

Find a Stochastic Path in a Robust-First Distributed System

Xinyu Chen

University of New Mexico, Albuquerque, New Mexico 87131
xychen@cs.unm.edu

Abstract

To find a robust-first communication approach in distributed systems, we present a case study of one stochastic path-finding in the Movable Feast Machine(MFM). The model let computation requests follow some signals to find the service providers. Instead of using location-dependent addresses or relative coordinates, these signals can indicate the gradient of themselves so that they form stochastic paths towards the source. Such paths are self-constructed and self-repairable. This case study helps to extend communication distances outside Event Windows. This is a step towards a more robust distributed system.

Introduction

determinism is always preferable because it seems simpler and more efficient than uncertainty. Traditional computer architectures based on this principle have helped to deal with many real-world problems. But the simplicity and the efficiency here are not assured because the real world is full of uncertainties. Such uncertainty comes from the variety and redundancy of this world. Not a single event can destroy the ecosystem. In this sense, the uncertainty represents a robustness. This feature of the ecosystem is appreciated by computational systems. It is time that we begin to think in a robust-first way to make life-like systems. Then we can truly accomplish scalable and high-performance computations.

Our case study is designed on such a life-like system, the Movable Feast Machine (MFM). The Movable Feast Machine is an indefinite scalable distributed system with the robust- first thoughts. To achieve the indefinite scalability, the MFM is designed to distribute processors and memories in spatial grids while still processing regular structures such as short spatial dependencies and error tolerance (Ackley et al, 2013) The computation tasks are fulfilled by asynchronous Cellular Automata, also called *Atoms*, which are instances of *Elements*. A Element in the MFM is like a class in an object-oriented language. It describes the behaviors and other properties of Automata. When given an event, an Atom can operate a limited unnumber of sites called the *Event Window* in a parallel manner. The Event Window defines both temporal and spatial restrictions. It restricts Atoms to

interact with a limited number of objects. Thus every objects in the MFM are equal and they are only responsible for their own behaviors. There is no super unit that controls global issues. This life-like feature renders computations on the MFM the ability of self-organize and self-repair and eventually the ability of being robust. However, to achieve such robustness in the MFM is not easy. We have to change our serial thinking to spatial thinking and random thinking. The behaviors of Automata must be reasonable in the context of their limited knowledge of the environment and limited time constraints within an event.

Path-finding problem is a useful subject both in real-world applications and theoretical researches. These fields include transportation, communication, game design, computer networks and so on (Chen, 2005). However, with the light of robust-first thinking, we need to reconsider this problem under spatial and temporal distributed conditions (Lamport, 1978). The uncertainty of the order of events and the asynchronous feature in distributed systems give many path-finding algorithms great challenges. Without global addresses, how can algorithms construct paths? On the other hand, with some global addresses, can these systems really be scalable (Ackley et al, 2013)? Some primitive methods that we try to avoid under the efficient-first goal need reevaluations under the robust-first concerns. Such methods include but not limited to bubble sort and broadcast-ing communications. Not surprisingly, some simple mechanisms work robustly in the biology environment. This encourage us to look for some simple and robust path-finding approaches.

Model Descriptions

In biology world, we find such effective examples of path-findings. An firefly sends out flash signals to attract partners, an E-Coli takes biased random walks towards food source, neuronal growth cones are guided by both chemoattractants and chemorepellents *in vitro* (Philipsborn,A and Bastmeyer,M, 2007). All these reminds us there are some path-finding mechanisms based on the gradient of physics signals or chemical medium. In this case study, we design

a signal-emit and gradient-check path-finding model with respect to those biology analogs. This method also looks like broadcasting communications. But in fact, the gradient-check part is quite different. To describe our model better, we abstract the real world scenario to the interactions between three participants: *Requests*, *Signals* and *Providers*.

