Resource Finding and Distribution in an Indefinitely Scalable Machine

Ezra Stallings

University of New Mexico, Department of Computer Science. Albuquerque, NM 87131
vyross@unm.edu

Abstract

A large-scale network for communications or resource sharing is a potentially valuable tool for spatially-embedded distributed computations. We propose a mechanism for building such a network, using a node-based pathfinding system for use in a simulation where a number of nodes attempt to engage in need-based resource allocation. In this paper, we present an "organic" network construction mechanism based on spatially linked lists and diffusive search, and demonstrate its effectiveness at discovering routes and managing resource distribution and message passing, even in a dense, chaotic environment.

Introduction

Massively scalable computer architecture presents many design challenges. One of the most notable of these is the impossibility of global identifiers of any kind, as no matter how many bits you allocate to addressing, you can always eventually run out. One proposed solution to this and other concerns is an asynchronous cellular automata, which is currently being explored as the Movable Feast Machine (MFM) (Ackley et al., 2013).

MFM Overview

In this simulated architecture, there are a series of tiles, each one being run in parallel by any number processors or distinct computer chips. These tiles can be plugged together to create a computer of arbitrary size. Within the MFM there are Elements and Atoms; where an Element is a type of automaton, and an Atom is a specific instantiation of it somewhere in the simulation. There is also a notion of an Event Window, which is an area in which an event can take place; every time an Atom receives an update, it can only affect a small, local area around itself. There are no global coordinates in this system, and in order to promote emergent behavior and to avoid some of the frailties of most traditional computational environments, there is a high degree of spatially distributed computation, and running lots of small events quickly, rather than doing a lot of work inside of individual Atoms.

Some of the consequences of working within this environment are extremely desirable. For example, if part of the machine goes down, experiences latency, or is physically destroyed, the rest can continue to do useful work. If small errors occur on a local level, these errors can (potentially) be smoothed out, as thousands of atoms work together on small parts of a larger problem. And due to the lack of global address spaces and synchronization issues, the machine can be constructed by piecing an arbitrary number of computers together in parallel.

However, there are a number of side effects which make doing "useful" work difficult; the effective speed of light in the simulation becomes the radius of the event window, and each atoms internal space is restrictively small. This has potentially good consequences in terms of designing inherently robust systems, but it does make it difficult for an atom to find another in a potentially sparse environment.

Spatial Computing

These drawbacks make it problematic to establish any kind of large-scale communication or routing networks. Search and network routing algorithms are some of the most important contributions to modern computing, and it seems highly desirable to attempt to adapt these techniques to a new architecture, as well as to conduct further exploration into methodologies for not just adapting old algorithms, but inventing new ones, for spatial computation further in the future(Ackley, 2013b). Even without thinking about strictly spatial computations like the MFM, large-scale computing is becoming increasingly spatial anyway, due to the massive numbers of computers and hard drives involved (Kelley, 2014) in "the cloud".

Generally, traditional search algorithms are designed to find a specific item of interest. In a spatial computing context, it seems more intuitive to consider search from the perspective of an individual atom; it might potentially be interested in any number of things, like resources, other atoms of its type, or in the case of a predator/prey type scenario, it might try to avoid threats or seek out food. In the cloud computing example, we might be storing data in a redundant fashion, and we just need to find one copy or version of it. This kind of peer-to-peer spatial network is increasingly the way of the world; it seems natural to start thinking about it in the extreme, as scalability in search becomes an increasing concern (Ke and Mostafa, 2013)

Similarly, the notion of a network in such a system should be reconsidered from the point of view of an atom within the simulation; it wants to be linked to other atoms, spatially, in order to share resources, communicate, or otherwise benefit from cooperative behavior. It seems reasonable to expect a particularly good network would be one such that there is a large amount of space between each node. In this way, a node can benefit from having plenty of empty space nearby to utilize, while still having the benefits of being a part of a larger system.

Objectives

We sought to grow a network organically inside of the MFM, in which isolated atoms must find and link with each other, and then utilize the network for a simplistic resource distribution technique. Network optimization for resource distribution is an interesting problem (Cui et al., 2006), and while these are in many ways separate problems, it is difficult to do any kind of routing in spatial computation without also performing discovery, and it seemed natural to try to test the results of the search portion of the simulation by imposing some kind of load upon the resulting network.

We set four primary objectives;

Spatial Linking

Our system should establish links between node atoms, while maintaining distance greater than the event window size, in order to create a spatially diverse network.

Optimized Routes

System links should become spatially optimal, with no unnecessary curved paths.

Message Passing

Nodes should be able to utilize the network to communicate some basic information to other nodes.

Demand-Based Resource Allocation

Network should engage in simple resource allocation between nodes.

