MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A.I. Memo No. 1626 Feb 16, 1998

An Algorithm for Group Formation and Maximal Independent Set in an Amorphous Computer

Radhika Nagpal, Daniel Coore

{radhi, newts}@martigny.ai.mit.edu

Abstract

Amorphous computing is the study of programming ultra-scale computing environments of smart sensors and actuators [1]. The individual elements are identical, asynchronous, randomly placed, unreliable, embedded and communicate with a small local neighborhood via wireless broadcast. In such environments, where individual processors have limited resources, aggregating the processors into groups is useful for specialization, increased robustness, and efficient resource allocation.

This paper presents a new algorithm, called the *clubs algorithm*, for efficiently aggregating processors into groups in an amorphous computer, in time proportional to the local density of processors. The clubs algorithm takes advantage of the local broadcast communication model of the amorphous computer and is efficient in an asynchronous setting. In addition, the algorithm derives two properties from the physical embedding of the amorphous computer: an upper bound on the number of groups formed and a constant upper bound on the density of groups. The clubs algorithm forms a general mechanism for symmetry breaking and can be extended to find the maximal independent set (MIS) and $\Delta + 1$ vertex coloring in $O(\log N)$ rounds, where N is the total number of elements and Δ is the maximum degree. Simulation results and example applications of clubs are also presented.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for this research was provided in part by DARPA under contract number N00014-96-1-1228 administered by ONR. Nagpal is also supported by Lucent GRPW Program. Address: Room 432, 545 Technology Square, Cambridge MA 02139.

1 Introduction

Recent developments in micro-fabrication and nanotechnology will enable the inexpensive manufacturing of massive numbers of tiny computing elements with integrated sensors and actuators. These computing and sensing agents can be applied to surfaces or embedded in structures to create active surfaces, improved materials and responsive environments [4, 8]. Amorphous computing [1] is the study of such ultra-scale computing environments, where the individual elements are bulk manufactured, randomly and densely distributed in the material, and communicate wirelessly within a small local neighborhood. The objective of this research is to determine paradigms for coordinating the behavior and local interactions of millions of processing elements to achieve global goals. In such environments, where individual processors have limited resources, aggregating processors into groups is a useful paradigm for programming. Groups can be used for increased robustness, task specialization and efficient resource allocation [2, 3, 5].

This paper presents a new algorithm, called *clubs*, for efficiently organizing an amorphous computer into groups. The local wireless broadcast and the asynchronicity of the amorphous computing elements make it difficult and inefficient to implement group forming algorithms that are designed for synchronous point-to-point networks, such as [3, 9]. The clubs algorithm forms groups in time proportional to the local density of processors by taking advantage of the local broadcast mechanism. The algorithm can be extended to the asynchronous environment without the use of complex synchronization and without sacrificing efficiency.

The algorithm also satisfies many constraints that arise in other distributed environments such as not having access to global IDs or knowledge of the topology. We show how the algorithm can be extended to deal with processor failure. In addition, two interesting bounds on the clubs algorithm can be derived from the physical embedding of the amorphous computer - an upper bound on the number of groups formed by the clubs algorithm and a maximum density with which these groups can be packed, irrespective of the total number of processors. We present results from running the clubs algorithm on an amorphous computer simulation to support the analysis.

The clubs algorithm also provides a general mechanism for symmetry breaking in an amorphous computer. We show how the clubs algorithm can be extended to solve traditional distributed computing problems like MIS and $\Delta+1$ vertex coloring in $O(\log N)$ rounds, where N is the total number of elements and Δ is the maximum degree. Lastly we present some example applications of using the clubs algorithm to self-organize non-local and point-to-point communication infrastructure on top of the local broadcast mechanism.

Section 2 presents the model for an amorphous computer. Section 3 presents the clubs algorithm. Section 4 presents the analysis of the algorithm with synchronous, asynchronous and unreliable processors. Section 5 presents the properties derived from the physical embedding. Section 6 presents simulation results. In Section 7 we show how the clubs algorithm can be extended to solve for MIS and $\Delta + 1$ coloring. Section 8 presents example applications of the clubs algorithm.

2 Computational Model

In this section we describe the model for an amorphous computer. This model, and the intuition behind it, are presented in more detail in [1]. An amorphous computer consists of myriad processing elements. The processing elements:

- are bulk manufactured and identical. They run the same program and do not have knowledge about the global topology.
- have limited computing resources.
- do not have globally unique identifiers but instead have random number generators for breaking symmetry.
- are asynchronous. They have similar clocks speeds but do not operate in lockstep.
- are unreliable. A processor may stop executing at any time - this failure model is known as stopping failures [10].
- have no precise interconnect. The processors are randomly and densely distributed. We assume that the density is sufficiently high so that all processors are connected and the variance in density is low.
- The processors are distributed on a surface or in a volume. Processors occupy physical

space and cannot be packed arbitrarily close. In this case we assume that the processors are placed on a two dimensional plane.

These features make it possible to cheaply manufacture and program large quantities of smart elements, and embed them in materials. What makes an amorphous computer different from traditional distributed and parallel computers is the physical embedding of the amorphous computer and the *communications model*.

- Processors communicate only with physically nearby processors. Each processor communicates locally with all processors within a circular region of radius r. The local neighborhood size is much smaller than the total number of processors and r is much smaller than the dimensions of the surface.

