AREA: an Automatic Runtime Evolutionary Adaptation mechanism for Creating Self-Adaptation Algorithms in Wireless Networks

Qingzhi Liu[1,2], *Stefan Dulman*[1], *Martijn Warnier*[2]
1. Embedded Software, EEMCS Faculty, Delft University of Technology, The Netherlands
2. System Engineering, TPM Faculty, Delft University of Technology, The Netherlands
{q.liu-1, s.o.dulman, m.e.warnier}@tudelft.nl

***Abstract*—The application requirements and the spatial environments of wireless networks continue to become more and more complex and changeable. Most existing algorithms for wireless sensor networks are designed with a specific type of environment in mind. While such algorithms work well in the environment they have been designed for, once the environment changes beyond the domain in which the design can adapt, the algorithms can hardly work properly. This paper proposes a novel design mechanism called the *automatic runtime evolutionary adaptation (AREA)* mechanism. It has been designed to automatically adapt, during runtime, to the variation of environments in wireless networks. This adaption is realized by self-creating and self-evolving algorithms: *AREA* allows the created algorithm not only to be adaptive but also to evolve to other adaptive abilities according to the variation of the application requirement and the spatial environment. The *AREA* mechanism is validated by applying it to a data aggregation example in wireless networks. This shows that the mechanism can adapt to the changing environments and outperform the other strategies.**

I. Introduction

Nowadays an increasing amount of research is focused on various wireless network applications, such as environment monitoring [1], traffic control [2] or navigation localization [3]. As the application complexity increases, more new research requirements are involved. However, most existing algorithms are designed with a specific type of (spatial) environment in mind, such as assumed bandwidth, node density, etc. Once the environment changes beyond the presumed domain, the algorithm will no longer be able to adapt: it will not function properly anymore. Therefore, it is necessary to design a mechanism that allows wireless networks to automatically self-create and self-evolve algorithms according to the changes in the (spatial) environment.

Based on this motivation, we propose a novel algorithm design mechanism called an *Automatic Runtime Evolutionary Adaptation (AREA)* mechanism. The *AREA* mechanism has three main properties, including automatic computing, runtime processing, and evolutionary adaptation.

The self-adaptation property has been widely recognized as an important performance metric for wireless networks. However, existing self-adaptive algorithms, based on design mechanisms such as swarm intelligence [4], stigmergy [5], and autopoiesis [6] only maintain the adaptation properties for specific environments. Once the deployment environment changes beyond the scope of the originally envisioned domain, the algorithm performance will decrease. The *AREA* mechanism allows the created algorithm not only to be adaptive but also to evolve to other adaptive abilities based on the variation of the application requirements and the spatial environment. In addition, *AREA* is totally distributed. Each agent only spatially coordinates with neighbors, uses runtime local information, and the whole processing flow is automatically executed in each agent during the runtime.

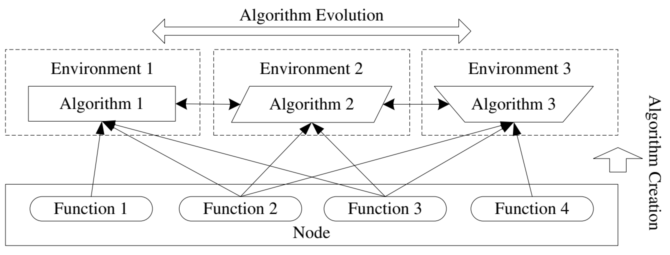


Fig. 1. *AREA* lets each agent (node) self-create algorithms that adapt to different application requirements. The created algorithms self-evolve to other function combinations, self-adapting to their (changing) environment.

The *AREA* mechanism assumes that each agent has some basic functions, such as routing, forwarding messages, etc. Each agent mutates local function combinations and learns from neighbors. Finally, the function combination that meets the environment requirements emerges and spreads throughout the network. If the environment or the agent function parameters change, and the selected function combination no longer meets the application requirements, agents evolve the algorithm and converge to new function combinations. In this way, agents always use the function combinations that suit the application requirements and the spatial environment. We also present a stabilization algorithm that reduces the churning phenomenon in different function combinations while maintains fast convergence of function selection.