Requests diffuse in the given space of the MFM until they find a Provider and disappear. We call this the Request is satisfied. Signals can also diffuse. As pheromones, they have an evaporate rate (Velloso, B and Roisemberg, M, 2008). The whole collection of Signals forms a cloud-like shape. The density of Signals around Providers is higher than the densities on the edge of this cloud. This difference of density can be interpreted as the gradient towards Providers. Our intuitive idea is to let Requests detect and follow this gradient so that they can move towards the locations with the highest signal density. Hopefully, this location is close to Providers so Requests can find them. To detect Signal densities, Requests count the number of Signals they see in the current Event Window and record this number. By the same manner, Signals also count and report the number of fellow Signals in the Event Window. Requests can compare the density they observed with the density a Signal recorded, thus they can decide the approximate direction towards Providers. Due to the asynchronous feature of the MFM, this comparison is not absolutely accurate because Requests are comparing the newest observation with some historical records. Nonetheless, we can show this gradient-check approach still improves the success of Requests' path-finding.

The success of path-findings in this case study is measured by the percentage of the satisfied Requests within a given period. A distance restriction between Requests and Providers is also required. The distance setup is due to the evaporation of Signals. They can not cover a big area before they disappear. On the other hand, if they do not disappear, they will sparsely located in the universe. In both cases, they cannot indicate the gradient well. The time restriction is also necessary because Requests diffuse randomly. When only a few Requests are left unsatisfied, the probability for one of them to find the right location drops. It always takes very long time for the last Request to be satisfied. We only need to show a trend of the path-finding without having to wait for the last job to finish. The relation between the population of Signal Atoms and Request Atoms can help us to make some quantitative measurement about this stochastic path-finding approach.

This case study models the interactions between the Elements of Request, Signal and Provider. To make some simplifications, we set Providers in a fixed area and let Requests to search for Providers. In real-world applications, both requests and providers are dynamic. We can see this mobility from the usage of smart phones and other mobile devices. Requests and providers may continuously vanish and appear

(Braubach, L and Pokahr, A, 2007). By the contrast of these moving Requests and static Providers, it seems that Requests are captured by Providers. In fact, Requests are the positive part in this interaction. In the MFM, such interactions are implemented by Element behaviors. One thing we do differently on the MFM is that we do not have global algorithms. The path-finding process is a series of interactions between the Elements. In Figure.1, we show the process of this stochastic path-finding.

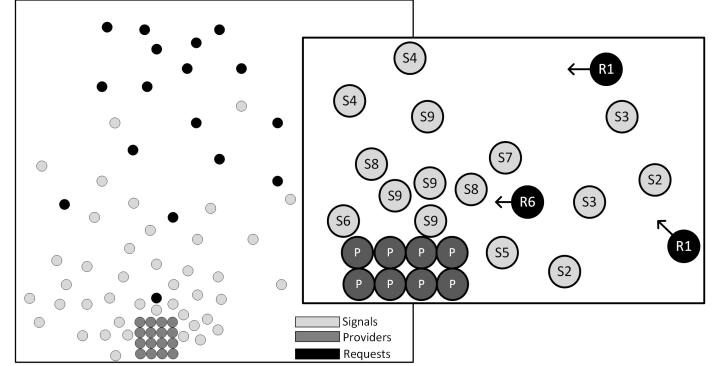


Figure 1: How *Requests* follow *Signals* to find *Providers*. The left: an overall view of this path-finding process. The right: a zoom in view. Light grey dots :Signals *S*. Dark grey dots:Providers *P*. Black dots:Requests *R*. The numbers represent the density that Atom observes. The arrows indicate a swap direction.

Provider Behaviors

Providers' primary behavior is to send out Signals. This *SignalEmit* behavior decide how often a Signal Atom is created and how big the evaporate rate is. We can adjust these two parameters to get various signal populations in the experiment universe. The *EmitOdds* is between 0 to 10. The chance of emission is $EmitOdds \times 10\%$. Providers create no Signal in an event if the *EmitOdds* is zero. They create one Signal in an event if the *EmitOdds* is set to 10. Although the evaporate rate is set by Providers, the actual behavior of evaporation belongs to Signals.

