T D C C

M M

Department of Computer Science, University of Cyprus
75 Kallipoleos St., CY-1678 Nicosia, Cyprus
mavronic@cs.ucy.ac.cy

EIGHT OPEN PROBLEMS IN DISTRIBUTED COMPUTING

James Aspnes
Dept. of Computer Science
Yale University
New Haven, CT 06520-8285
aspnes@cs.yale.edu

Costas Busch
Dept. of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180
buschc@cs.rpi.edu

Shlomi Dolev
Dept. of Computer Science
Ben-Gurion University of the Negev
Beer-Sheva, Israel 84105
dolev@cs.bgu.ac.il

Panagiota Fatourou
Dept. of Computer Science
University of Ioannina
45110 Ioannina, Greece
faturu@cs.uoi.gr

Chryssis Georgiou
Dept. of Computer Science
University of Cyprus
CY-1678 Nicosia, Cyprus
chryssis@cs.ucy.ac.cy

Alex Shvartsman
Dept. of Computer Science and Engineering
University of Connecticut
Storrs, CT 06269
aas@cse.uconn.edu

Bulletin of the EATCS no 90, pp. 109–126, October 2006
© European Association for Theoretical Computer Science

Paul Spirakis
RACTI
265 00 Rio, Patras
spirakis@cti.gr

Roger Wattenhofer
Computer Engineering and Networks Lab.
ETH Zurich
8092 Zurich, Switzerland
wattenhofer@tik.ee.ethz.ch

Abstract

Distributed Computing Theory continues to be one of the most active research fields in Theoretical Computer Science today. Besides its foundational topics (such as consensus and synchronization), it is currently being enriched with many new topics inspired from modern technological advances (e.g., the Internet). In this note, we present eight open problems in Distributed Computing Theory that span a wide range of topics – both classical and modern.

1 Wait-Free Consensus

A *consensus protocol* is a distributed algorithm where n processes collectively arrive at a common decision value starting from individual process inputs. It must satisfy *agreement* (all processes decide on the same value), *validity* (the decision value is an input to some process), and *termination* (all processes eventually decide). A protocol in an asynchronous shared-memory system is *wait-free* if each process terminates in a finite number of its own steps regardless of scheduling. From the FLP impossibility result [31, 54], wait-free consensus is impossible. However, it becomes possible using randomization with the termination condition relaxed to hold with probability 1.

The **open question** that then arises is the complexity of solving consensus, measured by the expected number of register operations carried out by all processes (*total work*) or by any one process (*per-process work*).

This complexity depends strongly on assumptions about the power of the adversary scheduler. For an *adaptive adversary* that chooses the next process to run based on total knowledge of the current state of the system, the best known protocol using only atomic read-write registers takes $O(n^2 \log n)$ expected total work [14]. If counters supporting increment, decrement, and read operations are available, this drops to $O(n^2)$ expected total work [4]. No faster protocol is known using any objects that can be built from atomic registers, and there is a lower bound of $\Omega(n^2/\log^2 n)$ that holds even given powerful tools like unit-cost snapshots [6].

Closing the gap between the upper and lower bounds is interesting because all known polynomial-time wait-free consensus protocols are based on collecting enough random votes that one standard deviation in the total is larger than the $n-1$ votes that can be "hidden" by the adversary by selectively stopping processes, and it is not hard to show that simple variants on voting cannot yield subquadratic protocols. A faster protocol would thus require a significantly new approach. Conversely, an $\Omega(n^2)$ lower bound would show that voting is optimal.

With a weaker adversary that cannot observe coin flips that have not yet been made public, consensus can be solved in $O(\log n)$ work per process using multi-writer registers [11]. There is no corresponding non-trivial lower bound. It would be interesting to see if an $\Omega(\log n)$ lower bound could be proved for multi-writer registers or even with strong objects like unit-cost snapshots. Closing the gap in both models would show whether the cost of weak-adversary consensus arises from fundamental limitations of grouping local coin-flips together or merely from the weakness of atomic registers.

2 Oblivious Routing

A typical distributed computing environment consists of several processing units which communicate through some underlying multi-hop network. The network is usually modeled after a graph, possibly weighted, where nodes represent the processing units and the edges the communication links. The nodes communicate by exchanging messages in the form of packets. *Routing* is the task of selecting the paths that the packets will follow in the network. Ideally the selected paths should have small *congestion*, that is, the maximum number of paths crossing any edge should be small, and the paths should have small *stretch*, that is, the ratio between the selected path and the respective shortest path should be as small as possible.

Oblivious routing is a type of distributed routing suitable for dynamic packet arrivals. In oblivious routing, the path for a newly injected packet is selected in a way that it is not affected by the path choices of the other packets in the network. Räcke [66] gives an existential result that shows that for any network there exists an oblivious routing algorithm with congestion within factor $\log^3 n$ from that of the optimal off-line centralized algorithm, where n is the number of nodes. This oblivious algorithm constructs a path by choosing a logarithmic number of random intermediate nodes in the network. Azar *et al.* [13] showed that the probabilities for the random intermediate nodes can be computed a priori in polynomial time.

