On the Universal Computing Power of Amorphous Computing Systems

Wiedermann, Jiří
2007

Institute of Computer Science
Academy of Sciences of the Czech Republic

On the universal computing power of amorphous computing systems

Jiří Wiedermann and Lukáš Petrů

Technical report No. 1009

October 2007

Pod Vodárenskou věží 2, 182 07 Prague 8, phone: +4202 6605 3520, fax: +420 286 585 789,
e-mail:jiri.wiedermann@cs.cas.cz

Institute of Computer Science
Academy of Sciences of the Czech Republic

On the universal computing power of amorphous computing systems[1]

Jiří Wiedermann[2] and Lukáš Petrů[3]

Technical report No. 1009

October 2007

Abstract:

Amorphous computing differs from the classical ideas about computations almost in every aspect. The architecture of amorphous computers is random, since they consist of a plethora of identical computational units spread randomly over a given area. Within a limited radius the units can communicate wirelessly with their neighbors via a single-channel radio. We consider a model whose assumptions on the underlying computing and communication abilities are among the weakest possible: all computational units are finite state probabilistic automata working asynchronously, there is no broadcasting collision detection mechanism and no network addresses. We show that under reasonable probabilistic assumptions such amorphous computing systems can possess universal computing power with a high probability. The underlying theory makes use of properties of random graphs and that of probabilistic analysis of algorithms. To the best of our knowledge this is the first result showing the universality of such computing systems.

Keywords:
amorphous computing systems; universal computing; E random access machine; simulation; complexity

[1]This research was carried out within the institutional research plan AV0Z10300504 and partially supported by the GA ČR grant No. 1ET100300419 and GD201/05/H014. A preliminary, shorter version of this paper has been presented at the Third Conference on Computability in Europe, CiE 2007, Siena, Italy, June 2007 and published in the proceedings from this conference.

[2]Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod Vodárenskou věží 2, 182 07 Prague 8, Czech Republic

[3]Faculty of Mathematics and Physics, Charles University, Malostranské náměstí 25, 118 00 Prague 1, Czech Republic, lukas.petru@st.cuni.cz

1 Introduction

Classical models of universal computations, such as Turing machines, RAMs, etc., are rigorously defined mathematical structures in whose design there is no room for randomness. The situation is slightly different when the computing systems represented by networks of processors (such as the Internet, wireless networks, etc.) are considered: here, the network topology may result from a random process. In order to "compute" bold assumptions about such networks have been usually made: at least we require that all network nodes are connected by communication links, that prior to the start of computation each network node possesses a unique "network address", that there are communication primitives supporting message exchange and, last but not least, that each network node does posses a universal computing power. Such models have been the domain of the classical computational theory of distributed systems. However, recent developments in micro-electro-mechanical systems, wireless communications and digital electronics have brought yet a new challenge into the area of distributed computing systems. Their new instances integrate sensing, data processing and wireless communication capabilities. Typical representatives of such systems are sensor, mobile, or ad-hoc wireless networks (cf. [11]). At an extreme end, people consider exotic systems such as smart dust (cf. [14]) or amorphous computers (cf. [1], [2], [4]). In these systems the miniaturization is pushed to its limits resulting, presumably, into processors of almost molecular size with the respective communication and computing facilities adequately (and thus severely) restricted. These limitations call for the change of the basic computational and communication model of distributed computing systems which must subsequently be also reflected in the design of the corresponding algorithms.

It seems that so far the related research has mainly concentrated on the concrete hardware, software and algorithmic issues neglecting almost completely the computational and complexity aspects in that kind of computing (cf. [13]). Very often the designers of such algorithms have paid little attention to the underlying computational model and, e.g., they have taken for granted the universal computing power of all processors, synchronicity of time in all processors, the existence of unique node identifiers and those of communication primitives allowing efficient message delivery.

In our paper we concentrate on a computational model of a wireless communication network where such assumptions do not hold. This could be the case of e.g., the smart dust mentioned earlier. Our model, called *amorphous computer* works under very week assumptions: basically, it is a random graph which emerges by distributing the nodes randomly in the bounded planar area. The graph's nodes are processors represented by probabilistic finite state automata possessing no unique identifiers ("addresses"). The graph's edges exist only among the nodes within the bounded reach of each node's radio. Each node operates asynchronously, in either broadcasting or listening mode, hearing a message only if it is sent exactly by one of the node's neighbors. That is, there is no mechanism distinguishing the case of a multiple broadcast from the case of no broadcast. This model has been introduced recently by the authors in [12].

Due to its weak (and thus, general) underlying assumptions which correspond well to various instances of amorphous computing as described in the literature, we believe that such a model presents a fundamental model of amorphous computing (cf. [1], [2], or an overview in [4]). Within the theory of computation a model of an amorphous computer, as given by our definition, represents an interesting object of study by itself since it contains elements of randomness built–in into both the computer's "set–up process" and its operations. The fundamental question is, of course, whether such a model does possess a universal computational power. The first steps towards this end have been taken in [12]. Here, under the above mentioned mild assumptions concerning the communication among the nodes of an amorphous computer and under reasonable statistical assumptions on the underlying graph a scalable randomized auto-configuration protocol enabling message delivery from a source node to all other nodes has been designed. For networks whose underlying communication graph has N nodes, diameter D and node degree Q, the complexity of this protocol is $O(DQ\log(N/\varepsilon))$ with probability $\varepsilon > 0$ of failure.