Signal Behaviors

Signals plays the roles of flashlight or chemoattractant in this stochastic path-finding. To examine the behaviors of Signals, we should distinguish the behaviors of a single Signal Atom and the behavior of the whole Signal collection. A single Signal Atom has three behaviors :*CheckDensity*, *Evaporate* and *Move*. When a Signal Atom gets an Event, it first checks how many other Signals are in this Event Window. The other two behaviors depend on the observation of fellow Signals. We set the initial density record of Signal

Atoms to two even when the Event Window contains only one Signal Atom. The observation of signal density is

$$0.1 \times NumProvider + NumSignal + 2 \quad (1)$$

The default density number starts from two based on the following consideration. The Signal Atom itself should at least count for density one when there is no fellow Signal in its Event Window. But when a Request meet a single Signal, the Request's observation is also density one. To let the Request move towards this Signal, we raise the initial density. This situation happens a lot at the edge of cloud and the initial value of two helps Requests to enter the cloud quite well. Also we reward Signals near Providers a bit by count Providers as Signals too. That is why we add $0.1 \times NumberProvider$ to the signal density record.

The next behavior *Evaporate* is based on the *EvaporateRate* set by Providers. The actual probability of *Evaporate* is the inverse of the product of this base number and the observation of signal density. With this formula, Signals that are far from Providers will evaporate faster than those are close to Providers. The probability of evaporate is

$$PrbEvap = \begin{cases} \frac{1}{|EvapRate - Obsv|}, & obsv \leq 2 \\ \frac{1}{|3 \times EvapRate - Obsv|}, & otherwise \end{cases} \quad (2)$$

Governed by this probability, Signals destroy themselves before moving further.

The last behavior of single Signals is *Move*. If a Signal doesn't destroy itself, it decides how to move. At this moment, a Signal Atom has a record of density to *report* to Requests. If it meets no Request in the current Event, it diffuses. Otherwise it *stays* and *reports* the density record. Here we use the verb *report* as if a Signal Atom can write this record to Request Atoms. In fact, this density record are stored inside Signal Atoms and Request Atoms will read out this information. This stay behavior is designed because we want the difference between the observation of Requests and *reports* of Signals to be as small as possible. If we let Signals diffuse without staying, Requests can only read historical reports about some events ago or even somewhere else. However, due to the asynchronous feature in the MFM, this design is only an attempt to keep these differences close. There is a side effect of this stay behavior and we will discuss in the discussion section. A non-stay behavior is also interesting and needs some studies in the future.

Behaviors of Signals are described in Algorithm.1. A single Signal takes random walks until it dies. The longest distance from Providers it can walk is a function of the number of events it lives. This distance is approximately a function of the square root of the life length.

As a result, we have the behavior of the entire collection of Signals. Viewed as a group, all Signals form a cloud-like shape. The center of this cloud is Providers. The range is bounded by the longest distance that Signal Atoms can walk

Algorithm 1 Signal Behavior

```

1: procedure SIGNALBEHAVIOR
2:   numFellows  $\leftarrow$  checkDensity(eventWindow)
3:   evaporate(evapOdds, numFellows)
4:   if not Evaporate then
5:     if RequestsExists then
6:       Stay
7:       Diffuse

```

away from the center. The signal density in the center area is much higher than the density on the edge due to the different probabilities of evaporation. We can manipulate the parameters of *EmitOdds* and *EvaporateRate* to alter the range of the cloud. If we tune up the *EmitOdds*, Signal population will go higher. If we tune up the *EvaporateRate*, Signals on the edge will be less likely to destroy themselves. Thus the range of cloud expands.

Request Behaviors

Requests behave quite similar to Signals. They also have three primary behaviors : *Evaporate*, *CheckDensity* and *Move*. For Requests, the most important thing is to find Providers and get satisfied. As soon as a Request Atom is *captured* and satisfied, it disappears. In this sense, Requests can evaporate too. This evaporation is not affected by some probabilities but by the presence of Providers. We design this *Evaporate* behavior happens if Providers are found at the distance of one: a Request Atom can only be captured by an adjacent Provider Atom. This restriction is based on the assumption that too much Signals in the center area would hinder Requests' *Evaporate* process. In specific applications, we can remove this limit and allow Requests to find Providers within the full Event Window range.