Even though congestion is a fundamental metric for the performance of routing algorithms, stretch is important too, since it represents the extra delay of the packets when there is no congestion. Ideally, stretch should be a constant. So far, the main research on oblivious routing algorithms has focused on optimizing the congestion while ignoring the stretch. For example, a packet may have destination to a neighbor node of the source and still the path chosen by an oblivious algorithm may be as long as the number of nodes in the network.

An interesting **open problem** is to examine the circumstances in which congestion and stretch can be optimized simultaneously.

There is a simple counter-example network that shows that in general the two metrics are orthogonal to each other: take an adjacent pair of nodes u, v and $\Theta(\sqrt{n})$ disjoint paths of length $\Theta(\sqrt{n})$ between u and v. For packets travelling from u to v, any routing algorithm that minimizes congestion has to use all the paths, however, in this way some packets follow long paths, giving high stretch. Nevertheless, there are special cases of interesting networks where congestion and dilation can be minimized simultaneously. For example, in grids [15], and in networks of uniformly distributed nodes in convex-like areas [16], the congestion is within a poly-logarithmic factor from optimal and stretch is constant.

A second interesting **open problem** is to find other classes of networks where the congestion and stretch are minimized simultaneously.

Possible candidates for such networks could be for example bounded-growth networks, or networks whose nodes are uniformly distributed in closed polygons, which describe interesting cases of wireless networks. Another interesting open problem is to find classes of networks in which oblivious routing gives $C + D$ close to the off-line optimal, where C is the congestion and D is the maximum path length. Such a result will have immediate consequences in packet scheduling algorithms since it is known from [52] that it is feasible to deliver the packets in time proportional to $C + D$.

3 Stability of Continuous Consensus

Consensus is a fundamental task in distributed computing, it allows to reduce a distributed task to a centralized task by agreeing on the system state, the inputs and (hence) the common transition. One shot consensus cannot be self-stabilizing [22] since it can terminate with disagreeing outputs. On the other hand, on-going consensus may stabilize to eventually ensure that when a new consensus instance is invoked the safety property for the output of this instance is correct [23].

In the scope of on-going (self-stabilizing) consensus task one may consider the sequence of inputs and outputs of instances [20, 24] and require stability of outputs as long as the inputs allow such a stability. For example, when one consensus instance output has been 1, and the next instance has 1 as a possible output value, then 1 should be preferred. Namely, we would like to minimize the number of times the output is changed.

The **open problem** is, to determine the most stable (consensus) function to use, given flexibility in deciding on the output of the system.

Namely, given a particular sequences of input changes, choose the function that changes output as least as possible, assuming that the function from the inputs to the common output is only restricted to ensure that the output has a value equal to at least $t + 1$ inputs. For example, if the system can remember (in memory) the last output, the system may stick to the output as long as it can: say the system includes five processors, at most two of which maybe faulty, i.e., $t = 2$. In case the inputs are 1, 1, 1, 1, 1 the system must output 1, then if the inputs are repeatedly changed to 1, 1, 1, 1, 0 and then to, say, 1, 1, 0, 1, 0 the system may stay with a stable output 1, but once the inputs are changed to, say, 1, 0, 0, 1, 0 the system output must be changed to 0.

The case of a geodesic path of input changes, where each input can be changed at most once is considered in [20, 24]. The upper bound for the memoryless binary input case in [20] is $2t + 1$ (where the majority of the first $2t + 1$ inputs defines the output). Multi-valued consensus extends the case of binary-valued consensus, allowing the inputs (and the output) to be a non-necessarily binary value.

An upper bound for the number of output changes for a memoryless symmetric system (where the function has the same output regardless of the position of inputs in the input vector) is presented in [20]. The upper bound is a factor of approximately 2 away from a corresponding lower bound shown using concepts from Algebraic Topology.

Closing this gap, as well as considering non geodesic input path changes that are useful to separate the performance and to evaluate consensus functions are **open questions**.

Also in the case of multi-valued consensus one may only require an output value that is within the range of values of the correct values, further exploring functions is also open, and we believe that it is fruitful field of research with application to several domains, including sensor activated devices, stable aggregation of distributed information and alike.

4 Complexity of Implementing Atomic Snapshots

A *snapshot* object consists of m components (shared variables), each storing a value. Processes can perform UPDATE operations to change the value of each individual component, and SCANS, each of which returns a consistent view of the contents of all the components. These operations can be performed simultaneously by different processes. Snapshots have been widely used to facilitate the design and verification of numerous distributed algorithms because they provide an immediate solution to the fundamental problem of calculating consistent views of shared variables; this happens while these variables may be concurrently updated by other processes.

A snapshot *implementation* from registers uses shared registers to simulate the snapshot components and provides algorithms for SCAN and UPDATE. Assuming that processes may fail by crashing, an implementation is *wait-free* if each non-faulty process terminates executing a SCAN/UPDATE within a finite number of its own steps. An implementation is *linearizable* if (roughly speaking) the execution of a SCAN or an UPDATE operation in any execution of the implementation appears to take effect instantaneously.