- Processors communicate with their local neighborhood by wireless broadcast. All processors share the same channel. Therefore collisions occur when two processors with overlapping broadcast regions send messages simultaneously. Collisions result in both messages being lost. A processor listening to the channel can detect a collision because it receives a garbled message. However the sender can not detect a collision because it it can not listen and transmit at the same time. This model is similar to that of multihop broadcast networks such as packet radio networks [11].

This communication model allows for the simple assembly of large numbers of nodes. However, the interference due to overlapping broadcast regions, lack of collision detection and asynchronicity of the processors make it difficult and inefficient to emulate point to point networks. In environments where processors have limited individual resources and are unreliable, assembling them into groups is advantageous. However group forming algorithms such as [3, 9], that are designed for point-to-point networks, are difficult to implement without a huge loss in efficiency. In addition such algorithms often require synchronizers to function correctly in asynchronous environments. Typical synchronizing techniques generate large numbers of messages [10], further exacerbating the problem of message loss. Hence algorithms designed for synchronous point-to-point distributed computers do not easily extend to the amorphous computing environment.

In the next sections we present the clubs algorithm, that takes advantage of the broadcast nature of the communications model and asynchronicity of the processors. The remainder of this section provides the notation that will be used in analyzing the performance of amorphous computing algorithms.

The amorphous computer can be represented as a graph where the nodes of the graph, V, are the processors and $N = |V|$. From the communication model, the set of edges $E = \{(i,j) | i,j \in V \wedge distance(i,j) \leq r\}$. Hence an edge has a maximum physical distance of r associated with it. Each processor has a local neighborhood defined by its communication region. For a processor i, its local neighborhood is $n(i) = \{j | (i,j) \in E\}$ and its degree in the graph is the size of its neighborhood, $d(i) = |n(i)|$. The average neighborhood size is d_{avg}, where $d_{avg} \ll N$. The number of edges $|E| = (d_{avg}N)/2$.

Processors occupy physical space and can not be placed on top of each other in a two dimensional plane. Therefore, there is a limit, ρ_{max}, on the number of processors that can fit in a communication region. ρ_{max} is a physical constant and is equal to $(\pi r^2 / processor\ size)$. Hence ρ_{max} provides a physical upper bound on the degree of any processor, i.e. $d(i) < \rho_{max}$, for all processors i. In addition, the number of neighbors within h hops of a processor is physically upper bounded by $h^2 \rho_{max}$ in a two dimensional plane.

3 Clubs Algorithm

The objective is to aggregate processors into groups. The groups formed should have three main properties.

1. All processors should belong to some group.

2. All groups should have the same maximum diameter.

3. A group should have local routing [2], which means that all processors within the group should be able to talk to each other using only processors within that same group.

These features make groups useful for resource allocation and self-organizing communication networks [5]. The clubs algorithm forms groups, called *clubs*, with a maximum diameter of two hops. We will first describe this algorithm assuming that the processors are reliable

```
integer R (upper bound for random numbers)
boolean leader, follower = false

procedure CLUBS ()
  t_i := R
  r_i := random[0,R)
  while (not follower and not leader)
     t_i = t_i - 1
     if (r_i > 0)
        r_i := r_i - 1
        if (not_empty(msg_queue))
           if (first(msg_queue) = "recruit")
              follower := true
     else
        leader := true
        broadcast("recruit")
  while (t_i ≥ 0)
     listen for other leaders
     t_i = t_i - 1
```

Figure 1: Clubs Algorithm

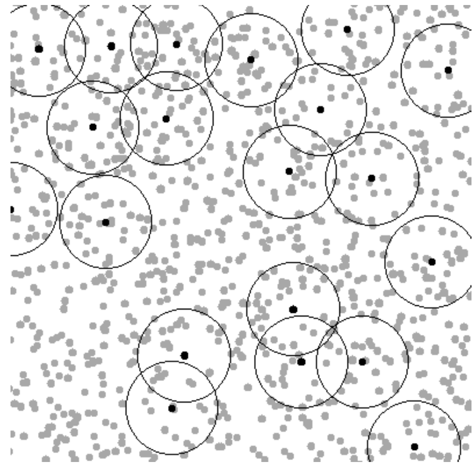

Figure 2: This figure shows leaders forming clubs. The dark processors are the current leaders and the circles around them represent their local broadcast region. All processors within this area are recruited as members of the leader's club.

and synchronous, and that messages are transmitted instantaneously. These assumptions will be removed in Section 4.

In the clubs algorithm, the processors compete to start new groups. The processors compete by choosing random numbers from a fixed integer range $[0, R)$. Then each processor counts down from that number silently. If it reaches zero without being interrupted, the processor becomes a group leader and recruits its local neighborhood into its group by broadcasting a "recruit" message. The processors that get recruited are called followers.

If a processor hears a recruit message from a neighbor before reaching zero, it becomes a follower. Once it has been recruited as a follower, it can no longer compete to form a new group. Therefore it stops counting down. However it keeps listening for additional recruit messages. Groups are allowed to overlap, and a processor can be a follower of more than one leader. If a processor detects a collision (hears a garbled message) while counting down, it assumes that more than one of its neighbors tried to recruit it at the same time. It becomes a follower and figures out its leaders later. The algorithm completes when all processors are members of some group (leaders or followers). Figure 1 presents the code run on a single processor and figures 2 and 3 show clubs forming on a simulation of an amorphous computer.