Like Signals, the Requests' *CheckDensity* behavior is also a precondition for their *Move* behavior. When a Request Atom is given an event, it counts the number of Signals in the Event Window. If there are some Signal Atoms, the Request can read their density reports and choose a Signal with the highest density record. This Signal's location indicates a potential gradient towards Providers. Because this comparison happens in the Request's Event, Requests will always have the accurate observation of signal density in this Event Window. While reports from Signals could be observations of some events before and even somewhere else.

The *Move* of Requests contains three sub-behaviors: *Swap*, *TryMemory* and *Diffuse*. *Swap* happens when a Request Atom finds a Signal's report is higher than its own observation. In this case, the Request Atom swaps location with that Signal Atom. At the mean time, it remembers the offset of this swap. The memory of this *recent swap offset* can help Requests to take *TryMemory* behavior. If a Request finds no Signal, it has some probability to move according to the memory of the *recent swap offset*. Hopefully they can

find some Signals using this memory. The probability for *TryMemory* is set to 0.5 because We discover some bad luck Requests fly out of the universe. This happens when Requests keep doing *TryMemory* but never meet any Signals again. To avoid this misleading, we let them have a 0.5 probability to clean the memory and *diffuse*. Compared with the *Move* behavior of Signals, Requests' *Move* is more like an E-Coli's movements. They take biased random walks which are influenced by the presence of Signals(Kardar,M, 2014).

Algorithm2 describes the behaviors of Requests.

Algorithm 2 Request Behavior

```

1: procedure REQUESTBEHAVIOR
2:   if ProviderExists then
3:     Evaporate
4:     Return
5:   if notEvaporate then
6:     numSignal  $\leftarrow$  checkDensity(eventWindow)
7:     if noSignal then
8:       TryMemory
9:       Return
10:    Swap(HighestDensitySignal)
11:    Remember(SwapOffset)

```

Experiments and Results

The universe of our case study is the small size MFM with 2×2 tiles, 2304(48×48) sites. The distance between the start site of Requests and Providers is 40 vertical sites. In each run, we put 100 Requests into the universe. The Providers occupy a 5×5 sites' area. Our initial experiment aims to set up the baseline with *EmitOdds* set to zero. We can evaluate the effectiveness of our stochastic path-finding approach by comparing the Requests' population with this baseline. We test 10 runs in which Requests move randomly with zero *EmitOdds*. After 6 KAEPS, the percentage of satisfied Requests has an average of 19.4% with a standard deviation of 5.78%. With the same configuration, our second experiment has an *EmitOdds* of 10 and *EvaporateRate* of 7. With these parameters, the average population of Signals in the universe of 10 runs goes to 3.57% after 6 KAEPS. This amount of Signals already form a small-range cloud. It takes Requests about 1 KAEPS to enter the edge of this cloud. But after entering the cloud, the path-findings by gradient-check is quite quick. This experiment shows a big improvement in the effectiveness of Requests' path-finding. The average percentage of satisfied Requests grows to 89.7% with a standard deviation of 2.45%. Figure.2 shows that Signals helps Requests to locate Providers more successfully.

In the following experiments, we fixed the *EmitOdds* to 10 and adjust the *EvaporateRate* to obtain different Signal strength. The Signal population rises as the *EvaporateRate* goes up. When we set the *EvaporateRate* to forever, Signals will not die. The mean Signal population goes up to 41.10%

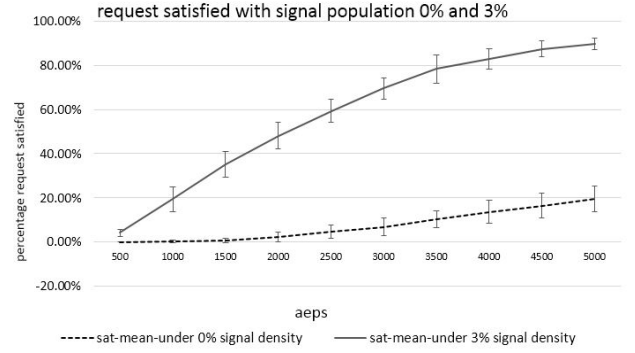


Figure 2: Comparison of successful path-findings with low strength of Signals. The x-axis: the time line of experiments. The y-axis: percentage of Requests satisfied. After 5 KAEPS, the effectiveness of path-finding under low strength Signals is more than 4 times of that without Signals