Since snapshots have several applications, the design of efficient snapshot implementations is crucial. The *time complexity* of SCAN (UPDATE) of an implementation is the maximum number of steps executed by a process to perform a SCAN (UPDATE, respectively) in any execution of the implementation. The *time complexity* of the implementation is the maximum of the time complexities of its SCAN and UPDATE. Despite the numerous work that has been performed on designing efficient snapshot implementations (see [30] for a survey), their time complexity is not yet fully understood. Some implementations use a small number of registers but they have large time complexity while others employ more registers to achieve better time complexity.

It is known [27] that at least m registers are required to implement an m-component snapshot. An implementation that uses only m registers is provided in [1, 28]. Its time complexity is $O(mn)$ for both SCAN and UPDATE, where n is the number of processes in the system. A lower bound of $\Omega(mn)$ on the time complexity of SCAN for space-optimal implementations (that use only m registers), proved in [28], shows that this implementation is optimal. An implementation that uses n registers and has time complexity $O(n)$ for SCAN and $O(n \log n)$ for UPDATE (or vice versa) is provided by combining results in [2, 10, 42]. The fastest known implementation [8] has time complexity $O(n)$ for both SCAN and UPDATE and uses $O(n^2)$ registers. Another implementation with time complexity $O(n)$ which, however, uses an unbounded number of registers can be obtained by combining results in [2, 41]. Lower bounds on the space-time tradeoff are provided in [29], where it is proved that the time complexity of SCAN in any implementation that uses a fixed number of registers grows without bound as *n* increases.

Bridging the gap between the lower bounds provided in [29] and the best known upper bounds (discussed above) is a challenging **open problem**.

Even less is known for the time complexity of UPDATE. A lower bound of $\Omega(m)$ on the time complexity of UPDATE is proved in [7]. This lower bound extends a similar result presented in [5] for the weaker version of a *single-writer* snapshot (where each component can be updated by only one process associated to the component). Since the best known snapshot implementation [8] has time complexity $O(n)$ for UPDATE, it is unknown if this lower bound is optimal.

Proving better lower bounds for the time complexity of UPDATE or designing more efficient algorithms (in terms of the UPDATE time complexity) is an intriguing **open problem**.

The identification of tradeoffs between the number of registers used in an implementation, the time complexity of SCAN, and the time complexity of UPDATE is another interesting **open problem**.

The lower bounds proved in [28, 29] hold for deterministic algorithms and they can be possibly beaten by employing randomization. Some randomized implementations for the weaker version of single-writer snapshot objects are presented in [9]. Finding efficient randomized implementations for multi-writer snapshot objects remains a challenging **open problem**.

5 Pure Nash Equilibria in Selfish Routing

In modern non-cooperative networks, such as the Internet, participants, acting selfishly, wish to efficiently route their traffic from some source to some destination with the least possible delay. In such environments, *Nash Equilibria* [62, 63] represent steady states of the system where no user may profit by unilaterally changing its strategy.

Koutsoupias and Padadimitriou [47], formulated the study of selfish routing in non-cooperative networks by casting the problem as a non-cooperative game, known in the literature as the KP-model; *n* selfish users wish to route their unsplitable traffic onto *m* parallel links from a source to a destination. Each link has a certain capacity representing the rate at which the link processes traffic, and users have complete knowledge of the system's parameters such as the link capacities and the traffic of other users. Also, users choose how to route their traffic based on a common payoff function, which essentially captures the delay to be experienced on each link. However, modern non-cooperative systems present incomplete information on various aspects of their behavior. For example, it is often the case, that network users have incomplete information regarding the link capacities. Such uncertainty may be caused if the network links are complex paths created by routers which are constructed differently on separate occasions according to the presence of congestion or link failures.

Gairing *et al.* [32] were the first to consider an extension of the KP-model with incomplete information. Their model considers a game of parallel links with incomplete information on the traffics of the users. The payoff functions employed by the users take into account probabilistic information on the user traffics. The authors show (along with other interesting results) that their model always admits a Pure Nash Equilibrium and propose a polynomial-time algorithm for computing such equilibria for some special cases.

In [38] an extension of the KP-model was introduced, where the network links may present a number of different capacities and each user's uncertainty about the capacity of the links, called *belief*, is modeled via a probability distribution over all the possibilities. It is assumed that the users may have different sources of information regarding the network and, therefore, take their probability distributions to be distinct from one another. This gives rise to a model with user-specific payoff functions, where each user uses its distinct probability distribution to take decisions as to how to route its traffic. In particular, the model is an instance of weighted congestion games with user-specific functions studied by Milchtaich [59].

The authors of [38], among other problems, studied the existence of Pure Nash Equilibria in this new model; they proposed Polynomial-time algorithms for computing pure Nash equilibria for some special cases and they showed that the negative results of [59], for the non-existence of pure Nash equilibria in the case of three users, do not apply to their model.

The problem of existence of pure Nash Equilibria for this new model in the general case is a non-trivial problem; as of this writing, it remains **open**.