We validate the *AREA* mechanism by applying it to the simulation of a data aggregation example. In the simulation example, each agent is supposed to have four basic functions: forwarding messages, routing messages, joining into clusters, and increasing the transmission range. The application requirement for every agent is to maintain the message arriving rate above a predefined threshold. By changing the agent density and transmission bandwidth parameters, agents in the network self-create and self-evolve various function combinations according to the spatial distribution. For the implementation of each component in *AREA*, we in detail illustrate how to select the parameters of fitness functions, etc.

based on the requirement. We use three other algorithms with fixed strategies for comparison. According to the test results, the *AREA* mechanism always maintains the best performance even when the environment and agent parameters change.

The remainder of the paper is organized as follows. Section II overviews the *AREA* mechanism and the application example. Section III illustrates *AREA*'s components in detail and the example implementation. We present the simulation results and evaluations in Section IV, related work follows in Section V and the paper ends with conclusions.

II. The *AREA* Mechanism Overview

In this section, we present the design framework of the *AREA* mechanism, the working components and the processing flow. And we demonstrate the general implementation of the data aggregation example based on *AREA*.

A. Mechanism Framework

It is supposed that each agent has some basic functions in the function set, such as forwarding messages, routing, etc. The agents are given an application requirement, such as "the message arriving rate should be larger than a predefined value". The agents self-create algorithms by selecting suitable combinations from predefined basic functions. When the environment changes, the agents self-evolve to select a different function combination as the new algorithm. In the system, each agent is independent and distributed, and only accesses the information of neighboring agents. The working process is automatic and during runtime. Figure 1 demonstrates an example of the *AREA* design framework. In the environment 1, the agent (node) converges to select the combination of function 1, 2 and 3 as the algorithm to fulfill the requirement. As the environment changes from environment 1 to 2, the originally created algorithm cannot meet the application requirement any longer. The agent evolves the algorithm and converge to select the combination of function 2 and 3. No matter how the environment changes, the agent always evolves to the function combination that will fulfill the predefined application requirements. After these requirements are fulfilled, the algorithm selected by the agents converges to a spatially stable state.

The *AREA* mechanism has four working components as shown in Figure 2(a). The first component is the definition of the *basic functions* of the agents in the function set. All predefined basic functions should work independently of each other. The second component is *function mutation*. Each basic function has a mutation probability that can change between being used and unused. So each agent can use different function combinations. The third component is *environment selection*. The agents in the network need to meet application requirements, such as maximizing the message arriving rate, minimizing the power consumption, etc. Each agent calculates a fitness credit for every function in the environment, and learns the function usage rule (using or unusing) of the agent with the largest function fitness credit in the neighboring

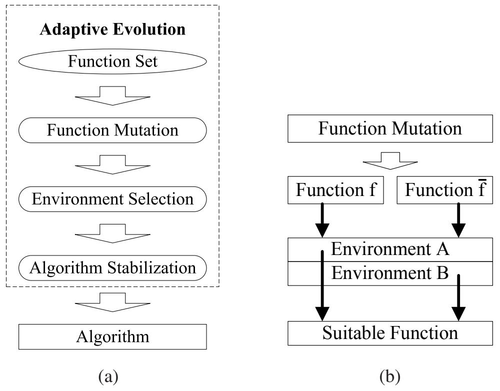


Fig. 2. (a).The working flow components of the *AREA* mechanism combine the most suitable functions into an algorithm. (b).The function mutation component creates different function usage rules (f: use the function; $\overline{f}$: unuse the function), and the rule that best suits the environment (A or B) is selected as the suitable function.

agents. This component allows the most suitable function combinations to be survived and diffused in the spatial network. To make the algorithm usable in practice, the algorithm needs to converge once the most suitable function combination is found, For this, the *algorithm stabilization* component is used. This component decreases the churn of selecting different function combinations in the network by each agent. The four components outlined above, make it possible for the agents in the network to select a suitable and stabilized function combination as the new algorithm. If the environment changes and the created algorithm can no longer fulfill the application requirements, the *AREA* mechanism will make the existing algorithm evolve to another function combination in order to meet the new application requirements.