For the synchronous case, the problem of message delivery similar to the one mentioned above has been studied in the seminal paper by Ben-Yehuda, Goldreich and Itai in 1993 [3]. Under the same notation as above, the algorithm of Ben-Yehuda et al. runs in time $O((D + \log(N/\varepsilon))\log N)$. This algorithm is faster than the former one, but the assumption of synchronicity (allowing that

all nodes can start a required action simultaneously) is a crucial one for its correctness. However, synchronization is exactly the feature excluded by the very definition of amorphous computing.

In the present paper, a further modification of the protocol from [12] is used in designing an algorithm simulating a unit–cost RAM (for inputs of bounded size). This simulation shows that systems of amorphous computers do possess a universal computing power. To the best of our knowledge this is the first result of this kind.

A formal model of the amorphous computer is described in Section 2. In Section 3, for the sake of completeness, the asynchronous communication protocol from [12] is briefly presented followed by a new version of the broadcasting algorithm. Some useful properties of random graphs pertinent to our application are mentioned in Section 4. The main result of the paper, i.e., an algorithm simulating a unit-cost RAM with a bounded-size input on our model of the amorphous computer, and its complexity analysis, is given in Section 5. Section 6 is devoted to conclusions.

2 Model

In order to be able to prove universality of any model of computation we have to define this model quite rigorously: only then we can design plausible algorithms for it. To that end we give the definition of an amorphous computer as introduced in [12].

Definition 1 *An* **amorphous computer** *is a sextuple* $\mathcal{A} = (N, S, P, A, r, T)$ *where*

1. N *is the* number of processors *(also called* nodes*) in the underlying network. Each node is created by a RAM enhanced by a module for wireless sending and receiving. All nodes are identical, controlled by the same program, except of a single distinguished node called the* base station*. In addition to the standard node facilities (see below) this node is capable to send and receive data to/from a remote operator and is used to enter the data into the AC and to send the results of AC data processing to the operator.*

2. *Each RAM has a fixed number of registers. Each register has length* s *bits and can contain one of* $S = 2^s$ *different values. Every RAM is equipped by a special read-only register called* rand*, a special read-only register* $\mathrm{r_{in}}$ *and a special write-only register* $\mathrm{r_{out}}$*. On each read, register* rand *delivers a new random number. The registers in all nodes are initialized by the same starting values.*

3. P *is a* random process *assigning to each node a position with continuous uniform distribution over a* planar area A*, independently for each node.*

4. r *gives the radius of a* communication neighborhood*. Any two nodes at distance at most* $r > 0$ *are called neighbors. All neighbors of a node form the* node's neighborhood.

5. $T > 0$ *is* transmission time *of a message within a neighborhood of any node.*

6. *(*Asynchronicity:*) In each RAM any instruction takes one unit of time. The actions (computations, communication) of all processors are not synchronized.*

7. *The nodes communicate according to the following rules:*

   - *all nodes broadcast on the same channel;*
   - *if a node writes a value representing a* message *to* $\mathrm{r_{out}}$*, this message is broadcasted to its communication neighborhood;*
   - *if none of the given node's neighbors is broadcasting a message, then the given node register* $\mathrm{r_{in}}$ *contains an* empty message λ;
   - *if exactly one of a given node's neighbors is broadcasting a message* m*, then after time* T *register* $\mathrm{r_{in}}$ *in the given node contains* m;

- *if two or more of the node's neighbors are broadcasting a message and the time intervals of broadcasting these message transfers overlap, then there is a so–called* collision *and the* r_{in} *register of the receiving node contains empty message* λ;
- *the nodes have no means to detect a collision, i.e., to distinguish the case of no-broadcast from the case of a multiple broadcast.*

Note that since the register size of each RAM is finite, each RAM inclusively its random number generator can be seen as a probabilistic finite automaton of size $O(S)$ (because each RAM has but a constant number of registers). However, we have chosen to see each automaton as a "little RAM" since such a view will support the result we are after (i.e., a simulation of a unit-cost RAM) and corresponds more to practice.

An AC operates as follows. The input data enter the AC via its base station. From there, the data (which might also represent a program for the processors) spread to all nodes accessible via broadcasting. In a "real" AC additional data might also enter into individual processors via their sensors which, however, are not captured in our model since they do not influent the universality result. Then the data processing within processors and data exchange among processors begins. The results are delivered to the operator again via the base station.

In practice, nobody would probably try to simulate a universal computation by an AC. Nevertheless, a proof that this is in principle possible hints to the flexibility of the proposed model and shows that whatever computational problem will be encountered in practice, it would be solvable on our model. In this sense, the proof of universality presents a "killer application" for the underlying model.

3 Asynchronous Communication Protocol

In order to enable communication among all (or at least: a majority of) available processors the underlying communication graph of our AC must have certain desirable properties. The properties which are of importance in this case are: graph connectivity, graph diameter and the maximal degree of its nodes. Obviously, a good connectivity is a necessary condition in order to be able to harness a majority of all processors. Graph diameter bounds the length of the longest communication path. Finally, the node degree (i.e., the neighborhood size of a node) determines the collision probability on the communication channel.