Given the non-existence result for weighted congestion games with user specific payoff-functions [59], a natural step is to disprove the existence of pure Nash Equilibria for the new model described in [38].

It is conjectured that the model introduced in [38] always admits a Pure Nash equilibrium in general. Proving or disproving this conjecture is an interesting **open challenge**.

Work for answering this question has been carried out in various directions. In [33] it was shown that the game introduced in [38] is not an *ordinal potential game*, since it has been shown that the state space of an instance of the game contains a cycle. Therefore, potential functions [60], a powerful method for proving existence of Nash Equilibria, cannot be used for this model. Further attempts by the authors of [38], including applying graph-theoretic methods and inductive arguments have not been successful. The arguments end up failing mainly due to the arbitrary relation between the different user beliefs on the capacity of the network links.

Typically, simple counter-examples to the existence of pure Nash Equilibria considering a small number of resource (links) and users are used for such purposes (for example, in [59], the counter-example involves 3 users and 3 resources). This appears not to be the case for the new model: in [38] was shown that for the case of three users (and arbitrary number of links) pure Nash Equilibria always exist; also simulations ran on numerous instances of the model (dealing with small number of users and links) suggest the existence of pure NE.

6 Adverse Cooperative Computing

The problem of cooperatively performing a collection of tasks in a decentralized setting where the computing medium is subject to adversarial perturbations is one of the fundamental problems in distributed computing. Such perturbations can be caused by processor failures, unpredictable delays, and communication breakdowns. To develop efficient solutions for computation problems ranging from distributed search such as SETI@home to parallel simulation and GRID computing, it is important to understand efficiency trade-offs characterizing the ability of p processors to cooperate on t independent tasks in the presence of adversity. This basic problem of cooperation has been studied in a variety of models, including shared-memory [3, 40, 43, 45, 58], message-passing [18, 19, 21, 26, 34, 48], in partitionable networks [25, 37, 39], and also in the settings with limited communication, e.g., [36, 57, 65]. Developing efficient algorithms solving such task-performing problems in adversarial settings has proven to be difficult.

Here we tackle the problem of distributed cooperation in deterministic shared-memory settings where the processors are subject to arbitrary failures and delays. Kanellakis and Shvartsman [43] introduced and studied an abstraction of this problem, called Write-All, formulated in terms of p processors writing to t distinct memory locations in the presence of an adaptive adversary that introduces dynamic failures or delays. Here writing to a memory location models an independent task that can be performed by a single processor in constant time. The efficiency of algorithms in such settings is measured in terms of *work* that accounts for all steps taken by the processors in solving the problem. The upper bound for Write-All with synchronous crash-prone processors was shown to be $O(t + p\log^2 t/\log\log t)$, where $p \leq t$, and at least one processor is non-faulty. The algorithm exhibiting this bound has optimal work of $O(t)$ when $p \leq t\log\log t/\log^2 t$. However, Kedem, Palem, Raghunathan, and Spirakis [45] showed that when $p = t$, the work lower bound for Write-All is $\Omega(t\log t)$, thus no optimal algorithm for Write-All exists for the full range of processors ($p = t$). Although a small gap of $\log t/\log\log t$ remains between the upper and lower bounds, the problem can be considered substantially solved for synchronous processors.

Solutions for the Write-All problem become significantly more challenging when asynchrony is introduced. The most efficient deterministic asynchronous algorithm known for Write-All is the elegant algorithm of Anderson and Woll [3] that has work $O(t \cdot p^\varepsilon)$ for $p \leq t$ and any $\varepsilon > 0$. The strongest corresponding lower bound, due to Buss, Kanellakis, Ragde, and Shvartsman [17], is $\Omega(t + p\log p)$, and it holds even if no processor crashes. Note that in complexity-theoretic terms, the relative gap between these bounds on work is very large (i.e., polynomial in p, being p^ε for $p = t$), since the lower bound is only a logarithm away from linear work. Given that this gap is now 15 years old, and that this problem continues to be of interest, it appears that narrowing this gap is extremely challenging.

Thus we formulate our first, two-pronged, **open problem** as follows: (a) can a stronger than $\Omega(t\log t)$ lower bound on work be shown for asynchronus Write-All problem, and/or (b) is there an algorithm for asynchronous processors that solves the problem with work asymptotically less than $O(t^{1+\varepsilon})$ for $p = t$?

Next observe that an optimal algorithm for Write-All must have work $\Theta(t)$, however the lower bounds on work of $\Omega(t + p\log p)$ make optimality out of reach when $p = \Omega(t)$. Also note that the algorithm [3] has work complexity $\omega(t)$ for all but a trivial number p of processors. The quest then is to obtain work-optimal solutions for this problem using the largest possible, and non-trivial compared to t, number of processors p in order to maximize the parallelism of the solution. Recently Malewicz [56] presented the first qualitative advancement in the search for optimal work complexity by exhibiting an algorithm that has work $\Theta(t)$ using a non-trivial number g of processors, where $g = \sqrt[4]{t/\log t}$. Using different techniques, Kowalski and Shvartsman [49] exhibited an algorithm that has work complexity of $O(t + p^{2+\varepsilon})$, achieving optimality for a larger range of processors, specifically for $p = O(t^{1/(2+\varepsilon)})$.