4 Analysis

4.1 Synchronous and Reliable Processors

Theorem 1: *The clubs algorithm completes in R steps and produces valid groups, when the processors are synchronous.*

Proof 1: If $[0, R)$ is the range from which random numbers are chosen, then the algorithm completes in time R. This is because each processor chooses to be a follower or leader by the end of its countdown and the countdown is chosen to be smaller than R. The club leader is adjacent to each processor in its club, which guarantees local routing as well as the maximum diameter of two hops. Clubs can be made non-overlapping by followers arbitrarily choosing one of the clubs they belong to. This does not violate the required group properties because the group leader still guarantees that its members are locally connected in two hops.

If we remove the assumption that messages are instantaneous and each message takes time m to transmit (all the messages are the same length in the algorithm), then each processor should mul-

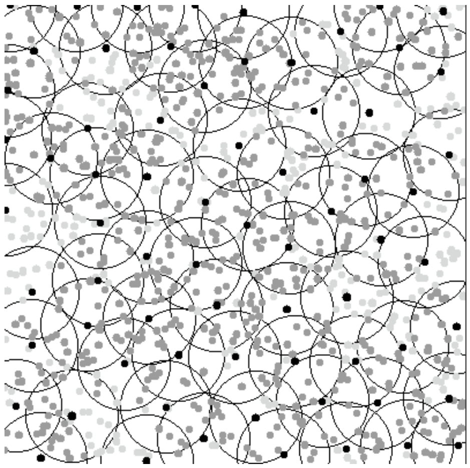

Figure 3: This shows the final clubs formed. All processors are either leaders or members of some club. Processors with a darker shade of gray belong to more than one club because they are in the overlapping region of several leaders' broadcast range.

tiply its random count, r_i, by m. If the processors are synchronous, they will broadcast only at intervals of m, thus preventing conflicts due to partially overlapping messages. The total time in that case is mR. By treating messages as instantaneous, we are simply normalizing the unit time to the transmission time, m.

Leadership conflicts: The algorithm introduces a natural spacing between clubs. Leaders prevent their neighbors from competing to form new clubs. Therefore leaders will be non-adjacent and at least r distance apart. However if two neighboring nodes declare leadership at the same time, then two clubs are formed such that their leaders are adjacent. This is a *leadership conflict*. The expected number of leadership conflicts is intimately related to the choice of the upper bound R. For many applications of clubs it is desirable that group leaders belong to only one club (their own) and that the overlap between clubs be limited [5]. In addition the spacing between clubs allows us to derive important properties on the graph induced by the clubs (Section 5). Therefore, we would like to keep the number of leadership conflicts low.

Theorem 2: *The expected number of leadership conflicts, $E(\text{conflicts})$, is at most $(\frac{d_{avg}}{2R})N$, for a synchronous amorphous computer.*

Proof 2: We will first analyze the expected leadership conflicts in a simplified version of clubs, called *sclubs*. Then we will show that the expected number of leadership conflicts in sclubs is at least as large as that in the original clubs algorithm.

In sclubs, leaders do not remove their neighbors from competition. Each processor chooses a random number, counts down and declares leadership when it reaches zero. Hence there are no followers.

Each processor i chooses a value r_i from the range $[0, R)$. At step k of the algorithm, processor i declares leadership if $r_i = k$. Processor i experiences a leadership conflict at step k if it chose $r_i = k$ and some neighbor did as well.

We can determine the expected number of leadership conflicts at step k.

$$E(\mathit{conflicts}_k) \leq \frac{1}{2} \sum_{i \in V} P(r_i = k \wedge \exists j \in n(i), r_j = k)$$

A leadership conflict involves at least 2 processors, therefore summing the probabilities of conflicts over all processors overcounts the number of conflicts at least twice. We can calculate the probability that a node i experiences a leadership conflict at step k.

$$P(r_i = k \wedge \exists j \in n(i), r_j = k) \leq \frac{1}{R} \cdot \sum_{j \in n(i)} \frac{1}{R} \leq \frac{d(i)}{R^2}$$

Hence,

$$\begin{aligned} E(\mathit{conflicts}_k) &\leq \frac{1}{2} \sum_{i \in V} \frac{d(i)}{R^2} \\ &\leq \frac{|E|}{R^2} \end{aligned}$$

Since the random numbers are chosen in the first step and the random numbers are less than R, there are R total steps.

$$\begin{aligned} E(\mathit{conflicts}) &\leq R \cdot \frac{|E|}{R^2} \\ &\leq \frac{d_{avg}}{2R} N \end{aligned}$$

Lemma: *The expected number of leadership conflicts for sclubs is at least as large as the expected number of leadership conflicts in the clubs algorithm.*

This is derived from the observation that for a given set of random values chosen by the nodes, the graph for sclubs, V_{sclubs}, is a superset of the graph for clubs, V_{clubs} at every step of the algorithm. This is easily proven using induction (proof omitted). As a result, the number of leadership conflicts in V_{sclubs} is greater than or equal to the number of leadership conflicts in V_{clubs} at any step k. Since this is true for any initial choice of random values, the overall $E(\mathit{conflicts})$ in sclubs is $\geq E(\mathit{conflicts})$ for the original clubs.