To provide a reasonable abstraction for a large scale system with supply and demand, we define a *Village* — an immobile site that is capable of both supply and demand — to represent an endpoint of the communications and routing network. For the purposes of this paper, issues such as resource location/regional fertility, colonization, and population growth or decline are ignored. We consider this a reasonable omission, given the scope of the problems already being considered. In order to find each other, we allow Villages to create *Scouts*, which engage in a Brownian-motion inspired random walk, creating links between Villages by creating *Trails*. These Trails are then used to pass resources between Villages.

Metrics

To measure the success of our model at achieving the objectives, we will need to show several things: first, that links do get established regularly; second, that these links self-optimize to at least a reasonable degree; third, that information is able to reach Villages (nodes) from other Villages; and fourth that this network works for basic resource allocation.

For measuring the effectiveness of the resource distribution, we introduce an *attempt* — generated when a Village tries to consume nearby resources — which will result in either a *success event* or a *failure event*. Multiple success events can occur simultaneously, when Villages successfully find many resources at once. We are particularly interested in the ratio of success events to total attempts, since this will be normalized for the amount of attempts made. For simplicity's sake, we will assume that a Village will always consume all nearby resources upon making an attempt, and that even a single resource will prevent failure. Finally, we will divide Villages into high-demand and low-demand categories; because the system is designed to be demand-based, we would expect to see the greatest gains in high-demand nodes.

Networking and Resource Distribution

Working within the constraints of the MFM, we used four Element types to represent the components discussed above: Resources (Res), Villages, Trails, and Scouts. At a high level, Villages create Scouts, and produce and consume Res based on their supply and demand. Scouts leave Trails, and will attempt to find other Villages. The Res element is one of the built-in components of the MFM as described in (Ackley, 2013a).

The Village Atom

Villages are created with a fixed supply of 8, and a randomized demand between 0 and 15. They also have a local demand, representing an average of the demand of Villages they have links to, plus their own. Finally, each Village has a 10-bit ID, chosen randomly. Each time a Village updates, it engages in the following behavior:

1. Probability of creating a new Res atom, and place it randomly in the event window:
$$Pr_{res} = \frac{1}{16 - Supply_v}$$
2. Probability of attempting to consume all nearby Res atoms:
$$Pr_{consume} = \frac{1}{16 - Demand_v}$$

3. If it attempts to consume and is able to find at least one Res within its event window to consume, generate success events for each Res consumed.

4. If it is unable to find any Res atoms, we consider this a failure event.

5. After consumption and creation, the Village will count the number of links it currently has established. If it has fewer than 4, it has a 1% chance of creating a new Scout.

6. Finally, the Village recalculates its local demand.

The Scout Atom

Scouts travel in a randomly selected initial direction at half of the maximum speed possible in the MFM - the event window radius, or speed of light - with a 5% chance of changing direction randomly during an update, and a 5% chance of moving slightly off-direction (stuttering). Every time a Scout moves, it leaves a Trail atom in its previous position. Each Scout has a randomly selected 10-bit ID tag, which is used to prevent paths from interfering with each other, and a 10-bit tag corresponding to its Village of origin. When a Scout scans its event window, if it finds a Village that does not have the same ID as its origin, it will replace itself with a Trail, effectively destroying itself and ending the search.

The Trail Atom

The Trails left by Scouts form a literal trail on the grid, each separated by half an event window. Each Trail atom has a sequential numerical ID of 10 bits, which starts at 0 with the first Trail placed by any given Scout, and is incremented each time a new Trail is placed. All Trails also have a 10-bit ID field for their predecessor and successor, which are the IDs for the Trail atoms placed immediately before and after itself by the same parent Scout. The Trail can use these IDs to find their predecessor and successor in the event window; they use their immediate neighbors to move Res, adjust internal supply and demand counters, and to move themselves to optimize the trail of Trail atoms to create a more direct route. Trails also have 4 bits each to store upstream demand and downstream demand, and 5 bits to store upstream local demand and downstream local demand, which are used to communicate between Villages what their neighbors demand is, as well as to determine which direction the Trails should send Res. Upstream refers to the Home Village; the one at which the Scout that made the Trails originated from. The trail nearest to the upstream Village will have an ID of 0. Conversely, Downstream refers to the direction away from home, and the Trail atom closest to that Village (assuming one exists) will have the highest numerical ID in the path of Trails. The distinction between demand and local demand is the same for Trails as for Villages. Finally, each Trail atom has a single bit each to know if it is an endpoint Trail (has no predecessor or has no predecessor), which endpoint it is (upstream or downstream) if it indeed is an endpoint, and what direction Res should be routed - by default, this is downstream (towards the Home Village).