B. Application Example

The *AREA* mechanism is validated by implementing it in an application example: data aggregation. Data aggregation is a basic building block for spatial network application. Most of the existing data aggregation algorithms are designed for specific deployment environments. In deployment scenarios where the environment may change sometimes, these algorithms can no longer work properly. We use the *AREA* mechanism to implement data aggregation in changing environments.

Suppose agents are randomly scattered in an area. Each agent can only communicate with neighbors. A sink agent is predefined to aggregate messages from other agents. If new agents come into the network, the agent density increases. If existing agents leave the network, the agent density decreases. We define the bandwidth as the number of the messages that can be forwarded by a agent at one tick. The agent density and bandwidth are changeable in the environment. We suppose that each agent knows the number of messages that are sent out and arrive at the destination. The data aggregation implementation based on *AREA* allows the agents in the network to automatically and during runtime find suitable function combination that meet its application requirements. The created function combination for data aggregation maintains a high message arriving rate and low power consumption values.

In the simulation, the environment is changed by adapting the agent density and bandwidth. The different simulation scenarios are outlined in Figure 3. At the start of the experiment the agents are randomly deployed. Under standard agent density and bandwidth, agents self-create function combinations that use forwarding and routing functions, as shown in Figure 3(a). Then we change the environment by increasing the agent density. Because the bandwidth is limited, some messages are now dropped. So the agents start to evolve the existing data aggregation approach, and spatially self-organize to clusters to increase the message arriving rate in the network as shown in Figure 3(b). In case the agent density decreases, and the network becomes disconnected the message arriving rates of the agents that are not connected to the destination agent become 0. When this happens, some of the agents evolve their algorithm and increase the transmission range to connect the network as shown in Figure 3(c). The increased transmission range of the agents will increase their message arriving rates.

III. The *AREA* Components

Based on the overall design presented in Section II, in the next subsections, we first define the content and structure of each component from *AREA*, and then present the detailed component implementation for the data aggregation example.

A. The Function Set

Each agent has some basic functions in a function set. These basic functions, such as joining into a cluster, are predefined, For each agent, all the possible function combinations form the search space of the available algorithms. For the data aggregation application, we use four basic functions in the function set for each agent: forwarding messages, routing the messages via the shortest path, joining to a cluster with other neighbors and increasing the transmission range.

Firstly, each agent has the basic function that it can forward messages to a neighboring agent. The bandwidth of forwarding messages is limited. It is supposed that the agent can only forward a limited number of messages during one time slot. If the number of messages to be forwarded is larger than the bandwidth, the agent drops the excess messages.

For the second function it is assumed that each agent can calculate the next hop on the shortest path based on the gradient [8]. If the agent knows the next hop on the shortest path, the agent forwards the messages to the next hop. Otherwise, the agent forwards the messages to a random neighboring agent. We assume that if the message is forwarded to a hop that is further away from the destination agent, then next hop agent drops the message.

Agents also have the function to join in a cluster. In a cluster, the cluster leader represents all the members to send out messages. Because the bandwidth is limited, we suppose that the forwarding priority of the message from a cluster is higher than the message from a single agent. And it is assumed that the forwarding priority of the message from a cluster with larger number of member agents is higher than the message from a cluster with smaller number of member agents. Agents forward messages from higher to lower priority.

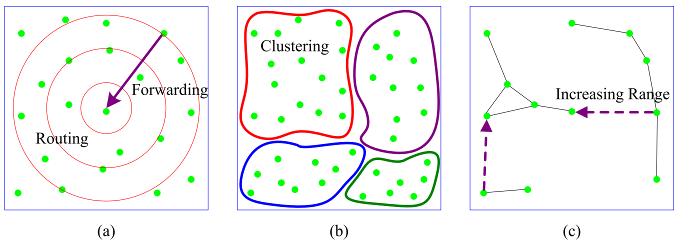


Fig. 3. The application example of *AREA* for data aggregation. (a). Agents evolve to use forwarding and routing functions. (b). Agents form clusters to send out messages. (c). Agents unconnected to the destination agent increase their transmission range.