An instance of an amorphous computer $\mathcal{A}$ whose underlying computational graph has a maximal connected component of size N containing the base station is called a *well–formed instance* of $\mathcal{A}$ of size N.

Assuming that the nodes of an AC could participate in its computation there must exist an algorithm of node–to–node communication used by the nodes to coordinate their actions. Such an algorithm will consist of two levels. The lower level is given by a basic randomized broadcasting protocol enabling each node to broadcast a message to its neighborhood. Making use of this protocol we extend it, on the second level, to a broadcasting algorithm that can be used to broadcast a message from a given node to all other network nodes.

Protocol Send: A node is to send a message m with a given probability $\varepsilon > 0$ of failure. The protocol must work correctly under the assumption that all nodes are working concurrently, asynchronously, using the same protocol and hence possibly interfering one with each other's broadcast.

The idea is for each node to broadcast sporadically, minimizing thus a communication collision probability in one node's neighborhood. This is realized as follows. Each node has a timer measuring *timeslots* (intervals) of length $2T$ (T is time to transfer a message between any two neighbors). During its own timeslot, each node is allowed either listen, or to send a message at the very beginning of its timeslot (and then listen till the end of this timeslot). At the start of each timeslot a node sends m with probability p and this is repeated for k subsequent timeslots. The probability of sending m is determined by the node's random number generator. The values of p and k are given in the proof of the following theorem.

Theorem 1 (Sending a message) *Let* $\mathcal{A}$ *be a well–formed instance of an amorphous computer, let the underlying computational graph have the maximal neighborhood size bounded by* Q. *Let* $1 >$

$\varepsilon > 0$ *be an priori given allowable probability of failure. Assume that all nodes send their messages asynchronously according to the* Protocol Send. *Let X be a node sending message m and Y be any of X's neighbors. Then* Protocol Send *delivers m to Y in time $O(Q\log(1/\varepsilon))$ with probability at least $1-\varepsilon$.*

Sketch of the proof: Thanks to our choice of the length of the timeslots, for each timeslot of a given node X there is exactly one corresponding timeslot of some other node Y such that if both nodes send asynchronously in their timeslots, only a single collision will occur. This is so because if X has started its sending at the beginning of its timeslot, X's and Y's sendings overlap if and only if Y had started a sending in a timeslot that was shifted w.r.t. the beginning of X's timeslot by less than T time units in either time direction. The timeslots of length shorter than $2T$ could cause more than a single broadcast collision between the arbitrary pairs of nodes, whereas longer timeslots would delay the communication.

We will treat message sendings as independent random events. Message m is correctly received by Y in one timeslot if X is transmitting m (the probability of such event is p) and none of Y's neighbors is transmitting (the corresponding probability is $(1-p)^Q$), giving the joint probability $p(1-p)^Q$. The value of $p(1-p)^Q$ is maximized for $p = 1/(Q+1)$. The probability of a failure after k timeslots is $[1-p(1-p)^Q]^k = \varepsilon$. Hence, $k = \log\varepsilon/\log[1-p(1-p)^Q]$. The denominator in the latter expression equals $-\sum_{i=1}^{\infty}[p(1-p)^Q]^i/i \leq -p(1-p)^Q = -1/(Q+1)(1+1/Q)^{-Q} \leq -e^{-1}/(Q+1)$ leading to $k = O(Q\log(1/\varepsilon))$.

□

Note that the protocol can work for any value of ε. However if we want k to fit into one register then $k \leq S$ must hold. This would impose a bound in the form $\varepsilon \geq [1-p(1-p)^Q]^S$ on the allowable value of ε.

In order to send a message to any node of an AC we use the idea of flooding the network by that message, i.e, broadcasting the message to all nodes of the network.

Algorithm Broadcast. This algorithm is used to deliver a message m from a node X in the network to all remaining nodes which are not locked w.r.t. to m. A node is called *locked* w.r.t. a message z if and only if except of z, the node transmits any other incoming message. To broadcast m to the network, X sends m using *Protocol Send* with probability ε/N of failure. Upon receiving m, any node sends m using *Protocol Send* with failure probability ε/N. The locking mechanism is implemented straightforwardly: each node remembers the last sent message and ignores it if it is received again—it "locks itself" w.r.t. that message.

Theorem 2 (Broadcasting) *Let D be the diameter of the communication graph. Then, for any ε : $0 < \varepsilon \leq 1$,* Algorithm Broadcast *delivers m to each node that has not been locked w.r.t. m in time $O(DQ\log(N/\varepsilon))$ and with probability $1-\varepsilon$. Afterwards, all nodes will be in a locked state w.r.t. m.*

Sketch of the proof: Starting from X, m spreads through the network in waves as a breadth-first search algorithm of the communication graph starting in X would do it. Transmission between waves takes time $O(Q\log(N/\varepsilon)$. Afterwards, the nodes of the previous wave enter a locked state. Obviously, after repeating this process in parallel, asynchronously, at most D times, m will reach all nodes and all nodes will be in a locked state w.r.t. m. The algorithm thus takes time $O(DQ\log(N/\varepsilon))$. For one node the failure probability is ε/N and for the whole network this probability will rise to ε.

