Bridging Adaptive Estimation and Control with Modern Machine Learning: A Quorum Sensing Inspired Algorithm for Dynamic Clustering

by

Feng Tan

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2012

Author .....
Department of Mechanical Engineering
August 20, 2012

Certified by .....
Jean-Jacques Slotine
Professor
Thesis Supervisor

Accepted by .....
David E. Hardt
Chairman, Department Committee on Graduate Students

Bridging Adaptive Estimation and Control with Modern Machine Learning: A Quorum Sensing Inspired Algorithm for Dynamic Clustering

by

Feng Tan

Submitted to the Department of Mechanical Engineering
on August 30, 2012, in partial fulfillment of the
requirements for the degree of
Master of Science in Mechanical Engineering

Abstract

Quorum sensing is a decentralized biological process, by which a community of bacterial cells with no global awareness can coordinate their functional behaviors based only on local decision and cell-medium interaction. This thesis draws inspiration from quorum sensing to study the data clustering problem, in both the time-invariant and the time-varying cases.

Borrowing ideas from both adaptive estimation and control, and modern machine learning, we propose an algorithm to estimate an "influence radius" for each cell that represents a single data, which is similar to a kernel tuning process in classical machine learning. Then we utilize the knowledge of local connectivity and neighborhood to cluster data into multiple colonies simultaneously. The entire process consists of two steps: first, the algorithm spots sparsely distributed "core cells" and determines for each cell its influence radius; then, associated "influence molecules" are secreted from the core cells and diffuse into the whole environment. The density distribution in the environment eventually determines the colony associated with each cell. We integrate the two steps into a dynamic process, which gives the algorithm flexibility for problems with time-varying data, such as dynamic grouping of swarms of robots.

Finally, we demonstrate the algorithm on several applications, including benchmarks dataset testing, alleles information matching, and dynamic system grouping and identication. We hope our algorithm can shed light on the idea that biological inspiration can help design computational algorithms, as it provides a natural bond bridging adaptive estimation and control with modern machine learning.

Thesis Supervisor: Jean-Jacques Slotine
Title: Professor

Acknowledgments

I would like to thank my advisor, Professor Jean-Jacques Slotine with my sincerest gratitude. During the past two years, I have learned not only knowledge, but also principles of doing research under his guidance. His attitude to a valuable research topic, "Simple, and conceptually new", although sometimes "frustrating", is inspiring me all the time. His broad vision and detailed advice help me all the way towards novel and interesting explorations. He is a treasure to any student and it has been a honor to work with him.

I would also like to thank my parents for unwavering support and encouragement. All the time from childhood, my curiosity and creativity were so encouraged and my interests were developed with their support. I feel peaceful and warm with them on my side as always.

This research was sponsored in part by a grant from the Boeing corporation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis is primarily concerned with developing an algorithm that can bridge adaptive estimation and control with modern machine learning techniques. For the specific problem, we develop an algorithm inspired by nature, dynamically grouping and coordinating swarms of dynamic systems. One motivation of this topic is that, we will inevitably encounter control problems for groups or swarms of dynamic systems, such as manipulators, robots or basic oscilators, as researches in robotics advance. Consequently, incorporating current machine learning techniques into the control theory of groups of dynamic systems would enhance the performance and achieve better results by forming "swarm intelligence". This concept of swarm intelligence would be more intriguing if the computation can be decentralized and decisions can be made locally since such mechanism would be more flexible, consistent and also robust. With no central processor, computational load can be distrbuted to local computing units, which is both efficient and reconfigurable.

When talking about self-organization and group behavior, in control theory we have the ideas about synchronization and contraction analysis, while in the machine learning fields, we can track the progress in a lot of researches in both supervised or unsupervised learning. Currently researches are experimenting with various methods on classification and clustering problems, such as Support Vector Machine[1], K-Means clustering[2], Spectral clustering[3][4], etc. However, rarely have these algorithms been developed to fit into the problems of controling real-time dynamic systems, al-

though successful applications in image processing, video and audio recognition have thrived in the past decades.

The main challenge that we will consider in this thesis is how to modify and fomulate the current machine learning algorithms, so that they can fit in and improve the control performance of dynamic systems. The main contribution of this thesis is that we develop a clustering algorithm inspired by a natural phenomenon-quorum sensing, that is able to not only perform clustering on benchmark datasets as well as or even better than some existing algorithms, but also integrate dynamic systems and control strategies more easily. With further extensions made possible through this integration, control theory would be more intelligent and flexible. With the inspiration from nature, we would like to discuss more about the basic concepts like what is neighborhood, and how to determine the relative distance between data points. We hope these ideas provide machine learning with deeper understandings and also find the unity between control, machine learning and nature.