The behavior of a Trail is more complex than that of a Village or Scout. Each time the behavior function is called, a Trail does the following:

1. Scan the event window, and attempt to find pred and succ, the predecessor and successor Trail atoms.

2. If one or fewer is found, check for a nearby Tower to determine if this Trail is an endpoint.
   (a) If not an endpoint but unable to find both pred and succ, remove self from the simulation.
   (b) If an endpoint, determine which endpoint (upstream or downstream).
   (c) Copy either upstream or downstream demand and local demand, as appropriate, into self.

3. Copy demand and local demand into self; upstream from pred if pred was found, and downstream from succ, similarly.

4. Determine traffic direction by comparing upstream and downstream local demand. If upstream ¿ downstream, set traffic direction upstream, otherwise downstream.

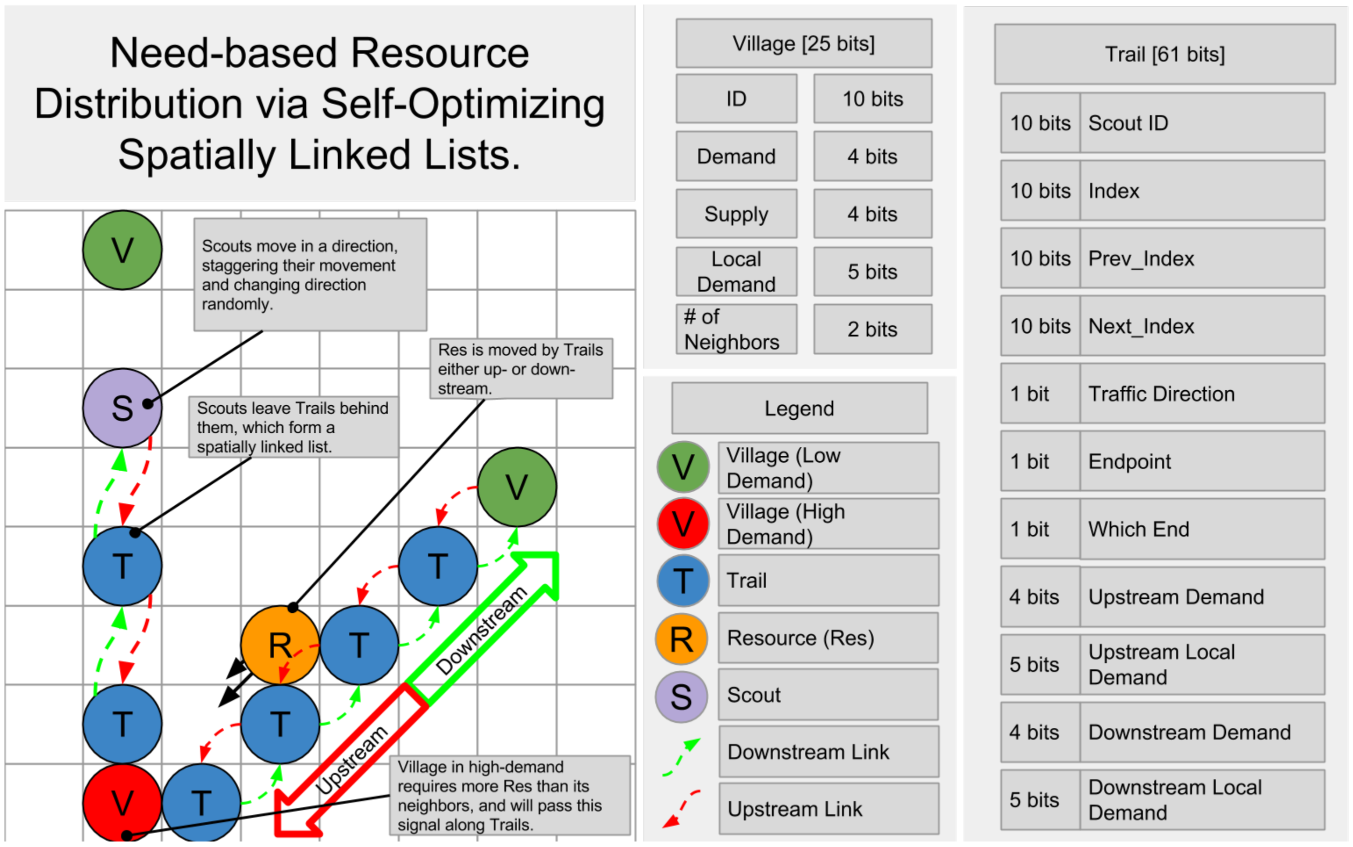
5. Set goal position to pred if traffic direction is upstream, otherwise succ.

