by "crowning" p first in ESPK. The analysis of ESPK shows that if p is indeed correct, then all correct processors will terminate after two phases. Again, it is clear that no single generalist can adapt to this hint function.

Similarly, if the hint gives a small subset S of size k of processors containing one correct processor, we construct specialist A_S by modifying ESPK in a similar fashion, which guarantees termination in $3(k + 1)$ rounds.

5 Conclusions

In this paper we have introduced adaptability, a new tool for evaluating the quality of algorithms. This concept allows us to present numerous results in on-line and distributed algorithms in a uniform framework. Moreover, our case studies show that for some applications a single algorithm is not sufficient, and one has to choose the algorithm that is most suitable for the typical/critical instances. The concepts of adaptability, hints, and specialists and generalists provide a proper language to describe in a clear fashion a toolkit of results that are best suited for such situations.

Acknowledgments

We are thankful to Jim Aspnes and Amotz Bar-Noy for many useful insights and suggestions.

References

[1] S. Albers, Better bounds for online scheduling. *Proc. 29th Annual ACM Symp. on the Theory of Computing*, pp. 130-139, El Paso, TX, May 1997.

[2] M. Ajtai, J. Aspnes, C. Dwork, and O.Waarts. A Theory of Competitive Analysis for Distributed Algorithms. *Proc. 33rd IEEE Symp. on the Foundations of Computer Science,* pp. 401-411, November 1994.

[3] Y. Azar, A. Broder and M. Manasse. On-line choice of on-line algorithms. *Proc. 4th Annual ACM/SIAM Symp. on Discrete Algorithms*, pp. 432-440, 1993.

[4] J. Aspnes and O. Waarts. Modular Competitiveness for Distributed Algorithms. *Proc. 28th Annual ACM Symp. on the Theory of Computing,* pp. 237-246, Philadelphia, PA, May 1996.

[5] A. Bar-Noy, D. Dolev, C. Dwork and H.R. Strong. Shifting gears: changing algorithms on the fly to expedite Byzantine Agreement. *Proc. 6th ACM Symposium on the Principles of Distributed Computing,*, pp. 42-51, 1987.

[6] Y. Bartal, A. Fiat, H. Karloff and Y. Rabani, New algorithms for the ancient scheduling problem. *Journal of Computer and System Sciences,* 51:359-366, 1995.

[7] P. Berman, J. Garay, and K. Perry. Bit Optimal Distributed Consensus. In *Computer Science Research* (ed. R. Yaeza-Bates and U. Manber), Plenum Publishing Corporation, NY, NY, pp. 313-322, 1992.

[8] P. Berman, J. Garay, and K. Perry. Optimal Early Stopping in Distributed Consensus. *Proc. 6th International Workshop on Distributed Algorithms*, LNCS (647), Springer-Verlag, pp. 221-237, Haifa, Israel, November 1992.

[9] A. Borodin, S. Irani, P. Raghavan, and B. Schieber. Competitive Paging with Locality of Reference. *Proc. 23rd Annual ACM Symp. on the Theory of Computing,* pp. 249-259, New Orleans, Louisiana, May 1991.

[10] B. Coan and J. Welch, Modular Construction of an Efficient 1-Bit Byzantine Agreement Protocol. *Mathematical Systems Theory*, special issue on Fault-Tolerant Distributed Algorithms (ed. H.R. Strong), Vol. 26, No. 1 (1993).

[11] D. Dolev and R. Reischuk, Bounds of Information Exchange for Byzantine Agreement. *JACM*, Vol. 32, No. 1, pp. 191-204, 1985.

[12] D. Dolev, R. Reischuk and H.R. Strong. Early Stopping in Byzantine Agreement. *JACM*, Vol. 37, No. 4 (1990), pp. 720-741.

[13] U. Faigle, W. Kern and G. Turan, On the performance of online algorithms for particular problems. *Acta Cybernetica*, 9:107-119, 1989.

[14] M. J. Fischer and N. A. Lynch. A lower bound for the time to assure interactive consistency. *Inf. Proc. Letters*, 14 (1982), pp. 183–186.