Finally, agents have the basic function to increase their transmission range. Each agent has a default and an increased transmission range. If the agent cannot connect to the network with the destination agent using the default transmission range, then it can reconfigure (mutate) to use the increased transmission range.

B. Function Mutation

In this section, we present how the function combinations are created for each agent. We define the function combination series $\{f_1,f_2,...,f_i,...,f_n\}$. Where n is the total number of functions. f_i represents the basic function i explained in Section III-A. The value of f_i can be 0 or 1. $f_i = 0$ means that the function i is not used in the function combination, and $f_i = 1$ means that the function i is used. All the basic functions together form a function combination series. In the initial state, agents select a random value (0 or 1) for f_i $(i = 1, 2, ..., n)$ of each basic function. In the function mutation component, *AREA* makes each agent change the function value f_i by setting $f_i = \overline{f_i}$ with a mutation probability.

In the application example, agent x has a $Switch_{xi}$ value 0 or 1 for each function i representing unused or used. We predefine a mutation probability P_m for all the basic functions. If a random value is smaller than P_m, then $Switch_{xi}$ does not change. Otherwise, $Switch_{xi}$ is changed to $\overline{Switch_{xi}}$.

C. Environment Learning

In this section, we demonstrate how agents learn from each other, and make the most suitable function combination spread throughout the network. We define the fitness credit of the function for each agent by $C_i = w_{i1}c_{i1} + w_{i2}c_{i2} + \cdots + w_{ik}c_{ik} + \cdots + w_{im}c_{im}$. Where i is the function number as defined in Section III-B. k $(k = 1, 2, ..., m)$ is the number of the evaluation criteria for the function, and m is the total number of evaluation criteria. For example, the performance of the network can be determined by the arriving rate, power consumption, etc. c_{ik} is the fitness credit of the function i evaluated by the criteria k. w_{ik} is the weight of the fitness credit with evaluation criteria k. In the learning component, each agent records the fitness credit of each function. At the same time, each agent detects the fitness values of the functions from neighboring agents and learns the function usage strategy (used or unused) from the agent with the largest fitness credit. Finally, the function combination that has the largest fitness credit in the network survives and spreads. The mutation and learning process is shown in Figure 2(b).

For the application example, we define the fitness credit for every function as follows. The fitness credit of forwarding the messages is defined as $AR{\cdot}W_{AR}{+}FR{\cdot}W_{FR}{+}FC{\cdot}W_{FC}$. The AR value equals the percentage of the number of messages arriving at their destination. If the agent forwards messages of other agents, then $FR = 1$; otherwise, $FR = 0$. FC is the power consumption credit for forwarding messages. If the agent forwards messages, FC is 0; otherwise, FC is 1. W_{AR}, W_{FR} and W_{FC} are the weight values of AR, FR and FC. The value domain of AR is $[0, 1]$. FR and FC can be 0 or 1. For different application requirements, AR, FR and FC can be set to different values. In the simulation, each agent sends out 10 messages for each communication round, so the precision accuracy is 0.1. The message arriving rate is assumed to be the most important evaluation criteria. Therefore, the weight value of the arriving rate is set as $W_{AR} = 1$. $W_{FR} = 0.02$ and $W_{FC} = 0.01$. Their sum is smaller than the arriving rate precision accuracy 0.1, so their weight values do not affect the distinction among different message arriving rate values. In the network, we suppose that forwarding messages is more important than saving energy. We set $W_{FR} = 0.02$, which is larger than $W_{FC} = 0.01$. The fitness credit of routing messages is defined as $AR{\cdot}W_{AR}{+}FR{\cdot}W_{FR}{+}RC{\cdot}W_{RC}$. RC is the power consumption credit for routing. If the agent calculates the next hop on the shortest path, $RC = 0$; otherwise $RC = 1$. The weight value $W_{RC} = 0.01$. The fitness credit of constructing a cluster is defined as $AR \cdot W_{AR} + CC \cdot W_{CC}$. CC is the power consumption credit for constructing and maintaining a cluster. If the agent joins in a cluster, $CC = 0$; otherwise $CC = 1$. The weight value $W_{CC} = 0.01$. The fitness credit of adjusting the transmission range is defined as $AR \cdot W_{AR} + RN \cdot RU \cdot W_{RU} + RC \cdot W_{RC}$. Where RN is the parameter that shows whether the agent has a connection route to the destination agent. If the agent cannot find a connection route to the destination agent, then RN is set to 1, which means that the agent has increased its transmission range to the maximum allowed value. RU is the usage value of increased transmission range. If the agent increases the transmission range, and the increased range is used to forward messages of other agents, then $RU = 1$; otherwise $RU = 0$. The weight value W_{RU} is set to 0.02. RC is the power consumption credit for using different transmission ranges. If the agent uses the default transmission range, $RC = 1$; if it uses the increased transmission range, $RC = 0$. The weight value $W_{RC} = 0.01$.