Corollary: *If we choose $R = \alpha d_{avg}$, where α is a constant and $\alpha > 1$, then the expected number of conflicts is at worst a constant fraction of the total number of nodes, $(1/2\alpha)N$.*

This follows immediately from Theorem 2. Thus, α can be chosen to make the percentage of leadership conflicts acceptably (or arbitrarily) small, at the expense of running time.

Apart from the leader conflicts, message collisions also occur when two leaders with overlapping broadcast ranges (i.e. within two hops of each other) broadcast at the same time. The choice of α also affects these collisions. Given that a processor has at most $4\rho_{max}$ neighbors within two hops and using arguments similar to those for calculating leadership conflicts, the expected number of collisions is $\leq \frac{4\rho_{max}}{2R}N$. Hence, if $R = \alpha d_{avg}$, then increasing α also decreases collisions.

Number of Messages: *The total number of messages is one per club.*

The only messages sent in this algorithm are leaders declaring the start of a new group, therefore the total number of messages is equal to the number of groups formed. In Section 5 we show that there is an upper bound on the number of clubs formed for a given physical embedding that does not depend on N. This gives us an upper bound on the number of messages. The algorithm is efficient in the number of messages because it takes advantage of the local broadcast mechanism, rather than using a point-to-point protocol.

Effect of the Distribution of Processors: The clubs algorithm works for arbitrary graphs and is not significantly affected by the processor distribution. The algorithm is most efficient when d_{avg} is small compared to N. The distribution does affect the resulting groups. The lower the variance in distribution, the lower the variance in the number of members in a group.

Global Knowledge: Since the processors may be programmed before being embedded in a surface, the algorithms should not depend on global knowledge of the topology. The clubs algorithm does not require knowing N or having global IDs. Processors can estimate the value of d_{avg} locally, if the variance in the density of processors is small. Alternatively processors can use the maximum neighborhood size, ρ_{max}, which is a physical property of the processor known at manufacture time.

No Global IDs: Although there are no global IDs, it is useful for leaders and groups to have names. A processor can randomly choose an id such that, with high probability, no processor within a two hop radius has the same id. If a processor chooses an id from $[0, \rho_{max}^4)$, the probability that another processor in the two hop radius chooses the same value is less than $(1/\rho_{max}^3)$ (i.e. very small). The two hop uniqueness is important from a follower's point of view, since it may belong to more than one group and therefore need to distinguish between two or more leaders.

A leader can broadcast its id along with the recruit message in the clubs algorithm. In the case of a collision, a follower broadcasts a request after the clubs algorithm has completed to determine which clubs it belongs to. The follower can use the simple strategy of broadcasting a request, waiting a random delay and then trying again. If the random delay is chosen from the range $[0, \rho_{max})$ and all processors are using the same strategy and range, it can be shown that the expected number of trials after which a processor i gets through is $(\frac{\rho_{max}}{\rho_{max}-1})^{d(i)-1}$, which is less than 3 [11]. We will refer to this broadcast strategy as *random-wait* protocol. This and alternative point-to-point protocols (like exponential backoff and CSMA) for broadcast networks are presented in detail in [11]. Resolving collisions using random-wait adds $O(\rho_{max})$ expected steps to the algorithm and increases the number of messages by $O(\mathit{conflicts})$.

4.2 Asynchronous Processors

The clubs algorithm is simple to implement in an asynchronous environment because no synchronization between processors is required during the execution of the algorithm. The only synchronization required is at the beginning of the algorithm. Processors may start at different times, so if a processor announces leadership while its neighbors are not listening there will be unnecessary leadership conflicts. A processor needs to wait only for its immediate neighbors to complete previous tasks, before choosing a random number and counting down. This can be determined if all processors broadcast a done message after completing previous tasks. A processor should continue to listen for recruit messages even after reaching the timeout. A processor can also locally determine when the algorithm is complete if its neighbors broadcast similar done messages after reaching their timeouts.

Theorem 3: *The clubs algorithm completes in $D + R$ steps, where D is the delay between when the first processor starts counting down and the last processor starts counting down, for asynchronous processors. The expected number of leadership conflicts $E(\text{conflicts})$, is still at most $(\frac{d_{avg}}{2R})N$, for the aynschronous amorphous computer.*

Proof 3: Let $delay_i$ be the time between when the first processor starts counting down and a processor i starts counting down[1]. Each processor can be treated as choosing a random number r_i and a random offset $delay_i$ which is equivalent to a processor starting at time 0 choosing from the range $[0, D + R)$ where D is the maximum delay. Hence each processor will have either declared leadership or become a follower by $D + R$ steps. Therefore all processors will belong to a group at the end of $D + R$ steps.

Let processor i start counting down d timesteps before some other processor j. Then the choices of R for which the two processors can conflict must lie within the range $[d, R)$ for processor i and $[0, R - d)$. The probability of having a conflict is $(\frac{1}{R^2}(R - d))$. If we allow an adversary to choose the delay, so as to maximize the probability of leadership conflicts, we see that the probability is maximized when the delay $d = 0$. The probability of a leadership conflict decreases when the processors are not synchronous. Hence the expected number of collisions is still at most $(\frac{d_{avg}}{2R})N$.