[15] Z. Galil, A. Mayer and M. Yung. Resolving Message Complexity of Byzantine Agreement and Beyond. *Proc. 336th IEEE Symp. on the Foundations of Computer Science,* pp. 724-732, Milwaukee, WI, October 1995.

[16] R.L. Graham. Bounds on multiprocessing anomalies. *SIAM Journal of Applied Mathematics*, 17:263–269, 1969.

[17] V. Hadzilacos and J. Halpern. Message-Optimal Protocols for Byzantine Agreement. *Proc. 10th Annual ACM Symp. on the Principles of Distributed Computing*, pp. 309-324, Montreal, Canada, August 1991.

[18] D.R. Karger, S.J. Phillips and E. Torng, A better algorithm for an ancient scheduling problem, *Journal of ALgorithms*, 20:400-430, 1996.

[19] A. Karlin, M. Manasse, L. Rudolph, and D. Sleator. Competitive Snoopy Caching. *Algorithmica*, 3(1):70-119, 1988.

[20] A. Karlin, S. Phillips, and P. Raghavan. Markov Paging. *Proc. 33rd Annual IEEE Symp. on Foundations of Computer Science,* pp. 208-217, 1992.

[21] L. Lamport, R.E. Shostak and M. Pease. The Byzantine generals problem. *ACM Trans. Prog. Lang. and Systems*, 4:3 (1982), pp. 382–401.

[22] S. Rudich. Inferring the structure of a Markov chain from its output. *Proc. 26th Annual IEEE Symp. on Foundations of Computer Science,* pp. 321-326, 1985.

[23] D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 32:652–686, 1985.

A Figures

```
Algorithm SA(k, P_k, t_k)
            (* global V contains the initial preference *)
            if t_k = 1
                run EIG among P_k
            else    (* t_k = a + b + 2 *)
                (* king's phase *)
1, k        {   broadcast V to P_k and remaining members of parent set
                count[i] ← # received < i, · >'s, for i = 0, 1
2, k+1          if (for some i, count[i] > 2t_k)
                    V ← < i, 1 >
                else V ← < 0, 0 >
                broadcast V to P_k
                count[i] ← # received < i, 1 >'s, for i = 0, 1
3, k+2          if (for some i, count[i] > 2t_k)
                    V ← < i, 1 >
                elseif (for some i, count[i] > t_k)
                    V ← < i, 0 >
                else V ← < 0, 0 >
                broadcast V to P_k
                count[i] ← # received < i, 1 >'s, for i = 0, 1
                if (for some i, count[i] > 2t_k) then          (* phase decisions *)
                {   stubborn ← true; V ← < i, 2 >              (* final decision reached *)
                } elseif (for some i, count[i] > t_k)
                    V ← < i, 1 >
                else V ← value received from p_k

                (* first committee's phase *)
4, k+3          broadcast V to P_k
                if stubborn
                    return
                count[i] ← # received < i, · >'s, for i = 0, 1
5, k+4          if (for some i, count[i] > 2t_k)
                    V ← < i, 1 >
                else V ← < 0, 0 >
                broadcast V to P_k
                count[i] ← # received < i, 1 >'s, for i = 0, 1
6, k+5          if (for some i, count[i] > 2t_k)
                    V ← < i, 1 >
                elseif (for some i, count[i] > t_k)
                    V ← < i, 0 >
                else V ← < 0, 0 >
                if p ∈ P_{k+5}
                    run SA(k+5, P_{k+5}, t_{k+5} = a)
                else broadcast V to P_k
                same phase and final decisions as in logical round 3;
                the value of the king is the outcome of SA(k+5, P_{k+5}, a)
                (i.e., > t_{k+5} + 1 equal values received from P_{k+5})

                (* second committee's phase *)
                Logical rounds 7,8,9 are identical to 4,5,6, substituting P_k − P_{k+5}
                for P_{k+5} and SA(k+7+r(a), P_k − P_{k+5}, b) for SA(k+5, P_{k+5}, a), resp.
            }
```

Figure 1: Adaptive consensus algorithm SA; sketch of code for processor p. Logical rounds and actual rounds are indicated on the left.