after 6 KAEPS with 1.79% standard deviation. Such strong Signals form a vast cloud covering the entire universe. Requests can be easily attracted into the edge of this cloud. However, they get lost in the middle of too many Signals. Due to Signals' *Stay* behavior, clusters are formed in some areas. Requests keep continuous swapping with Signals and they do not make real progress in getting closer to Providers. In Figure.3, we compare the effectiveness of path-findings under different signals strength. With a very high Signal population, gradient becomes ambiguous. Especially in the center area of the cloud. Requests are trapped within a short range near Providers but they cannot be satisfied.

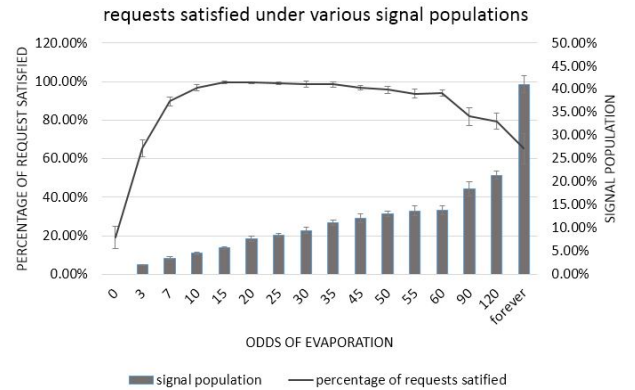


Figure 4: Requests satisfied under various signal populations. The x-axis: *EvaporateRate*. The left y-axis: percentage of Requests satisfied after 5 KAEPS. The right y-axis: population of Siognals represented by percentage of the universe after 5 KAEPS. The effect of path-finding is with signal population between 6% to 11%. The effectiveness drops when signal population goes up to 40%

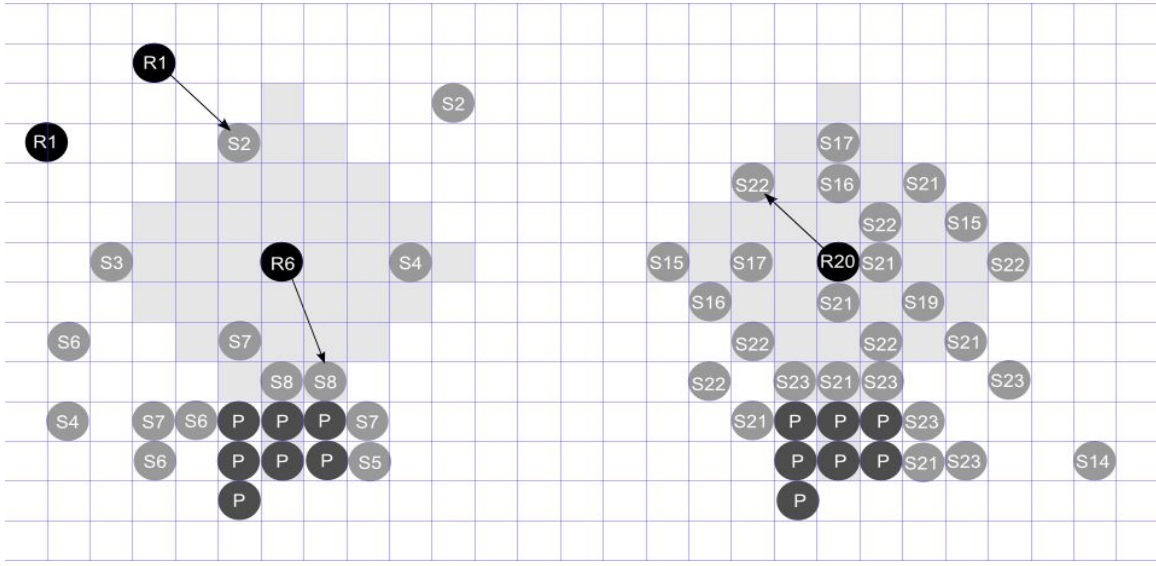


Figure 3: Requests get lost under high Signal population. The left: path-finding is effective with low Signal population. The right: a Request Atom swaps away from Providers under high Signal population. Faint grey grids: the Event Window of the Request Atom in the center. Light grey dots :Signals S . Dark grey dots: Providers P . Black dots: Requests R . The numbers represent the density that each Atom observes. The arrows indicate a possible swap direction.