A similar argument can be made when the processor speeds are different. Let processor i operate at a speed S times that of processor j, where $S \geq 1$, and let both processors start at time 0. Then collisions will occur only when processor j chooses values from the range $[0, \lfloor R/S \rfloor)$ and processor i chooses values from $[0, R)$ that are divisible by S. The probability of collision is $(\frac{1}{R^2}\frac{R}{S})$. Again this is maximized when $S = 1$, i.e. the speeds are the same. The reason is that the range of random choices over which a conflict can occur has decreased. The time taken by the algorithm to complete is the time taken by the slowest processor to count to zero.

Treating messages as having non-zero transmission times also does not affect the expected number of conflicts. Since a processor knows not to send a message while it is currently receiving one, conflicts will only occur when two adjacent processors choose the same time to start declaring leadership. If one processor precedes the other, then the second processor will sense that the channel is busy before sending a message and abort its claim for leadership. Thus, leadership conflicts will not occur due to partially overlapping recruit messages.

4.3 Processor Failures

Only failures of the leaders affect the clubs algorithm. Until now we have assumed that processors are reliable. A processor may stop executing at any time (stopping failures). The clubs algorithm is robust to most failures because processors execute relatively independently and the communication is simple. Processors failing before finishing the countdown or after becoming followers do not affect the algorithm. However, leaders guarantee that a group has local routing and a maximum diameter of two hops. If a leader fails, the group may potentially become disconnected and the diameter may increase, violating two group properties.

Adaptive Clubs: The clubs algorithm can be extended so that whenever a leader fails, its followers rerun the clubs algorithm to elect a new leader(s). After completing the initial group formation, each leader periodically reasserts its leadership. If a follower does not hear a leader for several time intervals (T_1), it broadcasts a challenge to the leader. If it does not hear a response from the leader within a certain timeout period (T_2), then it assumes the leader is dead and broadcasts a message declaring the leader dead. Upon hearing this message, only the processors that do not belong to any group need to rerun the clubs algorithm. Hence the overlapping groups add robustness by decreasing the effect of

[1] We assume that delay includes the time required to retransmit done messages.

the failure of a single leader.

Correctness: If a leader dies, members will eventually detect it. They will either hear a declaration that the leader is dead, or timeout themselves and challenge the leader. If they do not belong to any group, then they will compete using the clubs algorithm, until all of them belong to some group. Even if a leader is not dead, a processor may falsely think it is dead (it challenged the leader and for some reason did not hear the response within the timeout T_2). As a result, several processors may rerun the clubs algorithm and create new groups and later realize that the leader is alive. This results in unnecessary clubs (and leadership conflicts) but still guarantees that all processors belong to some club.

Parameters: The rate at which the leader reasserts leadership depends on the expected time to failure as well as how soon the application needs to detect a failure. The leader does not have to send a special message, as long as it broadcasts some message within the specified period. To challenge a leader or to respond to a challenge, a processor can use the random-wait protocol or exponential backoff. The timeouts T_1 and T_2 depend on the expected time for messages to get through, given the particular broadcast strategy. The timeouts should be chosen so that probability of false death declarations is low and the incidence of unnecessary leadership conflicts is low.

Thus, adaptive clubs can reorganize to accommodate failures. This method can also be used for accommodating new processors into an already existing clubs structure.

4.4 Conclusions

The clubs algorithm produces groups of diameter two hops in time proportional to the local neighborhood size. The algorithm does not require point-to-point communication or synchronous processors. Rather it takes advantage of those properties that are generally difficult to deal with. This is achieved by relying on the local broadcast mechanism rather than point-to-point message exchanges and allowing leadership conflicts to occur probabilistically. Hence complex synchronization is not required and the algorithm performs efficiently in both synchronous and asynchronous settings.

In addition the algorithm satisfies several other constraints that that generally occur in large distributed systems. The algorithm does not require global IDs, relying on randomization instead, and does not use require that processors know the diameter of the network or the number of nodes. The algorithm can be extended to reorganize groups automatically in response to the processor failures or the addition of new processors.

5 Physical Properties of Clubs

A distinctive property of an amorphous computer is that it has geometry as well as topology. The geometry is derived from the communication geometry and the space within which the amorphous computer is embedded. The geometry can be used to derive additional properties of the clubs algorithm.

Assuming that there are *no leadership conflicts*, we can derive two bounds on the clubs algorithms:

Theorem 4: *The maximum number of clubs formed is fixed for a given surface area and communication radius, and does not depend on N.*

Theorem 5: *The degree of a club is at worst 24.*

The proof for both theorems is based on modeling a processor as a circle of radius $r/2$, centered at the processor (figure 4). An edge implies a maximum physical distance of r. Therefore the circles will overlap if and only if the processors are adjacent.

Proof 4: If there are no leadership conflicts, then all the leaders are non-adjacent. The maximum number of clubs that can be formed in a given area is the same as the maximum number of leaders one can place in that area without violating the constraint that no two leaders be adjacent.