Our second and final **open problem** is as follows: Is it possible to solve the asynchronous Write-All problem with optimal work $O(t)$ using the number of processors $p = t^\delta$ for $\delta > 1/2$?

Summing up, we presented the Write-All problem that abstracts the distributed cooperation problem in the presence of adversity. Despite substantial research, there is a dearth of efficient deterministic shared-memory algorithms for Write-All with asynchronous crash-prone processors.

The most challenging **open problems** in this area deal with closing the gap between the lower and upper bounds on work, and with the development of work-optimal algorithms that use the largest possible number of processors in order to achieve high speedup in solving the problem of distributed cooperation.

7 Distributed Approximations

Initiated by Papadimitriou and Yannakakis [64], the distributed approximation of linear programs has attracted the interests of researchers for some time. Most of the past efforts considered the special class of packing and its dual (covering) linear programs. Note that many hard problems (e.g. dominating set, coloring etc.) can be cast in the form of Integer Linear Programs (ILPs) and their distributed complexity of a good approximation is, up to now, a major open issue.

It is fair to assume a setting of a network with classical message passing capability, in which a node can send a message of size $O(\log n)$ bits to each neighbor in the net, in each communication step. Here n is the network size. We can also assume that each network node has a distinct id of size $O(\log n)$ bits. This is a synchronous communication model where the computation is assumed to advance in (global) rounds. Imagine then a general ILP setting, where, for example, there are n "producing" nodes and m "accepting" nodes. In general m is less than n. Each producer, call her i, has to derive an integer x_i (this can be negative. In that case the producer demands x_i units). For each "accepting" node j, when an x_i arrives to it, it has an associated cost (or benefit) $a_i^j * x_i$. For each accepting node j, the sum of all $a_i^j * x_i$, $(i = 1 \dots n)$, must be at most an integer quantity b_j. Here a_j^i, b_j are integers. The x_i's produced have each a cost (or benefit) $c_i * x_i$ where c_i is an integer. Now, the whole system must minimize the sum of all $c_i * x_i$, $(i = 1 \dots n)$. Or, at least approximate this minimum. The reader can recognize that this is the general case of an ILP.

The distributed complexity (i.e. number of rounds to achieve a good approximation) of the general integer-linear programs is a major **open problem**.

Note that the "accepting" nodes can elect a leader and then she can get all the coefficients needed to solve the problem locally. But, the restriction in the message size, leads to an awful number of rounds, e.g. around n. We want here a small number of rounds (constant number would be fine). Note that we do not assume that the a^i_j form a metric. They can also be arbitrarily large.

Facility location is an example of a non-positive linear integer program that is not a covering or packing one. Only very recently, the works [61, 35] made some progress in the distributed approximation of the facility location problem. In the facility location problem, the network is a (usually complete but this does not help much) bipartite graph, where two node sets, C and F share edges between them. Here C is the set of clients and F is the set of facilities. Each facility i has a non-negative opening cost f_i. The connection cost between facility i and client j is an integer c^j_i. Let y_i, x^j_i be zero/one variables where $y_i = 1$ indicates that facility i is open and $x^j_i = 1$ indicates that client j is connected to facility i. The system has to minimize the sum of all opening and connection costs. Note that any good solution that works with the relaxed linear problem first, must round the non-integer solutions. Even this (e.g. randomized rounding) has to be well done in a distributed way.

An interesting variation of such problems is a selfish distributed optimization situation. Let me motivate this by a Stackelberg game: Here each node i wants to send flow x_i to a destination node j. The total flow sums to (say) a value r. Some of the nodes (belonging to a subset L) are not selfish but they agree to work together (e.g. under an elected leader). The rest route their flows selfishly to avoid big delays. But the nodes in L can put their flows in such a way so that the Nash Equilibrium reached by the other nodes is very close to an optimal flow routing (e.g. of min total latency).

Finding distributed solutions to the selfish optimization problem described above, remains an **open problem**

For centralized solutions, one can see [44]. In fact, [44] shows that the leader (i.e., the centralized equivalent to L) can put its flow in such a way so that the Nash equilibrium reached by the other selfish flows is indeed the optimal, provided that the leader controls a sizeable portion of the overall flow. For recent developments on distributed approximations to problems related to linear and integer programming, we refer the reader to [51].

8 Sensor Networks: Locality for Geometric Graphs

Wireless sensor networks currently exhibit an incredible research momentum. Computer scientists and engineers from all flavors are embracing the area. Sensor networks are explored by researchers from hardware technology to operating systems, from antenna design to middleware, from graph theory to computational geometry. The distributed algorithms community should join this big interdisciplinary party.

In the last twenty years, so-called *local* algorithms have been a thriving theoretical research subject. In a *k-local algorithm*, for some parameter k, each node can communicate at most k times with its neighbors. Hence, even in synchronous operation nodes can at most gather information about their k-neighborhood. Early work for this model includes some of the most wonderful results in distributed computing, such as Luby's randomized independent set algorithm [55], sparse partitions by Awerbuch and Peleg [12], or Linial's $\Omega(\log n)$ lower bound [53].