D. Stabilization

To make the selected function combination feasible, it must be stable. In this section, we introduce an algorithm that can stabilize the system. The mutation and learning probability are the main factors that affect churning in the selection and spread of the function combination among agents. First, we define a fitness credit threshold value T_i. This value equals the minimum acceptable fitness credit value by using or unusing the function i. Then we define a stabilization rate D, which equals the rate between the normal mutation or learning probability and the minimum acceptable mutation or learning probability. If the fitness credit of the function i is smaller than T_i, then the agent selects learning probability P_L and mutation probability P_M. Otherwise, the agent selects learning probability $\frac{P_L}{D}$ and mutation probability $\frac{P_M}{D}$. So each agent uses low mutation and learning probabilities in the state with acceptable fitness credit to decrease churning of different function combinations. The detail processing flow is shown in Algorithm 1. Each agent x is given a $Switch_{xi}$ value for each function i. Where the switch values change between 0 and 1 representing unusing and using the function, R is the random probability from 0 to 1 and Nb is the neighbor agents.

Algorithm 1 : Stabilization for Mutation and Learning

```
Mutation:
for all Agent $x$ do
    if ($C_i \le T_i \& R < P_M$)||($C_i > T_i \& R <$ ($P_M/D$)) then
        $Switch_{xi} = 1 - Switch_{xi}$
    end if
end for

Learning:
for all Agent $x$ do
    $maxC = max([C_i]\ of\ Nb)$
    if $maxC > C_i$ then
        if ($C_i \le T_i \& R < P_L$)||($C_i > T_i \& R <$ ($P_L/D$)) then
            $Switch_{xi} = [Switch_{xi}]\ of\ Nb\ with\ [maxC]$
        end if
    end if
end for
```

To encourage the use of the forwarding and routing functions, in the data aggregation application, the threshold values of forwarding and routing are set to 1.01. So only agents with fitness credit 1.02 and higher can use $\frac{P_L}{D}$ and $\frac{P_M}{D}$ to learn and mutate. Whether to actually use the clustering and increasing the transmission range functions is based on the environment condition, such as agent density, bandwidth, etc., so the threshold value of clustering and ranging are set to 0.99 and 1.0 respectively.

IV. Test and Evaluation

We use NetLogo [?] for the simulation experiment. The deployment area is a 200×200 square region. Agents are static and randomly scattered across the area. Each agent sends out one message per one tick. After sending 10 messages, each agent waits 10 ticks for calculating the message arriving rate. We call the 20 ticks as one communication round. The default transmission range of each agent is 40 units. All the agents send messages to one specified destination agent. We run the experiment 10 times for each testing point.