If we model each leader as a circle of radius $r/2$, as described before, the problem of finding the maximum number of clubs can be restated as a packing problem i.e. what is the densest packing of non-intersecting circles of radius $r/2$ in a plane. In a two dimensional plane, the densest packing of circles is a hexagonal packing. Hence the densest packing of leaders is a hexagonal lattice where the distance between two adjacent lattice points is r. This implies that, for a given surface and a given communication radius, the maximum number of clubs is fixed irrespective of N and is equal to the number of grid points on

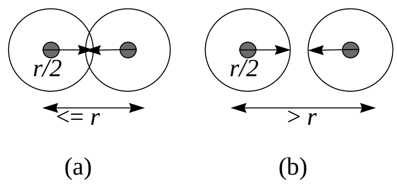

Figure 4: A node is modeled as a circle of radius $r/2$. (a) If the circles intersect, the nodes are adjacent. (b) If the circles do not intersect, the nodes are not adjacent.

the hexagonal lattice.

Proof 5: If we consider each club to be a node and clubs that overlap to be adjacent, we can talk about the graph induced by the clubs. The degree of a given club is the maximum number of clubs that it can be adjacent to. In order for two clubs to be adjacent, or overlap, their leaders must be less than $2r$ apart (by the triangle inequality). Hence for a particular leader, all leaders within the circle of radius $2r$ are potential neighbor clubs. However leaders must be at least r distance apart. If we model the neighboring leaders as non-overlapping circles of radius $r/2$, then all the circles must fit within an annulus of inner radius $r/2$ and outer radius $(2r + r/2)$, centered at the given leader. Since each circle occupies an area of $\pi r^2/4$, no more than 24 non-adjacent leaders can be placed in the annulus. Therefore a leader can have no more than 24 neighboring leaders and the degree is at worst 24. Hence we see that the degree is upper bounded by a constant and does not depend on the area or N. Using a similar argument a processor can belong to no more than 9 clubs.

If the number of leadership conflicts is small, these theorems will still hold with high probability. In Section 7.1 we provide an extension to the clubs algorithm that guarantees no leadership conflicts.

Both bounds are very useful for designing algorithms on top of clubs. The first bound provides an estimate of the number of groups to expect, given the area and communication radius. The second bound tells us that the graph induced by the clubs has small degree. The decomposition of processors into groups with small diameter such that the graph induced by the groups has small degree is very useful in the design of many algorithms [3, 5].

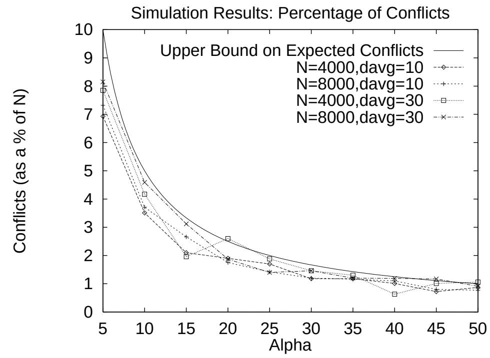

Figure 5: Leadership conflicts as a percentage of N vs. α for different (N, d_{avg}) pairs

6 Simulation Results

We simulated an amorphous computer running the clubs algorithm with 1000, 4000 and 8000 processors, each with average neighborhood sizes of 10, 30 and 50, for different values of α. The processors are uniformly distributed over a unit square surface. The processors are asynchronous with a small delay (less than a message transmission time) and the message transmission time is a hundred clock cycles. A collision occurs if messages partially overlap. The processors choose a random value from the range $[0, R)$ where $R = \alpha d_{avg}$ and multiply it by the transmission time.

The simulation results correspond well with the analysis for the synchronous case. Graph 5 plots the number of leadership conflicts, as a percentage of the total number of processors, for four (N, d_{avg}) pairs[2]. Each data point is averaged over several runs. In each run the layout of the processors is changed and new random values are chosen. Therefore there is significant variation in the number of conflicts in each run. As we can see, the average percentage of conflicts varies as expected with α and does not seem to be affected by N or d_{avg}.

In graph 6, we have plotted the number of clubs formed in the same experiments against the communication radius. As we see, even with up to 10% conflicts, the number of clubs varies with the radius close to expectations. Each data point is averaged over twenty runs with different processor layouts, however there is little variation in the number of clubs formed in each run. This is be-

[2]The remaining curves also occupy the same region, so for clarity we have plotted only 4 of the 9 curves

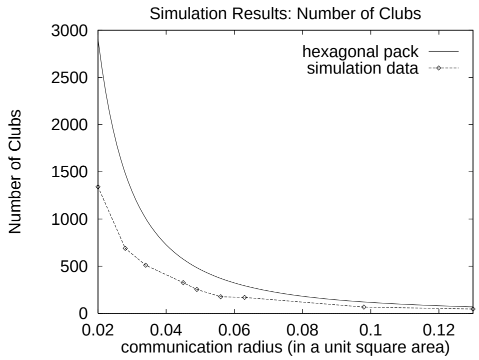

Figure 6: Number of clubs formed vs. communication radius for processors distributed in a unit square

cause the number of clubs depends more on the surface and communication geometry than processor layout or initial random state. In all of the runs, no processor belonged to more than 5 clubs, in spite of the conflicts.

7 Extensions

In this section we describe the extension of the clubs algorithm to solve problems like MIS and $\Delta + 1$ vertex coloring.

7.1 Maximal Independent Set

A maximal independent set (MIS) is a set of nodes in a graph such that no two nodes in the set are adjacent (independent) and no more nodes can be added to that set without violating independence (maximal). Computing the MIS of the graph induced by a network is a useful tool for solving many distributed computing problems [10, 9].