6. Move any nearby res to the closest empty position to goal.

7. Move self to a random position halfway between pred and succ.

8. If pred and succ are within 4 of each other, delete self from simulation, and:
   (a) Modify pred to have the successor index of succ,
   (b) Modify succ to have the predecessor index of pred

This can be broadly described as: first, copy upstream and downstream messages into self from appropriate sources, then move any nearby Res according to demand, and finally move self to midway between neighbors. If the neighbors can see each other, have them point at each other, then delete self. The desired outcome of this is to create a spatial, doubly linked list, with endpoints anchored at Villages. If a hanging endpoint exists, part 2.a will execute, resulting in a chain reaction that destroys the list. This is vital, as it prevents the cluttering of the system with useless partial lists. This can occur for a number of reasons: if something interferes with a Village or a Trail in the middle of a list, or if a Scout times out and dies, this will trigger the garbage collection behavior. Having a dedicated space for upstream and downstream messages allows each node to behave independently of the others, with the endpoints responsible for


Need-based Resource Distribution via Self-Optimizing Spatially Linked Lists.
Scouts move in a direction, staggering their movement and changing direction randomly.
Res is moved by Trails either up- or down-stream.
Scouts leave Trails behind them, which form a spatially linked list.
Village in high-demand requires more Res than its neighbors, and will pass this signal along Trails.
Downstream
Upstream
V
S
T
R
Legend
Downstream Link
Upstream Link

Village [25 bits]	
ID	10 bits
Demand	4 bits
Supply	4 bits
Local Demand	5 bits
# of Neighbors	2 bits

Legend	
V	Village (Low Demand)
V	Village (High Demand)
T	Trail
R	Resource (Res)
S	Scout
	Downstream Link
	Upstream Link

Trail [61 bits]	
10 bits	Scout ID
10 bits	Index
10 bits	Prev_Index
10 bits	Next_Index
1 bit	Traffic Direction
1 bit	Endpoint
1 bit	Which End
4 bits	Upstream Demand
5 bits	Upstream Local Demand
4 bits	Downstream Demand
5 bits	Downstream Local Demand

Figure 1: Resource distribution in action: **Far Left:** A Scout lays out Trails as it wanders. **Left:** Trails move Res towards a high demand Village. **Center:** The bit-by-bit breakdown of a Trail. **Right:** Breakdown of a Village, along with the 4 categories of Village based on Supply and Demand.

getting the initial data, and each other Trail only needing to update whatever is most recently available from their immediate neighbors.

Case Studies

In order to test for each of our four objectives outlined above, we devised the following specific case studies, each of which relates to a specific objective or a portion thereof:

A) With 5 randomly placed Village atoms and Scout creation enabled, find the ratio of Scouts that find a Village versus those that timeout and die over a period of 1000 AEPS when:

1. There is a large, circular obstacle with r=20 made of Wall atoms in the center of the simulation.

2. There is no obstacle.

B) For the first above case, verify that paths are in fact forming, and count the number of times a path ceases to self-optimize by forming a hard angle, loop, or switchback, or fails to form altogether.

C) With 5 randomly placed Village atoms of varying supply and demand and Scout creation enabled, verify that information regarding supply and demand is being passed correctly along properly formed paths, and that traffic direction is updating correctly.

D) Find the average prosperity index for 5, 10, 20, 40, 80, 160, 320, and 640 randomly placed Village atoms over a period of 5,000 AEPS when:

1. Scout creation is enabled.

2. Scout creation is disabled.

All simulations were run on the standard-size MFM simulation grid, with dimensions 160 wide by 96 high, spread across 12 tiles. The Epoch length was set to the default value of 100 AEPS. We will define a *High-Demand Village* as a Village atom with demand greater than or equal to 8; a *Low-Demand Village* is a Village atom with demand less than 8. To make the simulation easier to analyze, demand for a Village atom never changes.

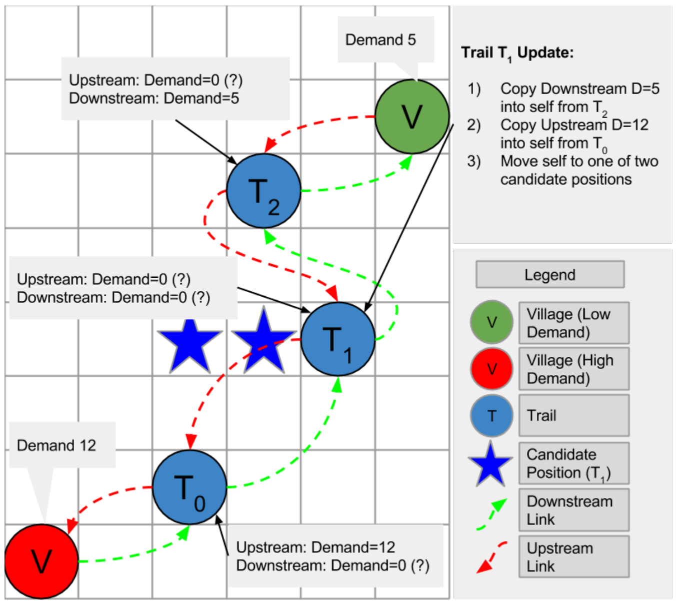


Figure 2: Trail self optimizing and message passing

Results

To verify that Scouts are reaching Villages (case study A), we tallied the percentage of Scouts that reached a Village before timing out in a scenario with 5 Villages, over the course of 1000 AEPS. We found that roughly 10% reached a Village successfully, with the remainder dying of ”old age”. When a large obstacle is placed in the center of the simulation, that percentage actually increased to roughly 15%.