A. Adaptation to Environments

In the experiments, the environment changes by varying the agent density and the bandwidth. The agent density ranges from 5 to 40, with 5 offset. The bandwidth is 5×2 units. m is from 0 to 7, with 1 offset. Three evaluation parameters are

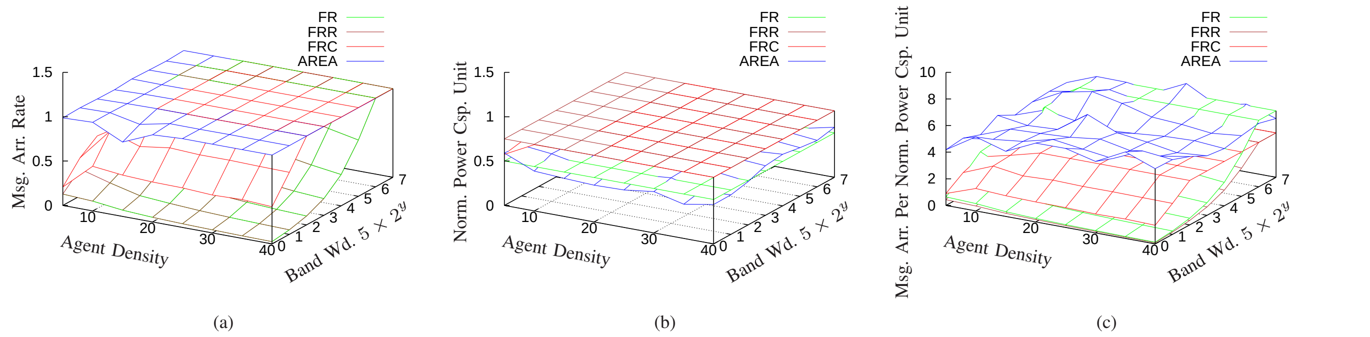


Fig. 4. (a).Average message arriving rate for various agent densities and bandwidths. (b).Normalized average power consumption unit for various agent densities and bandwidths. (c).Average message arriving count per normalized power consumption unit for various agent densities and bandwidths.

used: average message arriving rate, average power consumption unit and average arriving rate per power consumption unit. Three other data aggregation methods are used for comparison. The first method, named as *FR*, makes all agents forward and route the messages. The second method, named as *FRC*, makes all agents construct a fixed number of clusters. All agents can forward and route the messages, but only the cluster leader represents the cluster to send out messages. The cluster number is set to 20. The third method, named as *FRR*, makes all agents use the increased transmission range to forward and route the messages. The increased transmission range is 80 units. To evaluate the power consumption performance, we suppose that the running of each basic function consumes 1 unit power per round.

Figure 4(a) shows the average message arriving rate for various agent densities and bandwidths. The *AREA* mechanism has the best arriving rate in the low density area 5. This is because the network is generally unconnected with agent density 5. If the agents that are not connected to the destination agent do not increase their transmission range, they cannot send messages to the destination. Furthermore, the *AREA* mechanism makes some agents increase their transmission range to link the connected network. In the low bandwidth area from 0 to 2, the *AREA* mechanism constructs clusters to meet the bandwidth restrictions. The method *FRC* uses clusters to increase the message arriving rate. When the bandwidth is equal or larger than 20, which is equal to the fixed cluster number of *FRC*, the *FRC* method has the best arriving rate. But because it is construct with a fixed number of clusters, the method is not adaptive to the variation of the environment. If the environment has unconnected subnetwork or very low bandwidth, agents using the *FRC* method will drop messages. Agents that use the *FRR* method also use the increased transmission range all the time. In the connected network with low bandwidth, increasing the transmission range cannot always increase the message arriving rate. Although *AREA* does not have the best arriving rate, the difference to the best arriving rate is very small. This is because agents need to keep a mutation rate to adapt to the new environments in *AREA*. When the bandwidth is larger than 6, the *FR* and *FRR* methods can also have near optimal message arriving rate. This is because that the bandwidth is larger than the total number of agents in the network. So no matter how the network topology is constructed, all the messages will finally arrive at their destination.