In an amorphous computer, the set of club leaders is almost an MIS on the amorphous computer graph. All non-leader processors (followers) are adjacent to at least one leader, and hence cannot be added to the MIS. Most leaders are non-adjacent, except those that have a leadership conflict. Solving for a MIS is equivalent to guaranteeing that no leadership conflicts occur.

This can be achieved by running several rounds of the clubs algorithm. After each round, there is a conflict-detection stage during which leaders determine if they experienced a leadership conflict. If so they abdicate leadership and they and their followers compete in the next round of clubs. The algorithm completes when there are no leadership conflicts left. We call this algorithm *clubs-MIS*.

The clubs algorithm guarantees that all non-leader nodes are adjacent to some leader. The clubs-MIS algorithm runs a new round of clubs if any of the leaders are adjacent. The followers of the conflicting leaders are forced to compete as well if they are no longer adjacent to some leader. The algorithm keeps running until there are no conflicting leaders. Hence, the final set of leaders forms an MIS.

Theorem 6 : *The expected time to find the MIS is $O(\rho_{max} \log N)$ in a synchronous amorphous computer*

Proof 6: In each round, the competing nodes choose new random numbers from the range $[0, R)$. Therefore, each round is independent. The upper bound R is chosen to be αd_{avg} and is the same for every round. Let the number of nodes in round k be N_k. In round $k+1$ only the nodes that experienced a conflict will re-compete.

$$\begin{aligned} E(N_{k+1}) &= E(\textit{conflicts} \text{ in round } k) \\ &\leq \frac{1}{2\alpha} N_k \end{aligned}$$

Therefore,

$$E(N_{k+1}) \leq 1 \text{ after } (k = \log_{2\alpha} N) \text{ rounds}$$

Hence, all processors are expected to have been removed from the graph in $O(\log N)$ rounds. Each round of clubs takes R steps where $R = \alpha d_{avg}$. In the conflict resolution stage, only the leaders need to exchange messages to determine if there were any conflicts. Using the random-wait protocol for communication, the conflict-detection stage takes $O(\rho_{max})$ expected steps. Therefore the expected time to find the MIS using clubs-MIS is $O(\rho_{max} \log N)$.

The clubs algorithm uses a small number of messages per round and naturally staggers messages to avoid collisions. Furthermore one can choose α to reduce the number of rounds. This makes clubs particularly suited to the asynchronous amorphous environment. In the asynchronous implementation, there needs to be synchronization at the beginning of each new round to make sure all conflicts have been detected before running the next round. Both synchronization and conflict-detection are expensive compared to the clubs algorithm. α can be chosen to minimize the overall time.

Luby [9] presents an algorithm for finding an MIS, which also takes $O(\rho_{max} \log N)$ time in an amorphous computer. Processors choose a random value and compare it with their neighbors' values. The processors with the minimum values become leaders and remove their neighborhood from the graph. The remaining processors take part in a new round. The algorithm continues until there are no processors left. In this algorithm, each round requires a complete exchange of messages between all neighbors which takes $O(\rho_{max})$ steps. This is difficult to implement efficiently, since processors do not synchronize message sending. Using a protocol like random-wait, a significant amount of time is wasted due to message collisions. There is also no control over the number of rounds. The clubs algorithm is simpler to implement in an amorphous computer and takes advantage of the local broadcast capability.

7.2 $\Delta + 1$ Coloring

Vertex coloring assigns colors to each node of a graph such that no two adjacent nodes have the same color. $\Delta + 1$ vertex coloring implies that the graph is colored with $\Delta + 1$ colors where Δ is the maximum degree of the graph ($\Delta < \rho_{max}$). MIS algorithms can be extended to do graph coloring [7]. In this section we will extend the clubs algorithm to do $\Delta + 1$ graph coloring.

The *clubs-coloring* algorithm proceeds by running multiple rounds of color-picking. After each round of color-picking, color conflicts are detected (i.e. cases where two adjacent nodes have the same color) and only the conflicting nodes participate in the next round of color-picking. The algorithm completes when there are no conflicts, i.e. all nodes have been assigned a valid color.

Color-picking uses a similar countdown mechanism as clubs. Figure 7 presents the code for executing a round of color-picking on a single processor. A processor chooses a random number from the range $[0, R)$. As it counts down silently, it collects colors that it hears from its neighbors. When it reaches zero it chooses the smallest color not chosen by its neighbors and broadcasts that color. Once the round is complete, the processor checks for leadership conflicts. If two leaders broadcasted at the same time (a leadership conflict), they might have chosen the same color. Or some node may not have heard the color they chose due to the collision, and may have chosen the same color. In either case the processors that broadcasted at the same time must renounce their colors and participate in the next round of color-picking. Since processors always choose the smallest color not chosen by their neighbors, and the maximum number of neighbors is Δ, the color values range from 1 to $(\Delta + 1)$.

```
  integer R (upper bound for random numbers)
  list color_list = empty (list of neighbor colors)

procedure CLUB_COLOR_PICKING ()
  t_i := R
  r_i := random[0,R)
  while (t_i ≥ 0)
     if (not_empty(msg_queue))
        newcolor := first(msg_queue)
        insert(color_list, newcolor)
     if (r = 0)
        broadcast(smallest color not ∈ color_list)
     r_i := r_i - 1
     t_i := t_i - 1
```

Figure 7: Algorithm for Color Picking

Theorem 7 : *The expected time for* $\Delta + 1$ *coloring is* $O(\rho_{max} \log N)$ *in a synchronous amorphous computer*

Proof 7: Color-picking is similar to the simplified clubs, *sclubs*, presented in Section 4 because processors that count down to zero do not prevent their neighbors from continuing to count. Hence, if R is chosen to be αd_{avg}, the expected number of conflicts is less than or equal to $(1/2\alpha)N$. By the same argument as clubs-MIS, the number of nodes in each round is less than a constant fraction of the previous nodes, therefore all nodes will be removed in $O(\log N)$ expected rounds. Each round takes $O(\rho_{max})$ expected time. therefore the total expected time is $O(\rho_{max} \log N)$.