□

Again as with the sending protocol, the achievable error ε is bounded if we want value k of the *Protocol Send* to fit into one register. For the error probability of algorithm broadcast it holds that $\varepsilon \geq N[1-p(1-p)^Q]^S$.

Note that our broadcasting protocol can reliably handle only one message at a time transferred through the network and that no same messages following one after the other can be processed by the protocol, since after re-sending it the nodes get locked w.r.t. the first message and remain locked until a different message arrives.

4 Properties of random networks

Note that our definition of an AC makes no assumptions about the underlying communications graph whereas the statements of both Theorems 1 and 2 have referred to the underlying communication graphs. This has been so since only some graphs are "good" for our purposes while the others cannot support any interesting computations. In the previous theorems the appropriateness of the underlying graphs has been ensured in theorems' assumptions. However, by the definition of an AC, its communication network is shaped by process P as a result of the node placement (cf. Definition 1, item 3), which means that the resulting network has a random structure. Now we will be interested under what conditions a randomly emerging network will have the properties assumed in the previous theorems. As we have seen, for the basic protocol to work we needed connected networks. Moreover, in order to estimate the efficiency of the protocol we made use of the diameter and of the maximum neighborhood size of the respective networks. Therefore we will focus onto the latter mentioned properties of random networks.

For an amorphous computer $\mathcal{A} = (N, S, P, A, r, T)$ its *node density* d is defined as $d = N\pi r^2/a$ (a denotes the size of area A). In the rest of the paper we assume that the nodes constituting a network are distributed uniformly randomly (by process P) over a square area A with a given density.

Connectivity A *connected component* of a graph comprises all nodes among which a multi-hop communication is possible.

The existence of an edge between two nodes is a random event depending on the random positions of the nodes. The probability of edge presence is higher with larger communication radius r and is lower when the nodes are spread over a larger area A.

Node density d gives the average number of nodes in the communication area of one node. Depending on the node density we expect to observe different topology of the node connections graph. For low densities, the majority of the nodes will be isolated with high probability. For medium densities connected components of fixed average size (not depending on the total number of nodes) will be formed with high probability.

Similar model to ours is studied by *percolation theory* (cf. [7]). In the so-called continuum percolation model, circular disks are placed at random positions in an infinite plane. This would correspond to our model of amorphous computer, where the radius of the disk would be $r/2$. The *percolation theory* shows that there is a critical density of the disks above which there emerges an infinite connected component. The critical density is computed numerically and its value is around 4.5 (see [6]). This result tells us that we will not obtain any large component with density lower than this. However, even this critical density does not guarantee reasonably large component in our amorphous computer. We would like that there is only a little fraction of nodes that are not connected to the single large component. For this we need density even higher than the percolation threshold.

In order to find out which node density ensures reasonably large connected component we made experiments for several node counts. For each node count the experiment consisted of 400 runs. In each run we created one random graph and observed the size of its largest component. Then we estimated the component size such that it was not achieved only in 2 % of the experimental runs (i.e., we estimate the 2nd percentile). The results are shown in Fig. 1. We have verified that density $d = 6$ gets components of reasonable size, and all presented experiments are using this density.

The interpretation of the results is following. Let's take the case $N = 100$. The obtained value 0.48 means that a random realization of a graph with N nodes will have a component with more than $0.48N$ nodes with probability 98 %. As can be seen from our figure, for larger node counts the fraction tends to rise.

Thus, whenever an AC with at least 100 connected nodes is needed, we should actually create an AC with $100/0.48 = 210$ nodes. The expected AC will then have a component containing 100 connected nodes with 98 % probability. The penalty of this scenario is that a constant factor of nodes gets wasted which may only be acceptable for cheap devices. This is in contrast with, e.g., the ad-hoc networks scenario using expensive devices, where full connectivity is sought and the node density must rise above the connectivity threshold which is of order $\Omega(\log N)$ (cf. [8]).

Diameter The *diameter* of a graph is the maximum length of a shortest path between any two vertices of that graph.

Analytically, the size of a random graph diameter has been derived, e.g., in [5]. It shows that the diameter is about $O(D/r)$, where D is the diameter of a circle circumscribing the area containing the nodes. But we cannot directly apply this result to our AC. First, the result has been proved only for the asymptotic case when number N of nodes goes to infinity. Second, the referred result holds only when the node density is above the connectivity threshold (which is $\Omega(\log N)$). In our scenario, we have used constant node density that is above the percolation threshold but below the connectivity threshold.

In [10] an experiment was carried out measuring the diameter size while varying both the number of nodes and the transmission range. However, we are interested in the behavior of the graph diameter when the node density remains fixed. Therefore we performed 400 test runs with various numbers of nodes with density $d = 6$ and measured the 98th percentile of a graph diameter. The results are shown in Fig. 2.