For case study B, we observed the simulation run over the course of 1000 AEPS and determined that out of 21 successful paths, 4 formed a hard angle and did not continue to optimize, all were (eventually) interfered with by another path causing one or both to be destroyed, and none formed loops or switchbacks without either optimizing or failing as a hard angle after solving the loop or switchback. Due to the random choice of optimal position, an angled path will often jitter between possible ”good states”; this jittering in many cases allows a curve or loop to straighten itself properly. Furthermore, for case study C, we colorized the Trail atoms based on traffic direction and were able to see that changes did indeed propagate, sometimes taking as long as 200AEPS to resolve, but sometimes as few as 50. This confirms that supply and demand information is being transmitted, as it is used to determine traffic direction. Due to the difficulty of programmatically establishing any of this information — spatial data is hard to find patterns in — case studies A, B, and C were conducted by hand.

For case study D, we found that, in the case of an individual simulation, the results varied greatly. As shown in Fig. 3, most small simulations with between 5 and 160 villages yielded an overall loss of system success. However, there

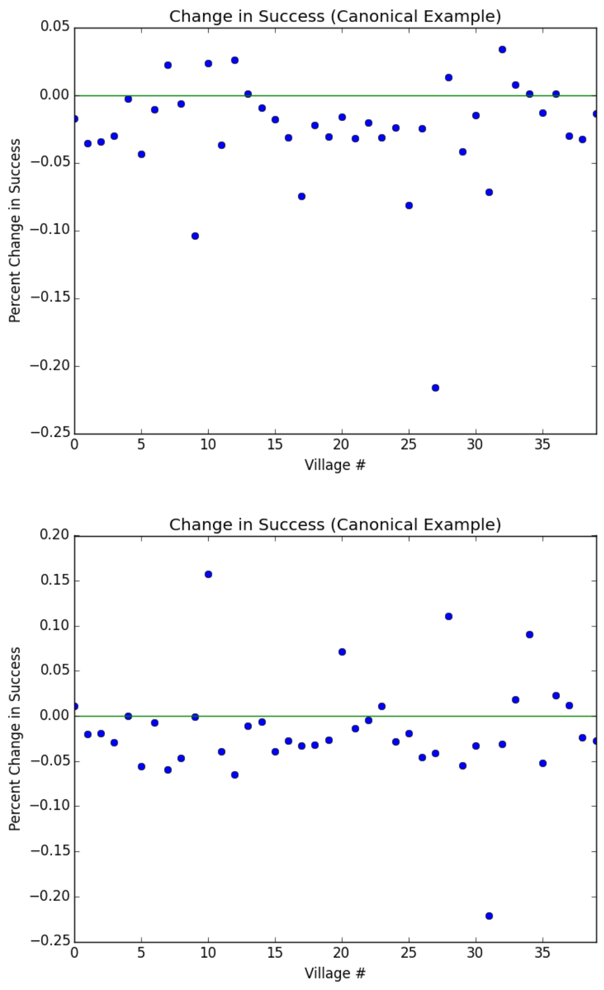
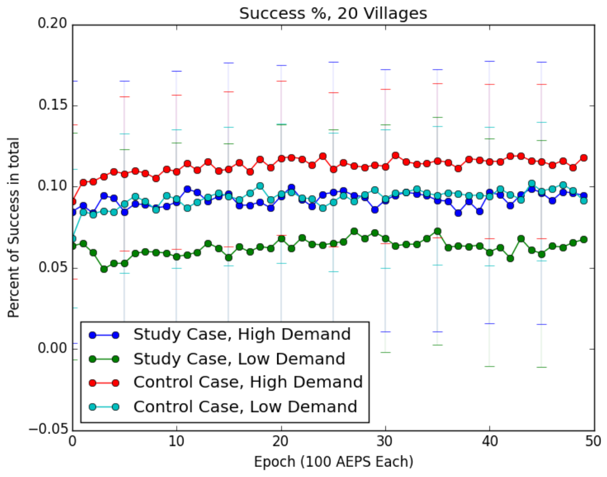


Figure 3: Two canonical examples, showing change in success after Trails were added to the simulation: **Top:** In most cases with only a few Villages, Trails result in a net loss for system success. **Bottom:** In a few, however, we can see improvement for select Villages — at cost to the others.

were a minority of runs that showed a few Village atoms performing better, at a cost for the rest. When this is averaged across many runs (20), we can see from Fig. 4 (Top) that the high-demand villages from the study case were only performing about as well as the low-demand villages from the control case.