There have been recent advances on both upper [51] and lower [50] bounds; however, many basic questions (e.g. a deterministic local construction of a maximal independent set) are still **open** (see also Section 7).

Until recently this theory was a bit *l'art pour l'art*. Sensor networks may be a first real-world application domain for local algorithms. Due to their wireless nature, the links of a sensor network are often unstable, in other words, the network is dynamic. In such an environment it is often impossible to run a centralized algorithm, as the network topology which serves as an input for an algorithm is usually different from the topology after running the algorithm. Local algorithms on the other hand are able to compromise approximation quality (efficacy) for communication time (efficiency) in order to keep up with the network dynamics.

Unfortunately, local algorithms are not exactly tailored for sensor networks. Apart from various other modeling issues [67], local algorithms are often developed for general graphs; in sensor networks, however, geometry comes into play as the distribution of nodes in space and the propagation range of wireless links usually adhere to geometric constraints. Several models inspired by both graph theory and geometry are possible; in a recent survey [67], a few such models are presented.

So far, very little is known about local algorithms for geometric graphs. To give a specific example, even for a simple model known as *unit disk graph*, the local complexity of typical coordination tasks (e.g., computing a dominating set) is an **open problem**. In fact, to the best of our knowledge, the currently best algorithm for this problem on unit disk graphs remains an algorithm for general graphs [51].

References

[1] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt and N. Shavit, "Atomic Snapshots of Shared Memory", *Journal of the ACM*, Vol 40, No 4, pp. 873–890, September 1993.

[2] J. H. Anderson, "Multi-Writer Composite Registers", *Distributed Computing*, Vol. 7, No 4, pp. 175–195, April 1994.

[3] R. J. Anderson and H. Woll, "Algorithms for the Certified Write-All Problem", *SIAM Journal on Computing*, Vol. 26, No. 5, pp. 1277–1283, October 1997.

[4] J. Aspnes, "Time- and Space-Efficient Randomized Consensus", *Journal of Algorithms*, Vol. 14, No. 2, pp. 414-431, May 1993.

[5] A. Israeli and A. Shirazi, "The Time Complexity of Updating Snapshot Memories", *Information Processing Letters*, Vol. 65, No. 1, pp. 33–40, January 1998.

[6] J. Aspnes, "Lower Bounds for Distributed Coin-Flipping and Randomized Consensus", *Journal of the ACM*, Vol. 45, No. 3, pp. 415-450, May 1998.

[7] H. Attiya, F. Ellen and P. Fatourou, "The Complexity of Updating Multi-Writer Snapshot Objects", *Proceedings of the 8th International Conference on Distributed Computing and Networking*, to appear, December 2006.

[8] H. Attiya and A. Fouren, "Adaptive and Efficient Algorithms for Lattice Agreement and Renaming", *SIAM Journal on Computing*, Vol. 31, No. 2, pp. 642–664, October 2001.

[9] H. Attiya, M. Herlihy and O. Rachman, "Atomic Snapshots Using Lattice Agreement", *Distributed Computing*, Vol. 8, No. 3, pp. 121–132, March 1995.

[10] H. Attiya and O. Rachman, "Atomic Snapshots in $O(n \log n)$ Operations", *SIAM Journal on Computing*, Vol. 27, No. 2, pp. 319–340, April 1998.

[11] Y. Aumann, "Efficient Asynchronous Consensus with the Weak Adversary Scheduler", *Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing*, pp. 209-218, August 1997.

[12] B. Awerbuch and D. Peleg, "Sparse Partitions", *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, Vol. 2, pp. 503–513, October 1990.

[13] Y. Azar, E. Cohen, A. Fiat, H. Kaplan and H. Racke, "Optimal Oblivious Routing in Polynomial Time", *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pp. 383–388, June 2003.

[14] G. Bracha and O. Rachman, "Randomized Consensus in Expected $O(n^2 \log n)$ Operations", *Proceedings of the 5th International Workshop on Distributed Algorithms*, pp. 143-150, October 1991.

[15] C. Busch, M. Ismail and J. Xi, "Optimal Oblivious Path Selection on the Mesh", *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, pp. 82–91, April 2005.

[16] C. Busch, M. Ismail and J. Xi, "Oblivious Routing on Geometric Networks", *Proceedings of the 17th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pp. 316–324, July 2005.

[17] J. Buss, P. Kanellakis, P. Ragde and A. Shvartsman, "Parallel Algorithms with Processor Failures and Delays", *Journal of Algorithms*, Vol. 20, No. 1, pp. 45–86, January 1996.

[18] B. Chlebus, R. De Prisco and A. Shvartsman, "Performing Tasks on Restartable Message-Passing Processors", *Distributed Computing*, Vol. 14, No. 1, pp. 49–64, January 2001.

[19] B. Chlebus, L. Gąsieniec, D. Kowalski and A. Shvartsman, "Bounding Work *and* Communication in Robust Cooperative Computation", *Proceedings of the 16th International Symposium on Distributed Computing*, pp. 295–310, October 2002.