The effectiveness of path-finding under different Signal strength is shown in Figure.4. The data is collected from 160 runs(Each *EvaporateRate* has 10 runs). The results show that this stochastic path-finding approach works well with moderate Signal strength. Under our distance and time settings, about 99% of the Requests are satisfied when Signal population is between 5.75% and 11.2%. At very high Signal population, the percentile of Requests that are satisfied drops to around 65%.

Discussions

The experiments show that with moderate amount of Signals, Requests can find Providers more successfully. In this section we will first discuss the robustness of this stochastic path-finding method. It is difficult to make quantitative measurement about the robustness by the data we collected in the above experiments. But it is easy to observe the qualitative effects. To test this, we add some obstacles into the initial layout of our experiments. These obstacles is constructed by two built-in Elements in the MFM: Wall and Block. The name of these two elements is quite clear that they don't diffuse. We use Blocks Atoms and Walls Atoms to separate the universe into Four quadrants. Figure.5. shows that obstacles do hinder the path-findings if Providers emit no Signals. But the effectiveness of our Signal cloud does not significantly affected by these obstacles. The mechanism behind this is also easy to understand. The *Swap* behavior of Requests helps them to go through Walls and Blocks. So although these obstacles alter the distribution of Request

Atoms and Signal Atoms, they do not really intercept the stochastic path.

Although the model has some robustness, one aspect of this path-finding still needs more discussions. We observed that as the population of Signals rises, the path-finding becomes less effective. The most obvious observation is the clusters formed at some areas. A bunch of Signal Atoms surround a Request Atom. The Request Atom keeps swap with one of the Signal Atom and other Signal Atoms remain relative static in a fixed area. We attribute this congestion to the *Stay* behavior of Signals.

We first assume the distribution of Requests and Signals is even throughout the whole universe. At the start of our experiments, there are 100 Requests in the universe of 2034 sites. The overall population of Requests is about 0.5%. So there is less than 0.2 Requests in the Event Window of a Signal Atom. In this case, a Signal Atom will meet one Request Atom after at least 5 events. According to the behavior of Signals, we assume a Signal Atom can diffuse in the first 4 events and stay at the 5th event when it meets a Request. With the same method, we can also estimate the average number of Signals in a area with the size of a Event Window. At a moderate signal strength, the population of Signals remains between 6% to 11% in the universe. Apply this population to the size of a Event Window(40 sites), we can expect about average 2.5 to 4.5 Signals in the Event Window. This number can be applied to both the Event Window of a Request Atom or a Signal Atom. So the expected signal density observed by a Request Atom is about 3. While