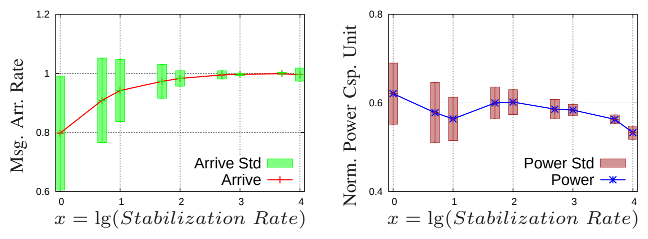
Figure 4(b) illustrates the power consumption for various agent density and bandwidth. Because there are four types of basic functions, the average power consumption unit is normalized to 4. It can be found that the *FRC* and *FRR* methods consume the same amount of energy in all the testing points. This can be explained because all agents always use clustering in the *FRC* method and increase their transmission range when using the *FRR* method. This is because it only uses two basic functions: forwarding and routing. The power consumption of *AREA* is larger than *FR*, because *AREA* sometimes increases its transmission range and constructs a cluster to increase the message arriving rate. But *AREA* makes all the agents in the network automatically adapt to the different environments.

In order to evaluate the efficiency of the *AREA* algorithm, we calculate the rate between the message arriving count and the power consumption unit for various agent densities and bandwidth values, as shown in Figure 4(c). It can be seen that the *AREA* algorithm is the most efficient method for almost all the testing points. Only in the very high bandwidth area, the *FR* method is slightly more efficient. This is again because the total bandwidth is larger than the total number of agents. Since the *AREA* algorithm always will mutate to other function combinations, its efficiency is slightly less at this point.

B. *Stabilization Efficiency*

Agents in environments with a different spatial distribution could churn in different function combinations. Stabilization of the function selection is an important evaluation parameter. We test the stabilization algorithm presented in Section III-D.

The testing results are shown in Figure 5. We initialize the environment with an agent density of 20 and bandwidth of 40. The y coordinates of Figure 5(a) and Figure 5(b) are the message arriving rate and the normalized power consumption unit respectively. The x coordinates in both figures are calculated by $x = \lg D$, in which D is the stabilization rate with value 1, 5, 10, 50, 100, 500, 1000, 5000, 10000. The other testing



(a) (b)

Fig. 5. (a).The average message arriving rate and standard deviation with various stabilization rates D. (b).The normalized power consumption unit and standard deviation with various stabilization rates D.

parameters are the same as the previous experiments. According to the results, as the stabilization rate D increases from 1 to 10000, the standard deviations of the message arriving rate and power consumption value decreases significantly. When the stabilization rate becomes 10000, which is 4 ($= \lg 10000$) in the figures, the standard deviations of the arriving rate and power consumption are 0.022 and 0.015. As the stabilization rate increases, the message arriving rate increases from 0.87 to 1. This is because the low stabilization rate D make the system unstable, and further decreases the message arriving rate. Therefore the stabilization Algorithm 1 can effectively stabilize the function mutation and learning.

V. Related Work

Traditional spatial computing algorithms [9] [10] primarily focus on function implementation in a fixed environment. The spatial and temporal distribution of the agents in the network significantly affect the relations among agents. In this paper, we make advantage of the coordination between agent interaction in the spatial network to promote the evolution of algorithms adapting to various changing environments.

Evolutionary dynamics [11] researches the power of advancing the system to evolves from one state to another on a global population level. Evolution dynamics theory forms the theoretical basis on which the *AREA* mechanism is founded. Evolutionary computing is widely researched for producing optimized systems [12]. Traditional evolutionary computation [13] selects members via centralized methods, which is not very efficient in distributed environments. Nakano and Suda [14] and Lee et al. [15] improve the adaptation mechanisms using evolutionary computing. They propose design structures that build adaptive network services using bio-inspired distributed agents. By evolutionary adaptation, agents can evolve and adapt their behavior to the changing environments. And Champrasert et al. [16] present a structure to self-optimize and self-stabilize cloud applications. It extends the biological evolutionary adaptation from agents to related platforms. But the above papers do not explain the influence of the parameters to the performance of the system. Moreover, these works focus on the construction of services using interacting agents, and the simulation is on a grid network, which does not map naturally (nor easily) to wireless networks. Mirko et al. [**?**] extends the tuple spaces by chemical-inspired model to coordinate spatially pervasive services. Although the mechanism can effectively make services diffuse and interact in a spatial network, it can not coordinate multiple different services together to cope with various problems.