Running it with 640 Villages yielded the graph in Fig. 4 (Bottom), where we see that the study case’s high-demand nodes have outpaced those from the control case, while the low-demand nodes from both are about equal. Other graphs, from the cases with 5, 10, 40, 80, 160, and 320 villages, have been excluded for brevity. We can see in the bottom graph that high-demand nodes experience about 16% and 14% success between the study and control cases, respec-



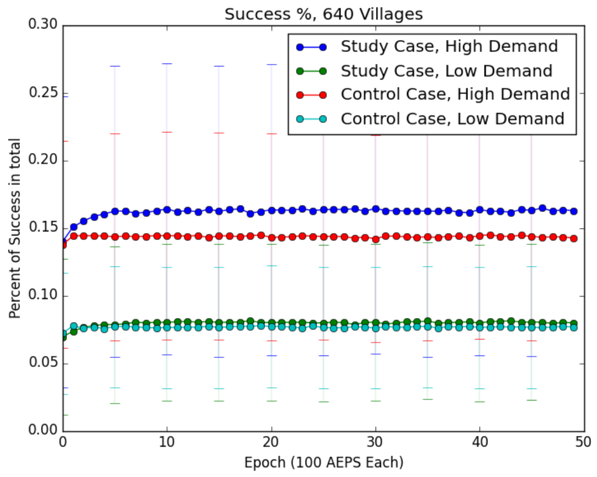


Figure 4: Comparison of percentage of successful attempts over time between 20 and 640 Villages

tively. In the top graph, the amount of chaos in the system is evident, but it also stabilizes, with success rates of 9% and 10%. The error bars represent standard deviation.

In Fig. 5, we can see the average system success ratio based on number of Village atoms in the simulation, with 20 runs for each data point. With a low number of villages, the control case high and low demand Village atoms both perform better; at the other end of the extreme, the high demand Villages from both the control and the study do much better than low demand villages, and the study case high-demand Villages do slightly better than those from the control. The standard deviation for these are once again shown as error bars.

Fig. 6 shows the percentage change in success between the control and the study case with Trails enabled; the strong upward trend clearly demonstrates that the simulation performs better with more Villages. With fewer than 80 Village

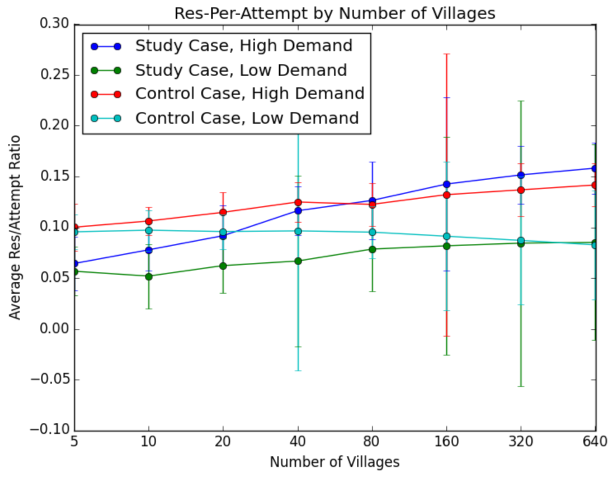


Figure 5: Success based on amount of Res found per attempt, with error bars for standard deviation

atoms, the results are negative for even high demand nodes, and it is not until the 640 Village case that it is beneficial for low demand Villages as well.

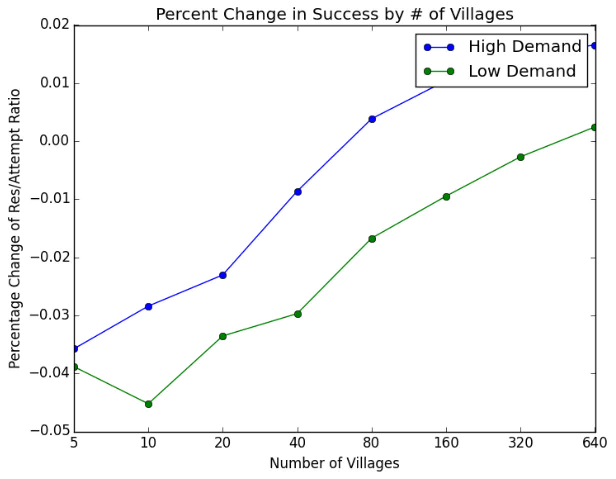


Figure 6: Percentage change in success by number of Village atoms in the simulation