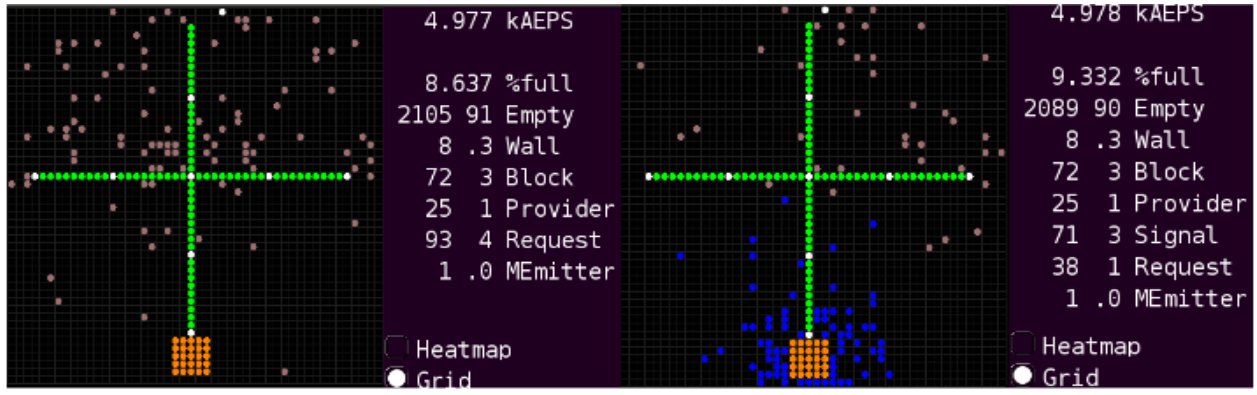


Figure 5: Robustness of this path-finding approach with some obstacles placed. The left: path-finding is significantly hindered by obstacles without Signals. The right: path-finding is slightly affected with moderate Signal population. Blue dots :Signals. Orange dots:Providers. Grey dots:Requests. Green Crosses: Blocks and Walls. The numbers represent the populations in the universe.

this expected observation for a Signal Atom is about 5(We add the actual existing 3 fellow Signals to the default density number of 2, then the Signal Atoms will have a density record around 5).

Now we can make some improvement to our assumptions by considering the distribution of Signals are uneven. On the edge of signal cloud, the density is lower, so is the signal population. By some adjustments, we make a revised assumption. When the overall Signal population is 6% to 11%, we assume that Signal Atoms on the edge of cloud have a density record around 2 and Signal Atoms in the center have a density record around 10. We can further assume these 8 to 10 Signals locate in the Event Window of a Request Atom with about 2 or 2.5 Signal Atoms in each of the four directions. In this case, even near the center of the cloud, a Request Atom can choose a Signal Atom to swap without much confusions. As the Request Atoms keep moving towards Providers, Signal Atoms will not stay too much and they can diffuse properly.

At a very high signal strength, the population of Signals goes above 40% in the universe. We adjust our estimation about the signal density at the edge to 7 to 9 and the density near the center will rise over 30. This means the Request Atom in the central area will see more than 30 Signals in its Event Window. Each Signal will also report a density about 30 fellow Signals. Some Signal Atoms report higher density even they are on the far end of the Event Window. The default density record of 2 will make this report more ambiguous. Influenced by this high density, the Request Atom will swap back and forth with some Signals which report higher density record. It seems they are trapped inside this Event Window. If this happens, Signals will take *Stay* behavior because a Request Atom is jumping around inside their Event Window. That's why some clusters form up. The formation of clusters is very like a deadlock during

interactions between two participants. In the future studies, we need some method to detect this clustering and break the deadlock. Figure.6. shows the real *cluster deadlock* and the density records prove our estimations.

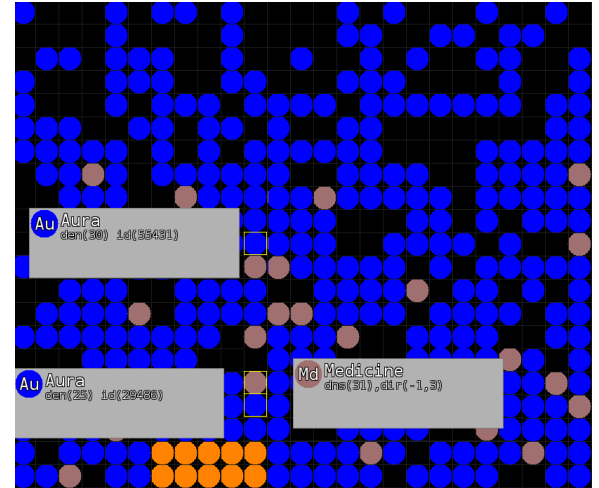


Figure 6: Deadlock clusters under high signal population. Blue dots :Signals. Orange dots:Providers. Grey dots:Requests. Green Crosses: Blocks and Walls. The numbers represent the signal density records stored in the selected Atoms. The coordinate is the *recent swap offset*.