VI. Conclusions

We propose a new mechanism: *automatic runtime evolutionary adaptation (AREA)* for spatial algorithm design. It can effectively create and evolve the algorithms in wireless networks to meet application requirements and adapt to the changing of the environment. The working process of *AREA* is automatically updated in every agent during runtime and the created algorithm works stably in the network. The mechanism is validated by the simulation experiment involving data aggregation. For the future work, we plan to use the *AREA* mechanism for the creation of other algorithms, such as self-created gradient, synchronization, etc. Further more, the *AREA* mechanism has the potential to be used in other domains, such as Internet services, swarm robotics control, etc., which will be further researched.

References

[1] G. Werner-Allen, J. Johnson, M. Ruiz, J. Lees, and M. Welsh, "Monitoring volcanic eruptions with a wireless sensor network," in *Wireless Sensor Networks, 2005. Proceeedings of the Second European Workshop on*. IEEE, 2005, pp. 108–120.

[2] U. Lee, B. Zhou, M. Gerla, E. Magistretti, P. Bellavista, and A. Corradi, "Mobeyes: smart mobs for urban monitoring with a vehicular sensor network," *Wireless Communications, IEEE*, vol. 13, no. 5, pp. 52–57, 2006.

[3] G. Mao, B. Fidan, and B. Anderson, "Wireless sensor network localization techniques," *Computer Networks*, vol. 51, no. 10, pp. 2529–2553, 2007.

[4] J. Kennedy, "Swarm intelligence," *Handbook of nature-inspired and innovative computing*, pp. 187–219, 2006.

[5] M. Dorigo, E. Bonabeau, and G. Theraulaz, "Ant algorithms and stigmergy," *Future Generation Computer Systems*, vol. 16, no. 8, pp. 851–871, 2000.

[6] N. Nanas and A. De Roeck, "Autopoiesis, the immune system, and adaptive information filtering," *Natural Computing*, vol. 8, no. 2, pp. 387–427, 2009.

[7] W. Jung, K. Lim, Y. Ko, and S. Park, "Efficient clustering-based data aggregation techniques for wireless sensor networks," *Wireless Networks*, vol. 17, no. 5, pp. 1387–1400, 2011.

[8] Q. Liu, A. Pruteanu, and S. Dulman, "Gde: a distributed gradient-based algorithm for distance estimation in large-scale networks," in *MSWiM 2011, ACM*, 2011, pp. 151–158.

[9] J. Beal and R. Schantz, "A spatial computing approach to distributed algorithms," in *45th Asilomar Conference on Signals, Systems, and Computers*, 2010.

[10] M. Duckham, "Decentralized spatial algorithm design," *Spatial Computing 2012 colocated with AAMAS*, p. 13.

[11] M. Nowak, *Evolutionary Dynamics: exploring the equations of life*. Belknap Press, 2006.

[12] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments : A survey," *Evolutionary Computation, IEEE Transactions on*, vol. 9, no. 3, pp. 303–317, 2005.

[13] M. Melanie, "An introduction to genetic algorithms," *Cambridge, Massachusetts London, England, Fifth printing*, 1999.

[14] T. Nakano and T. Suda, "Self-organizing network services with evolutionary adaptation," *Neural Networks, IEEE Transactions on*, vol. 16, no. 5, pp. 1269–1278, 2005.

[15] C. Lee, J. Suzuki, and A. Vasilakos, "An evolutionary game theoretic framework for adaptive, cooperative and stable network applications," *Bio-Inspired Models of Network, Information, and Computing Systems*, pp. 189–204, 2012.

[16] P. Champrasert, J. Suzuki, and C. Lee, “Exploring self-optimization and self-stabilization properties in bio-inspired autonomic cloud applications,” *Concurrency and Computation: Practice and Experience*, 2012.