Discussion

Based on the results for case study A, we can see that while a random walk is far from ideal for search in a typical computing environment, it is perhaps acceptable in a spatial environment, due to the considerations of scope. Scouts were able to find Villages regularly, and provided a link is established so that constant scouting can be kept to a minimum, this approach works decently. While random walk may be used frequently in artificial life and cellular automata, there are fewer if any cases of having randomly moving tendril structures that can form semi-permanent links. Due to the relative effectiveness of this Brownian walk at finding Village atoms, and in absence of an environment where traditional, efficient pathfinding is possible, we will consider this a reasonable solution to Objective 1. Note however that our parameters may be non ideal, especially when the average distance between Villages is taken into account, which we did not. If the chances of changing direction and stuttering are set too low, a Scout will often never change direction unless forced too; too high, and a Scout can often wander aimlessly for thousands of AEPS.

The results for case study B suggest that there may be either an implementation or conceptual problem in our optimization logic, resulting in the occasional formation of a hard angle. It is also possible (although unlikely) that this occurs due to some of the inherent stochastic qualities of the MFM itself. Additionally, paths are able to interfere with each other, with seemingly more regularity than should be permitted by a 10-bit random ID per scout. However, even in face of this adversity, optimal paths do typically form. The jittering appears to be vital to the formation of these paths, as the removal of the random choice of possible new position results in a high percentage of malformed paths. While the end results were not ideal, it is encouraging that the system is robust enough to handle these issues, and create a good path the majority of the time. Objective 2 we consider to be at least partially achieved, although some improvement might be desired for further work in this area.

A previous version of Trails relied on signals to "solidify", otherwise eventually timing out and dying like Scouts. This version resulted in Trails that would become "unhooked" at the source Village atom, which in turn resulted in many more Scouts being created. The net result was a MFM simulation with a huge density of Trails, none of which were able to do anything useful at all. The notion of allowing a Trail to exist at the source indefinitely unless the search fails from the Scout's point of view (and the failure is able to ripple back to the endpoint) represents a certain amount of investment on the part of a Village in its Scouts. This may not pay off for an individual Village, but it greatly contributes to the overall health of the system, benefiting all Villages in turn.

Case study C showed that the trails were able to pass data back and forth successfully, which admittedly is exactly what the system was built for. This could be modified to be made much more general - other approaches that were tested or concieved included having Message atoms that got passed around and gradually lost signal strength before dying, having a general-purpose message inside of Trails with a direction and strength for propagation, and having a two-way general-purpose message system. The Message atom idea resulted in massive crowding of the board, while the general-purpose messages proved to be either difficult to propagate messages with correctly, or to require so many bits for handling their logic that there was little space to store a message of any real length. This would be an interesting area to explore further, as communcations inside of spatial computing is a difficult problem, particularly with stringent space and scope budgets like that of the MFM.

As was previously mentioned, it is difficult to gather real data on the state of a large structure like a path in the MFM; the asynchronous execution and massively multithreaded nature make it inherently hard to collect data about specific atoms in any kind of predictable or controllable fashion. The constant state of churning in our system, as Trails are created, destroyed, and optimized, and new Scouts constantly disrupt the system by creating new Trails, adds to this difficulty. Therefore, we were unable to gather large amounts of data for statistical analysis. What was seen from basic visual inspection was considered sufficient to verify the basic functionality of the system, as well as to identify some areas in which improvement could be made.

With the results from case study D, it is clear that the Trails make things much worse for cases with fewer than 80 Villages. Based on visual inspection of the simulation during runtime, we believe this is caused by the spatial partitioning effect that Trails can have; as they often cut a long path across the simulation, impeding the movement of Res. Under some circumstances, this makes it difficult for a Res atom to randomly diffuse into the event window of a high-demand Village. This spatial partitioning effect is potentially useful, if it were more permanent, but the constant churning of the machine prevents this from ever truly solidifying.

The primary motivation for a dynamic resource allocation framework in something like the MFM would be to allow for more complex simulations where multiple spatial components are able to cooperate, and have more structure than currently found in the "muddy" simulations powered by the Dreg and Res framework(Ackley, 2013a). Despite this, our simulation actually performed much better with a large number of Villages, resulting in a crowded universe once again. Better persistence of Trails might help the low density simulations perform better, but would potentially come at a cost of robustness.

The metric used to judge the success of the routing is perhaps unfairly tipped to favor the high-demand Villages. Starvation of the low-demand Villages is common. The application of such a routing behavior might require a simulation where "feeder" nodes are creating lots of resources for a number of "hubs" which need the resources to do work. Whether such a structure would be practical is open to speculation, but we suspect it could be. Also worth considering is that while we fixed supply as a global constant and demand on a per-village scenario for simplicity, these could potentially be allowed to change.