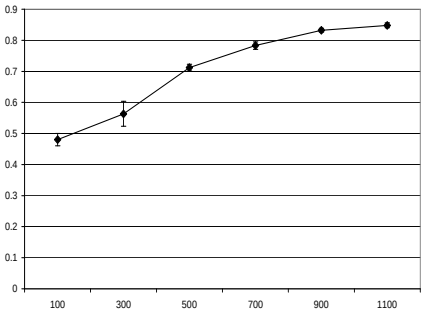

Fig. 1: The value of the 2nd percentile of the size of the largest component vs. node count in a random graph

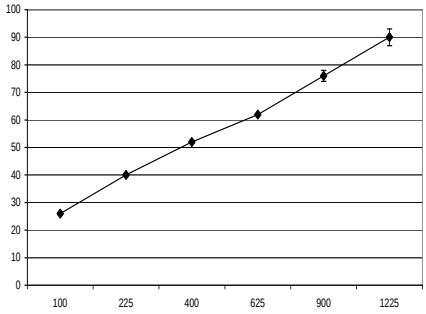

Fig. 2: The 98th percentile of graph diameter vs. node count in a random graph

Our experiments show that the graph diameter follows the asymptotic expression derived in [5] also below the connectivity threshold. When the node density d is fixed, then $O(D/r) = O(\sqrt{N/d})$. In Fig. 2, the node count increases quadratically and it can be easily seen that the graph diameter rises roughly linearly with $\sqrt{N}$. We see that an upper bound in the form *diameter* $= 2.7\sqrt{N}$ holds. We expect that at most in 2 % of random realizations the graph diameter will be larger than this value.

Maximum neighborhood size can be estimated by applying the techniques known from the solutions of the classical occupancy problem.

Theorem 3 *Let $\mathcal{A} = (N, S, P, A, r, T)$ be an AC with N nodes randomly uniformly dispersed by process P with density d over a square area A, let $Q = \lceil 8 \log N / \log\log N \rceil$. Then for a sufficiently large N, the probability that there are more than $12Q$ nodes in any communication neighborhood of a node is less than $4/(dN)$.*[4]

Proof: We start by exactly covering A of size a by $H = h \times h$ squares, with $h \in \mathbb{N}$; the size of each square is chosen so that its size is maximal, but not greater than the area πr^2 of a communication neighborhood. Then $(h-1)^2\pi r^2 < a \leq h^2\pi r^2$ and since $a = N\pi r^2/d$, we get $N/H \leq d$ and for $h \geq 4$, $H/N < 2/d$.

We estimate the probability $p_{=k}$ that in a randomly selected square there will be exactly k nodes for k "not too small" (see in the sequel). Let us consider all sequences of length N over $\{1, 2, \ldots, H\}$ of node "throws" into H squares numbered by $1, 2, \ldots H$. There are H^N of such sequences, each of them being equally probable. Consider any i, $1 \leq i \leq H$. There are $(H-1)^{N-k}\binom{N}{k}$ sequences containing exactly k occurrences of i. Then $p_{=k} = \binom{N}{k}\frac{(H-1)^{N-k}}{H^N} = \binom{N}{k}\frac{1}{H^k}\left(1-\frac{1}{H}\right)^{N-k}$ and the probability that there are at least k nodes in a square is $p_{\geq k} = \sum_{j=k}^{N} p_{=j} = \sum_{j=k}^{N}\binom{N}{j}\left(\frac{1}{H}\right)^j\left(1-\frac{1}{H}\right)^{N-j}$. Using

[4] All logarithms are to the base 2.

Stirling's approximation $\binom{N}{j} \leq (eN/j)^j$ and upper-bounding the last factor in the last expression by 1 we get $p_{\geq k} \leq \sum_{j=k}^{N} \left(\frac{eN}{jH}\right)^j \leq \sum_{j=k}^{N} \left(\frac{ed}{j}\right)^j \leq \left(\frac{ed}{k}\right)^k \sum_{j=0}^{\infty} \left(\frac{ed}{j}\right)^j \leq \left(\frac{ed}{k}\right)^k \sum_{j=0}^{\infty} \left(\frac{ed}{k}\right)^j$. The latter infinite series converges to $1/(1-ed/k)$ providing $ed < k$. Consider k such that $ed < k/2$; then the sum of the series is at most 2 and for $k \geq (ed)^2$, $p_{\geq k} \leq 2\left(\frac{ed}{k}\right)^k \leq 2.2^{-1/2k \log k}$.

For $k = Q$ we get $p_{\geq k} \leq 2.2^{-\frac{1}{2}\frac{8 \log N}{\log \log N}(3+\log \log N - \log \log \log N)} \leq 2/N^2$ (taking into account that for a sufficiently large N, $3 + \log \log N - \log \log \log N \geq \frac{1}{2} \log \log N$). It follows that the probability that in any of the H squares there will be at least Q nodes is $2H/N^2 < 4/(dN)$.

Finally, note that for $h \geq 2$ the size of a square is $s > ((h-1)/h)^2 \pi r^2 \geq 1/4\pi r^2$. Hence, the area of a communication neighborhood is smaller than the area of four squares. After realizing that the nodes from at most 12 squares can enter a circular neighborhood of area πr^2 the claim of the theorem follows. □

From Theorems 1 and 2 it follows that for graphs with the maximum neighborhood size bounded as in Theorem 3 the asymptotic time complexity of *ProtocolSend* is $O(\log N \log(1/\varepsilon)/\log \log N)$ and that of *AlgorithmBroadcast* $O(\sqrt{N} \log N \log(N/\varepsilon)/\log \log N)$, with high probability.

Note that the statistical properties of random networks do not depend much on the presence or non-presence of small random subsets of nodes. This is vital when considering the failure resilience of amorphous computers w.r.t. random node faults.