[20] L. Davidovitch, S. Dolev and S. Rajsbaum, "Consensus Continue? Stability of Multi-Valued Continuous Consensus!", *Proceedings of the 6th Workshop on Geometric and Topological Methods in Concurrency and Distributed Computing*, pp. 21-24, October 2004.

[21] R. De Prisco, A. Mayer and M. Yung, "Time-Optimal Message-Efficient Work Performance in the Presence of Faults", *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing*, pp. 161–172, August 1994.

[22] S. Dolev, *Self-Stabilization*, MIT Press, 2000.

[23] S. Dolev, R. Kat and E. Schiller, "When Consensus Meets Self-Stabilization, Self-Stabilizing Failure Detector, Consensus and Replicated State-Machine", Technical Report, Department of Computer Science, Ben-Gurion University of the Negev, 2006.

[24] S. Dolev and S. Rajsbaum, "Stability of Long-Lived Consensus", *Journal of Computer and System Sciences*, Vol. 67, No. 1, pp. 26-45, August 2003.

[25] S. Dolev, R. Segala and A. Shvartsman, "Dynamic Load Balancing with Group Communication", *Proceedings of the 6th International Colloquium on Structural Information and Communication Complexity*, pp. 111–125, July 1999.

[26] C. Dwork, J. Halpern and O. Waarts, "Performing Work Efficiently in the Presence of Faults", *SIAM Journal on Computing*, Vol. 27, No. 5, pp. 1457–1491, October 1998.

[27] P. Fatourou, F. Fich and E. Ruppert, "Space-Optimal Multi-Writer Snapshot Objects are Slow", *Proceedings of the 21st Annual ACM Symposium on Principles of Distributed Computing*, pp. 13–20, July 2002.

[28] P. Fatourou, F. Fich and E. Ruppert, "A Tight Time Lower Bound for Space-Optimal Implementations of Multi-Writer Snapshots", *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pp. 259–268, June 2003.

[29] P. Fatourou, F. Fich and E. Ruppert, "Time-Space Tradeoffs for Implementations of Snapshots", *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, pp. 169-178, May 2006.

[30] F. Fich, "How Hard is it To Take a Snapshot?", *Proceedings of the 31st Annual Conference on Current Trends in Theory and Practice of Informatics,* Vol. 3381, pp. 27–35, January 2005.

[31] M. J. Fischer, N. A. Lynch and M. S. Paterson, "Impossibility of Distributed Consensus with One Faulty Process", *Journal of the ACM*, Vol. 32, No. 3, pp. 374-382, April 1985.

[32] M. Gairing, B. Monien and K. Tiemann, "Selfish Routing with Incomplete Information", *Proceedings of the 17th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pp. 203–212, July 2005.

[33] M. Gairing, B. Monien and K. Tiemann, "Routing (Un-)Splittable Flow in Games with Player-Specific Linear Latency Functions", *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming*, Vol. 4051, pp. 501-512, July 2006.

[34] Z. Galil, A. Mayer and M. Yung, "Resolving Message Complexity of Byzantine Agreement and Beyond", *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science*, pp. 724–733, October 1995.

[35] J. Gehweiler, C. Lammersen and C. Sohler, "A Distributed $O(1)$-Approximation Algorithm for the Uniform Facility Location Problem", *Proceedings of the 18th Annual ACM Symposium on Parallelism in Algorithms and Architectures,* pp. 237-243, July 2006.

[36] S. Georgiades, M. Mavronicolas and P. Spirakis, "Optimal, Distributed Decision-Making: The Case of no Communication", *Proceedings of the 12th International Symposium on Fundamentals of Computation Theory*, Vol. 1684, pp. 293–303, August/September 1999.

[37] C. Georgiou, A. Russell and A. Shvartsman, "Work-Competitive Scheduling for Cooperative Computing with Dynamic Groups", *SIAM Journal on Computing*, Vol. 34, No. 4, pp. 848–862, 2005.

[38] C. Georgiou, T. Pavlides and A. Philippou, "Network Uncertainty in Selfish Routing", *CD-ROM Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium*, April 2006.

[39] C. Georgiou and A. Shvartsman, "Cooperative Computing with Fragmentable and Mergeable Groups", *Journal of Discrete Algorithms*, Vol. 1, No. 2, pp. 211–235, April 2003.

[40] J. Groote, W. Hesselink, S. Mauw and R. Vermeulen, "An Algorithm for the Asynchronous Write-All Problem Based on Process Collision", *Distributed Computing*, Vol. 14, No. 2, pp. 75–81, April 2001.

[41] M. Inoue, W. Chen, T. Masuzawa and N. Tokura, "Linear Time Snapshots Using Multi-Writer Multi-Reader Registers", *Proceedings of the 8th International Workshop on Distributed Algorithms*, Vol. 857, pp. 130–140, September/October 1994.

[42] A. Israeli, A. Shaham, A. Shirazi and T. Masuzawa, "Linear-Time Snapshot Implementations in Unbalanced Systems", *Mathematical Systems Theory*, Vol. 28, No. 5, pp. 469–486, September/October 1995.