8 Example Applications of the Clubs Algorithm

Clubs can be used for task specialization, increased robustness, or resource allocation. In this section we provide three examples of using the clubs to address the issue of efficient communication in an amorphous computer. Several other examples are presented in [5].

The clubs can be used as a higher level point-to-point network. The leaders communicate point-to-point with each other and relay messages to and from their members. The communication between adjacent leaders is accomplished via elected representatives in the overlap regions. This significantly reduces the number of messages and potential collisions. For example a full broadcast operation can be implemented constructing a spanning tree on the graph induced by the leaders. Collisions can be further reduced if leaders run a coloring algorithm to choose non-interfering channels and members choose a single clubs to belong to. Within the group, a leader can poll its members to prevent collisions between members. This is analogous to self-organizing a cellular network with clubs as cells and leaders as base stations. The upper bound of 24 on the degree of a club tells us the maximum number of distinct channels required.

An extension of this idea is to use the clubs algorithm to self-organize a hierarchical network for efficient non-local communication. The group leaders can run the clubs algorithm to form higher level groups. In a separate paper [5] we show how the clubs local leader election mechanism can be extended to form groups of a given diameter h. The efficiency of the resulting network will depend on the number of levels in the hierarchy and the diameter of groups at each level.

It is also possible to use the clubs-based coloring algorithm to create an efficient local point-to-point communication for applications that require frequent local exchanges of values, such as partial differential equation (PDE) calculations and cellular automata style local rules. For such algorithms protocols like random-wait are inefficient due to high percentage of collisions. The $\Delta + 1$ coloring can be used to implement CDMA (code division multiple access) in an asynchronous amorphous computer [11, 6]. In CDMA, messages modulated with different digital codes can be broadcast simultaneously without interfering. Processors use their color to determine which code to listen on. The sender broadcasts using the code of the intended receiver. The number of codes required $\Delta + 1$, which is upper bounded by ρ_{max}. By assigning nearby processors different channels to listen on the probability of collisions can be significantly reduced. However a receiver can still only receive from a single sender at any time, hence it is not the same model as a wired point-to-point network. Nevertheless it can be used to apply point-to-point algorithms more efficiently on the amorphous computer. Our hardware prototype will support spread spectrum CDMA and use this mechanism to assign channels.

9 Conclusion

In this paper we presented the *clubs algorithm* for forming groups in an amorphous computer. The clubs algorithm performs efficiently by taking advantage of the local broadcast mechanism and directly addressing the problem of message loss through collisions. The simplicity of the local leader election mechanism makes it easy to extend to asynchronous processors without complex synchronization. In addition the algorithm does not use global IDs and can be extended to deal with processor failures. The algorithm can also be used in point-to-point distributed environments with similar constraints.

In addition, we derive upper bounds on the number of groups formed and the density of groups formed by the clubs algorithm, using the physical embedding of the amorphous computer. We present simulation results for the clubs algorithm that concur with the analysis. Extensions of the clubs algorithm to solve for a maximal independent set and produce a $\Delta + 1$ coloring are presented. Lastly, we present three examples of applying the clubs algorithm to address communication issues in an amorphous computer.

References

[1] Abelson, Knight, and Sussman. Amorphous computing. *White paper*, October 1995. http: //www-swiss.ai.mit.edu/ ~switz/amorphous/.

[2] Awerbuch, Berger, Cowen, and Peleg. Fast distributed network decompositions and covers. *Journal of Parallel and Distributed Computing*, 39:105–114, 1996.

[3] Awerbuch, Goldberg, Luby, and Plotkin. Network decomposition and locality in distributed computation. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 364–369, October 1989.

[4] Berlin. *Towards Intelligent Structures: Active Control of Buckling.* PhD thesis, MIT, Department of Electrical Engineering and Computer Science, May 1994.

[5] Coore, Nagpal, and Weiss. Paradigms for structure in an amorphous computer. AI Memo 1614, MIT, 1997.

[6] Dixon. *Spread Spectrum Systems with Commercial Applications.* John Wiley & Sons, New York, 1994.

[7] Goldberg and Plotkin. Parallel $(\Delta + 1)$-coloring of constant-degree graphs. *Information Processing Letters*, 25(4):241–245, June 1987.

[8] Hall, Crawley, Howe, and Ward. A hierarchic control architecture for intelligent structures. *Journal of Guidance, Control and Dynamics*, 14(3):503–512, 1991.

[9] Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal of Computing*, 15(4), November 1986.

[10] Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers, Wonderland, 1996.

[11] Tanenbaum. *Computer Networks, second edition.* Prentice-Hall of India, New Delhi, 1990.