5 The universal computing power of an AC

In this section we show how to program the nodes of an amorphous computer so that it can actually compute according to an arbitrary unit-cost RAM program. Prior to the start of the simulation we must perform a setup procedure during which we initialize the amorphous computer. This initialization procedure consists of two phase: address assignment phase, and input reading phase.

After the computer is set up, the computation can start and run in an unattended way. After the termination of a computation, output data are obtained through the base station, using similar mechanism as that for entering the input data.

Each phase—address assignment, input entering, and simulation—can be performed with a given failure probability as decided by the operator. The failure probability of the complete simulation algorithm will then be the sum of failure probabilities of the individual phases.

5.1 Address assignment

Initially, all nodes of the amorphous computer are identical. The purpose of the address assignment phase is to break this symmetry and to assign different addresses to the different nodes. In the first part of the respective algorithm, the symmetry gets broken with the help of a random number generator: each node generates its own "random" address. In the second part of the algorithm, the node addresses are transformed into a continuous range between one and the number of different addresses generate in the previous part.

Assume that in order to perform the intended computation we need an address space of size $M > 0$. Assume that any register can hold $\log M + O(1)$ bit numbers. The following randomized algorithm makes use of $N = 2M$ processors in order to generate at least M different addresses of the registers with a high probability (the occurrence of registers with the same address does not harm).

The operator chooses the allowed error probability ε_1 with which the algorithm can fail due to a communication error. From this value the operator estimates the value $k = k(\varepsilon)$ for $\varepsilon = \varepsilon_1/N^2$, to be used by the underlying *Protocol Send*. Before the algorithm starts, the base station broadcasts the values of p, k and D to all nodes (cf. Theorem 1). The nodes will use parameter D to compute the so-called *flooding period*. This is the maximum time in which a broadcasted message reaches all nodes with an allowed failure probability. According to Theorem 1 and 2, flooding period has the length of $2TkD$. After that time all nodes stop sending that message.

Algorithm Generate_Addresses

1. All processors randomly generate and store a binary string of length $\lceil \log(2M+1) \rceil$ in a variable called *address*;
2. The base station initializes two variables, $round := 1$ and $max := 1$;
3. Using *Algorithm Broadcast,* the base broadcasts pair $(round, max)$ to all processors;
4. Upon receiving this message, each processor whose $address = round$ waits for the length of one flooding period. Then, using *Algorithm Broadcast* the node sends a confirmation — message "0" — back to the base and resets its *address* to max;
5. If within the time of three flooding periods the base receives at least one confirmation, max is increased by 1;
6. After the time of three flooding periods has elapsed, *round* is increased by 1.
7. If $round < 2M$ then go to step 3;
8. If $max \geq M$ then HALT else go to step 1.

Clearly, variable max counts the number of processors with different addresses whose confirmation has been delivered to the base station. In Step 4 waiting for one flooding period before sending a confirmation for each node is necessary in order to ensure that no two different messages travel in the network simultaneously (remember that the broadcasting protocol only works correctly under the latter assumption). Waiting for three flooding periods in Step 5 and 6 is sufficient for a message sent by the base station to reach the farthest node in the network, for this node to wait for the duration of one flooding period and eventually for a confirmation sent by this node to return back to the base station, with high probability. Finally, note that in order to reach the failure probability of algorithm *Generate_Addresses* to be less or equal ε_1 the failure probability of a single invocation of *Protocol Send* must have been set to ε_1/N^2 since during a single run of the previous algorithm this protocol is invoked $O(N^2)$ times in the worst case.

Algorithm *Generate_Addresses* is repeated until at least M different addresses are generated. The probability that this happens already after the first trial is high and tends to 1 with the increasing M:

Lemma 5.1 *Choosing* $2M$ *random numbers uniformly distributed in interval* 1 *to* $2M$*, the probability that only* M *or less different numbers were chosen is less than* $1/\sqrt{M+1}$*.*

Proof: There are $(2M)^{2M}$ sequences of length $2M$ over $\{1, 2, \ldots, M\}$. Among them, there are M^{2M} sequences "made of" at most M different numbers which can be selected in $\binom{2M}{M}$ different ways. Hence the probability that a sequence contains M or less different numbers is $\binom{2M}{M} M^{2M}/(2M)^{2M}$. By induction, one can prove that $\binom{2M}{M} < 2^{2M}/\sqrt{M+1}$. The claim of the theorem follows. □

Theorem 4 *In a well-formed instance of an amorphous computer of size* $N = 2M$ *algorithm* Generate_Addresses *generates* M *different addresses with probability* $1 - 1/\sqrt{M+1} - \varepsilon_1$ *in time* $O(NDQ \log(N^2/\varepsilon_1))$.

5.2 Input entering and register initialization

The operator chooses allowed error probability ε_2 for communication failure in the process of input entering and computes value $k = k(\varepsilon)$ for $\varepsilon = \varepsilon_2/(MN)$, to be subsequently used by the underlying *Protocol Send*. Before the algorithm starts, the base station broadcasts the values of p, k and D to all nodes.