Future Works

In addition to the problems we mentioned in the discussion section, we also have some interesting questions to study in the future. The initial distance between Requests and Providers in this case study is a big limitation. We ran some experiments on the standard size MFM($5 \times$

3tiles, 15360sites). But the result is not good. Many Requests went into the empty spaces where the Signal cloud cannot cover. For now our path-finding is not scalable due to this distance limitation of the Signal cloud. A naive idea is to tune up the Signal strength to extend the range. Other thoughts would be some *Repeaters*. Maybe we can put some Repeaters at the edge of the first generation cloud. Repeaters behave like Providers but they can not satisfy Requests. A second generation of cloud is formed by these Repeaters. Both Signals and Repeaters need to report their generation to Requests. In a broader sense, this generation can indicate a second level of the gradient towards Providers. So we can let this path-finding to be scalable.

The second limitation in this case study is Providers are static. In real-world applications, both Providers and Requests are dynamic. Path-finding under such dynamic condition is very interesting and useful because even Providers themselves do not know their physical positions. It will be worth trying to use this gradient-check path-finding in dynamic context. We want to keep the cloud-like shape of Signals when Providers begin to diffuse. Both *EmitOdds* and *EvaporateRate* need adjustments to maintain this shape. Also we can give Signals more aggressive behaviors in regulating their density. Besides *Evaporate* themselves, Signals can clean other Signals or create new Signals. According to the scalable approach mentioned above, Repeaters is another possible solution.

Another limitation in this case study is Providers are gathered in one area. This simplification actually equals to a single provider which can produce multiple Signals in an Event. However, Providers may be located in different sites and there may be different type of Requests that need different Providers. In the future studies, we can put multiple Providers in the universe. Each Provider can only satisfy one type of Requests. In this case, there would be overlapping clouds of Signals from different Providers. To maintain an effective path-finding process, we need Signals to indicate the type and name of their source Providers. Thus we can still let Requests follow the gradient to find the desired Providers.

Conclusion

This case study is an attempt to build a stochastic path-finding approach in a robust-first distributed system. The behaviors of Request, Signals and Providers are imitations of some biology gradient-check mechanism. The model and experiments have some sense of effectiveness and robustness. This project also provided a good chance to make a better understand of the robust-first consideration of the Movable Feast Machine. The MFM represents a new architecture for distributed computations. This new architecture uses spatial and temporal restrictions to provide an equal and robust computational environment. These restrictions are not limitations. On the contrary, the MFM is an ideal

framework to study life, evolution and other life-like subjects, because the architecture is so life-like itself.

Acknowledgements

I want to thank Professor Dave Ackley at the Department of Computer Science at the University of New Mexico. The same thanks is for Trent Small at the Department of Computer Science at the University of New Mexico as well. Their efforts presented us the life-like Movable Feast Machine. I also want to thank my wonderful classmates in the Artificial Life lectures in the Fall 2014. Without their ideas, crits and comments even a simple case study like this is not possible.

References

- Ackley,D and Cannon, D (2013). A Movable Architecture for Robust Spatial Computing. *The Computer Journal*, 56:1450-1468.
- Chen,Anthony and Ji, Zhaowang (2005). Path Finding Under Uncertainty. *Journal of Advanced Transportation*, 39-1:19-37.
- Lamport,L (1978). Time, Clocks,and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21:558-565.
- Philipsborn,A and Bastmeyer,M (2007). Mechanisms of Gradient Detection: A Comparison of Axon Pathfinding with Eukaryotic Cell Migration. *International Review of Cytology*, 263:1-62.
- Braubach,L and Pokahr,A (2012). Addressing Challenges of Distributed Systems Using Active Components. *Studies in Computational Intelligence*, 382:141-151.
- Kardar,M (2014). Chemotaxis:How a Small Organism Finds a Food Source. [http://www.mit.edu/~kardar/teaching/projects/chemotaxis\(AndreaSchmidt\)/index.htm](http://www.mit.edu/~kardar/teaching/projects/chemotaxis(AndreaSchmidt)/index.htm).
- Velloso,B and Roisenberg,M (2008). Percolation analyses in a swarm based algorithm for shortest-path finding. *Proceedings of the 2008 ACM symposium on Applied computing*, 1861-1865.