We expected to see a general decline in the effectiveness of the simulation with an increase in the number of Villages. Our prediction was that the number of Scouts and Trails would clutter the simulation, preventing any real movement on the part of Res atoms, and thus starving high-demand nodes. If Villages were merely close enough to get their Res naturally, we would continue to not see any improvement after adding Trails. As we can see in Fig. 6, the effect of the Trails starts out as a detriment, but eventually becomes useful — the opposite of what was anticipated.

The general robustness of the simulation is somewhat lacking. Add a single Dreg atom, nuke a small area of the board, or disable a tile and the whole thing starts to fall apart. There is currently no means for Villages to reproduce, and a single Trail deleted from a path will cause the entire path to be destroyed. The first problem was solved in an earlier version of the project, where Villages would send out Colonists to create new Villages. This was removed as it contributed to clutter of the simulation, and also made any meaningful data analysis more tricky without greatly contributing to the success of any of the major objectives. Continual creation of new Villages helps, but in order to really make this system resilient, Trail atoms would need to try to repair the path if one Trail atom gets deleted in the middle. One possible way of doing this is to allow a Trail atom to try to move towards the last known position of the missing neighbor, to try to find the next Trail in the series, and make a new Trail to fill the gap. Another possibility would be to make a special variety of a Scout that would try to stich a path back together again, given an initial vector. This might work, since most paths do converge to an optimal semi-straight line. Such a Scout would need a short timer, however, to ensure it did not clutter up the world if it failed, and it would also probably need to be created with no chance of changing direction, and little to no chance of stuttering.

While our model has a few problems in terms of robustness and the optimality of the paths, we have shown that the model is capable of creating a connected, mostly-optimal network of spatially distributed nodes, and allowing these nodes to communicate with each other, and performing rudimentary need-based resource allocation, to the overall benefit of the system of nodes - if there are enough nodes. It seems plausible to us that this general approach or portions thereof might be useful in other areas of artificial life or distributed computation, by creating a communication or traffic network within a spatial environment.

Future Work

As mentioned above, the main areas of immediate improvement with this model would be to reduce sensitivity to peturbation by Dregs(Ackley, 2013a), probably by granting Trails the ability to self-replicate in order to patch up paths, as well as smoothing out the remaining issues with the path optimization. Beyond these, it might be useful to allow Trail and Village atoms to use an internal counter to represent Res rather than having Res atoms diffuse, as this would both free up space for other work and prevent wandering Res atoms from getting lost. Making the system robust against Dregs and Res could make it useful for allowing, say, a component of the MFM running Demon Horde Sort(Ackley, 2013a) to communicate with something that uses the sorted data. This could give rise to larger scales of structure, although it is difficult to say for sure that this would work.

Additionally, it would be interesting to see the behavior of the system with some kind of dynamically changing supply and demand, perhaps where the demand of a Village continues to increase as long as it is able to find Res, before coming to a plateau or eventually decreasing if a long period of deficit endures. This would not necessarily demonstrate the effectiveness of a network for resource distribution, but rather be more akin to an economical model where prosperous cities tend to grow. This could be particularly interesting and potentially useful for artificial life applications if these networks were able to grow new nodes by a modified version of the colonizing mechanic discussed earlier, where successful nodes are able to further colonize.

Acknowledgements

This work would not be possible without Professor Dave Ackley of UNM, Trent Small, and the wonderful students of UNM's CS 491: Robust Artificial Life.

References

Ackley, D. H. (2013a). Bespoke physics for living technology. *Artificial Life*, 19(3_4):347–364.

Ackley, D. H. (2013b). Beyond efficiency. *Commun. ACM*, 56(10):38–40.

Ackley, D. H., Cannon, D. C., and Williams, L. R. (2013). A movable architecture for robust spatial computing. *The Computer Journal*, 56(12):1450–1468.

Cui, Y., Che, H., Lagoa, C., and Zheng, Z. (2006). Autonomic interference avoidance with extended shortest path algorithm. In *Proceedings of the Third International Conference on Autonomic and Trusted Computing*, ATC'06, pages 57–66, Berlin, Heidelberg. Springer-Verlag.

Ke, W. and Mostafa, J. (2013). Studying the clustering paradox and scalability of search in highly distributed environments. *ACM Trans. Inf. Syst.*, 31(2):8:1–8:36.

Kelley, I. (2014). A distributed architecture for intra- and inter-cloud data management. In *Proceedings of the 5th ACM Workshop on Scientific Cloud Computing*, ScienceCloud '14, pages 53–60, New York, NY, USA. ACM.