To make the subsequent simulation completely independent from the interaction of the AC with its operator, we will request that the input data of size n will be initially stored in the first n nodes of our AC. Originally, the input data are available to the base station which obtains them in a sequential

manner from the operator and broadcasts it, one by one, to the respective registers. For each address i in the range $1...M$ the base station broadcasts a message of the form $(STORE, i, x_i)$, where x_i is the initial value of register i. The base station waits for one flooding period after each broadcasted message.

In principle, this above described input entering scenario allows entering of data also during an interactive computation (which might be a more natural way of utilizing an AC than its use as a universal computer) through the sensors of individual nodes.

5.3 Simulation

We show the universal computing power of on AC by letting it simulate a unit–cost RAM. We will assume that the entire RAM program as well as two RAM accumulators are stored in the base station. In addition to the input data, the contents of the RAM registers will be also held in the individual AC processors.

The RAM program will consist of the usual kind of instructions. For simplicity we will assume that all instructions requiring two operands (indirect addressing, arithmetical operations) are realized in the following way. The first operand is assumed to be in the first accumulator. The second operand (if any) is to be delivered into the second accumulator. An instruction moving the register contents between a register and the accumulator is realized as follows. The base station broadcasts the current instruction holding the address of the register to which the instruction is pertinent to all nodes of the AC. The instruction is realized in the requested register (processor) which then sends back the confirmation along with the current contents of that register.

In order to perform a simulation the operator chooses an allowed communication failure probability ε_3. From this the operator computes value $k = k(\varepsilon)$ for $\varepsilon = \varepsilon_3/(2T(n)N)$, to be used by the underlying *Protocol Send*. Before the algorithm starts, the base station broadcasts the values of p, k and D to all nodes.

Then the simulation proceeds in rounds. In each round, the base station issues the instruction to be realized. The network is "flooded" by this instruction using *Algorithm Broadcast*. Upon arriving into any processor holding the respective register the instruction is realized. Subsequently, after waiting for one flooding period, a confirmation is broadcasted back to the base station, again by using the broadcast algorithm.

Algorithm Broadcast allows only one message at a time to be travelling over the network. As in the algorithm for addresses generation, the purpose of the flooding period is to allow all nodes to enter the locked state. Before replying to the base station's instruction message a node waits for one flooding period. On the other hand, the base station may send next instruction message only after delay of three flooding periods allowing enough time for the message to spread to the farthest node, a node's waiting for one flooding period and sufficient time for the reply to reach the base station.

In more detail, the simulation of the t-th instruction in the t-th round proceeds as follows.

Algorithm Simulate: Let ε_B be the probability of error of protocol *Broadcast*, $\varepsilon_B = N\varepsilon$ in a single roud of the simulation. At the beginning of the t–th round, we assume that the following invariant holds: the probability that communication error occured up to this round is $2(t-1)\varepsilon_B$.

In order to realize an instruction requiring a load from register i, or a store of value *contents* into register i, the base station broadcasts a triple of the form $(i, instr, contents)$, where $instr$ is either $LOAD$ or $STORE$. In the former case, *contents* is empty, whereas in the latter case *contents* holds the value to be stored in the i-th register. Upon arriving at any processor which is not in a locked state, the processor starts re-sending of the triple using *Protocol Send*. Moreover, upon arriving at a register whose $address = i$, after elapsing of one flooding period a load instruction makes the processor to broadcast a confirmation pair of form $(LOAD, reg[i])$ where $reg[i]$ is the contents of the i-th register. A store instruction is realized by performing the assignment $reg[i] := contents$ within the processor and again, after elapsing of one flooding period, by broadcasting a confirmation pair of the form $(STORE, empty)$. The base station waits for three flooding periods within which it should receive a confirmation of the t-th instruction realization. If during that time no confirmation is obtained, the simulation ends with an error.

By that time, the t-th instruction has been issued with probability of communication error ε_B. With the same probability of error, the base station has obtained the respective confirmation. The total probability of communication error up to now is $2(t-1)\varepsilon_B + 2\varepsilon_B = 2t\varepsilon_B$, hence the invariant holds.

Note that the assumption on no two same instructions following one after another is fulfilled since each transmitted instruction is followed by a confirmation.

□

As far as the reliability of our simulation algorithm is concerned, note that the algorithm consists of $T(n)$ rounds, where $T(n)$ denotes the time complexity of the original RAM algorithm. If each round fails with the probability $2\varepsilon_B$, the entire simulation will fail with probability $2\varepsilon_B T(n)$. Thus, choosing $\varepsilon_B = \varepsilon_3/(2T(n))$ leads to a simulation algorithm with the probability of failure at most ε_3.