[43] P. Kanellakis and A Shvartsman, *Fault-Tolerant Parallel Computation*, Kluwer Academic Publishers, 1997.

[44] A. Kaporis and P. Spirakis, "The Price of Optimum in Stackelberg Games on Arbitrary Single Commodity Networks and Latency Functions", *Proceedings of the 18th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pp. 19-28, July 2006.

[45] Z. Kedem, K. Palem, A. Raghunathan and P. Spirakis, "Combining Tentative and Definite Executions for Dependable Parallel Computing", *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pp. 381–390, May 1991.

[46] Z. Kedem, K. Palem and P. Spirakis, "Efficient Robust Parallel Computations", *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pp. 138–148, May 1990.

[47] E. Koutsoupias and C. H. Papadimitriou, "Worst-Case Equilibria", *Proccedings of the 16th International Symposium on Theoretical Aspects of Computer Science*, Vol. 1563, pp. 404–413, March 1999.

[48] D. Kowalski and A. Shvartsman, "Performing Work with Asynchronous Processors: Message-Delay-Sensitive Bounds", *Information and Computation*, Vol. 203, No. 2, pp. 181–210, December 2005.

[49] D. Kowalski and A. Shvartsman, "Writing-All Deterministically and Optimally Using a Non-Trivial Number of Asynchronous Processors", *Procedings of the 16th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pp. 311–320, June 2004.

[50] F. Kuhn, T. Moscibroda and R. Wattenhofer, "What Cannot be Computed Locally", *Proceedings of the 23rd Annual ACM Symposium on the Principles of Distributed Computing*, pp. 300–309, July 2004.

[51] F. Kuhn, T. Moscibroda and R. Wattenhofer, "The Price of Being Near-Sighted", *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 980-989, January 2006.

[52] F. Leighton, B. Maggs and S. Rao, "Packet Routing and Job-Shop Scheduling in $O(Congestion + Dilation)$ Steps", *Combinatorica*, Vol. 14, No. 2, pp. 167–186, June 1994.

[53] N. Linial, "Locality in Distributed Graph Algorithms", *SIAM Journal on Computing*, Vol. 21, No. 1, pp. 193–201, February 1992.

[54] M. Loui and H. Abu-Amara, "Memory Requirements for Agreement among Unreliable Asynchronous Processes", *Advances in Computing Research*, Vol. 4, pp. 163-183, 1987.

[55] M. Luby, “A Simple Parallel Algorithm for the Maximal Independent Set Problem”, *SIAM Journal on Computing*, Vol. 15, No. 4, pp. 1036-1053, November 1986.

[56] G. Malewicz, “A Work-Optimal Deterministic Algorithm for the Certified Write-All Problem with a Nontrivial Number of Asynchronous Processors”, *SIAM Journal on Computing*, Vol. 34, No. 4, pp. 993–1024, April/May 2005.

[57] G. Malewicz, A. Russell and A. Shvartsman, “Distributed Scheduling for Disconnected Cooperation”, *Distributed Computing*, Vol. 18, No. 6, pp. 409–420, June 2006.

[58] C. Martel, A. Park and R. Subramonian, “Work-Optimal Asynchronous Algorithms for Shared Memory Parallel Computers”, *SIAM Journal on Computing*, Vol. 21, No. 6, pp. 1070–1099, December 1992.

[59] I. Milchtaich, “Congestion Games with Player-Specific Payoff Functions”, *Games and Economic Behavior*, Vol. 13, No. 1, pp. 111–124, April 1996.

[60] D. Monderer and L. S. Shapley, “Potential Games”, *Games and Economic Behavior*, Vol. 14, No. 1, pp. 124–143, May 1996.

[61] T. Moscibroda and R. Wattenhofer, “Facility Location: Distributed Approximation”, *Proceedings of the 24th Annual ACM Symposium on the Principles of Distributed Computing*, pp. 108-117, July 2005.

[62] J. F. Nash, “Equilibrium Points in n-Person Games”, *Proceedings of the National Acanemy of Sciences of the United States of America*, Vol. 36, pp. 48–49, 1950.

[63] J. F. Nash, “Non-Cooperative Games”, *Annals of Mathematics*, Vol. 54, No. 2, pp. 286–295, 1951.

[64] C. Papadimitriou and M. Yannakakis, “Linear Programming Without the Matrix”, *Proceedings of the 25th Annual ACM Sumposium on Theory of Computing*, pp. 121-129, May 1993.

[65] C. Papadimitriou and M. Yannakakis, “On the Value of Information in Distributed Decision-Making”, *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing*, pp. 61–64, August 1991.

[66] H. Racke, “Minimizing Congestion in General Networks”, *Proceedings of the 43rd Annual Symposium on the Foundations of Computer Science*, pp. 43–52, November 2002.

[67] S. Schmid and R. Wattenhofer, “Algorithmic Models for Sensor Networks”, *Proceedings of the 14th International Workshop on Parallel and Distributed Real-Time Systems*, April 2006.