Putting all the results together we see that in order to simulate for a given input of size n a unit cost RAM of time complexity $T(n)$ and space complexity $S(n)$ with error probability at most ε_3, we must first "set up" a well-formed instance of an amorphous computer of size at least $N \geq max\{2S(n), n\}$ processors (the size of the amorphous computer must be at least n in order to accommodate the input data). We get the following result:

Theorem 5 (Simulation) *Let $\mathcal{R}$ be a unit-cost RAM of time complexity $T(n)$ and space complexity $S(n)$. Let $\mathcal{A} = (N, S, P, A, r, T)$ be a well–formed instance of an amorphous computer of size $N \geq max\{2S(n), n\}$ with a communication graph of diameter D and with maximal neighborhood size Q. Then for any input of size $n : N \geq max\{2S(n), n\}$ and any $0 < \varepsilon_3 \leq 1$, any computation of $\mathcal{R}$ can be simulated by $\mathcal{A}$ in time $O(T(n)DQ\log(2NT(n)/\varepsilon_3))$, with probability of failure at most ε_3.*

Proof: Each simulation round consists of sending the instruction message by the base station and sending an answer by some node. Thus, communication in each round must be done with error probability $\varepsilon_B = \varepsilon_3/(2T(n))$. Thus, each round takes time $O(DQ\log(2NT(n)/\varepsilon_3))$. The whole simulation takes time $O(T(n)DQ\log(2NT(n)/\varepsilon_3))$.

□

Requiring that k must fit into one register of size s the smallest error we can achieve is $2T(n)N[1-p(1-p)^Q]^S$. Thus, for larger N and $T(n)$ we may need nodes with longer registers allowing computation with higher-precision numbers. But in any practical realization of an amorphous computer this should not be a significant limitation because the necessary register size rises very slowly, $S = O(\log(T(n)N))$.

Note that the simulation algorithm can be changed so that the RAM program to be simulated need not be contained within the base station. Rather, as a numbered ordered sequence of instructions the program can be stored, instruction by instruction, in the nodes of amorphous computer, one program instruction per node. In this form, prior to the start of simulation, the RAM program must be broadcasted by the base unit to the network. Afterwards, during the simulation, the role of the base station from the previous simulation is taken over by the registers containing the currently simulated instruction.

6 Conclusion

We have shown a universal computing power of a formalized model of an amorphous computer. The main departure point of the amorphous computing structures from other models of wireless networks or distributed computing is the randomness of the underlying network topology, anonymity of processors not possessing universal computing power and a principal lack of synchronicity combined with the impossibility to detect broadcasting collisions. Unlike the majority of the known models which work whenever appropriately programmed, this need not be the case with an amorphous computer since its nodes can be dispersed in an unlucky manner that does not support the computer's functionality. For our model we have designed and analyzed an algorithm simulating a unit-cost RAM with bounded size inputs. To the best of our knowledge, our simulation algorithm seems to be the first result showing the universal computing power of a family of amorphous computers of the type we have considered. This result is interesting from the viewpoint of the computability theory since it shows that universal computing power can also emerge in randomly organized "amorphous", non-uniform communication structures consisting of anonymous building elements not possessing universal computing power.

Bibliography

[1] H. Abelson, Allen, D. Coore, Ch. Hanson, G. Homsy, T. F. Knight, Jr., R. Nagpal, E.Rauch, G. J. Sussman, R. Weiss.: Amorphous Computing. Communications of the ACM, Volume 43, No. 5, pp. 74–82, May 2000

[2] H. Abelson, et al.: Amorphous Computing. MIT Artificial Intelligence Laboratory Memo No. 1665, Aug. 1999

[3] R. Bar-Yehuda, O. Goldreich, A. Itai: On the Time-Complexity of Broadcast in Multi-hop Radio Networks: An Exponential Gap Between Determinism and Randomization. J. Comput. Syst. Sci. Vol. 45, No. 1, pp. 104-126, 1992

[4] D. Coore: Introduction to Amorphous Computing. Unconventional Programming Paradigms: International Workshop 2004, LNCS Volume 3566, pp. 99–109, Aug. 2005

[5] R. B. Ellis, et al.: Random Geometric Graph Diameter in the Unit Disk with ℓ_p Metric. Graph Drawing: 12th International Symposium, GD 2004, LNCS Volume 3383, pp. 167–172, Jan. 2005

[6] I. Glauche, et al.: Continuum percolation of wireless ad hoc communication networks. Physica A, Volume 325, pp. 577–600, 2003

[7] G. Grimmett: Percolation. 2nd ed., Springer, 1999

[8] P. Gupta, P. R. Kumar: Critical power for asymptotic connectivity in wireless networks, in Stochastic Analysis, Control, Optimization and Applications, pp. 547–566, Birkhauser, 1998.

[9] E. D'Hondt: Exploring the Amorphous Computing Paradigm. Master's Thesis, Vrije University, 2000

[10] Keqin Li: Topological Characteristics of Random Multihop Wireless Networks. Cluster Computing, Volume 8, Issue 2–3, pp. 119–126, July 2005

[11] S. Nikoletseas: Models and Algortihms for Wireless Sensor Networks (Smart Dust). In: SOFSEM 2006: Theory and Practice of Computer Science, Proceedings. Eds.: J. Wiedermann et al., LNCS Vol. 3831, Springer, pp. 65–83, 2007

[12] L. Petrů, J. Wiedermann: A Model of an Amorphous Computer and its Communication Protocol. Proc. SOFSEM'07, LNCS Vol. 4362, Springer, Berlin, 2007

[13] P. G. Spirakis: Algorithmic and Foundational Aspects of Sensor Systems: (Invited Talk). In: ALGOSENSORS 2004, Lecture Notes in Computer Science, 3121, 2004, pp. 3-8

[14] B. Warneke, et al.: Smart dust: communicating with a cubic-millimeter computer. Computer, Volume 34, No. 1, pp. 44–51, Jan. 2001