1.1 Road Map

The format in Chapter 1 is as follows: we will first introduce the motivations of this thesis, with backgrounds, potential applications and detailed goal of this thesis. Then we will briefly introduce the clustering alogorithms by their characteristics and limitations. Finally, we will illustrate our inspiration from nature, with detailed description of quorum sensing and the key factors of this phenomenon that we can extract to utilize for developing the algorithm.

Beyond the current chapter, we will describe detail analysis of our algorithm in the second chapter. We will provide mathematical analysis on the algorithm from both views from machine learning and dynamic system control stability. Then clustering merging and splitting policies will be introduced along with comparison with other clustering algorithms. In chapter three, we will put our clustering algorithm into actual experiments, including both synthetic and real benchmarks experiments, experiments on alleles classification, and finally on real time dynamic system grouping.

In the last chapter, we will discuss about the results shown in the previous chapters by comparing the strengths and weaknesses of our proposed algorithm, and provide our vision on future works and extensions.

1.2 Motivations

Control theory and machine learning share the same goal, which is to optimize certain functions to either reach a minimum/maxmimum or track a certain trajectory. It is the implementing methods that differs the two fields. In control theory, since the controled objects have the requirements of real-time optimization and stability, the designed control strategy starts from derivative equations, so that after a period of transient time, satisfying control performance can be achieved. On the other hand, machine learning faces mostly with static data or time varying data where a static feature vector can be extracted from, so optimization in machine learning is more direct to the goal by using advanced mathematical tools. We think if we can find a bridge connecting the two fields more closely, we can then utilize the knowledge learned from past to gain better control performance, and also we can bring in the concepts like synchronization, contraction analysis, and stability into machine learning theories.

1.2.1 Backgrounds

Research in adaptive control started in early 1950's as a technology for automatically adjusting the controller parameters for the design of autopilots for aircrafts[5]. Between 1960 and 1970 many fundamental areas in control theory such as state space and stability theory were developed. These progresses along with some other techniques including: dual control theory, identification and parameter estimation, helped scientists develop increased understanding of adaptive control theory. In 1970s and 1980s, the convergence proofs for adaptive control were developed. It was at that time a coherent theory of adaptive control was formed, using various tools from nonlinear control theory. Also in the early 1980s, personal computers with 16-bit microproces-

sors, high-level programming languages and process interface came on the market. A lot of applications including robotics, chemical process, power systems, aircraft control, etc. benefitting from the combination of adaptive control theory and computation power, emerged and changed the world.

Nowadays, with the development of computer science, computation power has been increasing rapidly to make high-speed computation more accessible and inexpensive. Available computation speed is about 2^{20} times what it was in the 1980's when key adaptive control techniques were developed. Machine learning, whose major advances have been made since the early 2000s, benefitted from the exploding computation power, and mushroomed with promising theory and applications. Consequently, we start looking into this possibility of bridging the gap between machine learning and control, so that the fruit of computation improvements can be utilized and transferred into control theory for better performance.

1.2.2 Potential Applications

Many potential applications would emerge if we can bridge control and machine learning smoothly, since such a combination will give traditional control "memory", intelligence and much more flexibility. By bringing in manifold learning into control, which is meant to find a low dimentional basis for describing high dimentional data, we can possibly develop an algorithm that can shrink the state space into a much smaller subset. High dimensional adaptive controllers, especially networks controllers like[6] would benefit hugely from this. Also we can use virtual systems along with dynamic clustering or classification analysis to improve performance of multi-model switching controllers, such as [7][8][9][10]. This kind of classification technique would also be helpful for diagnosis of control status, such as anonaly detections in [11]. Our effort presented in developing a clustering algorithm that can be incorporated with dynamic systems is a firm step right on building this bridge.

1.2.3 Goal of this thesis

The goal of this thesis is to develop a dynamic clustering algorithm suitable to implement on dynamic systems grouping. We propose this algorithm with better clustering performances on static benchmarks datasets compared to traditional clustering methods. In the algorithm, we would like to realized the optimization process not by direct mathematical analysis or designed iterative steps, yet by derivative equations convergence, since such algorithm would be easily combined with contraction analysis or control stability analysis for coordinating group behavior. We want to develop the algorithm not so "artificial", yet in a more natural view.

1.3 Clustering Algorithms

When thinking about the concept of intelligence, two basic parts contributing to intelligence are learning and recognition. Robots, who can calculate millions of times faster that human, are good at conducting repetitive tasks. Yet, they can hardly function when faced with a task that has not been predefined. Learning and recognition requires the ability to process the incoming information, conceptualize it and finally come to a conclusion that can describe or represent the characteristics of the perceived object or some abstract concepts. The concluding process requires to define the similarities and dissimilarities between classes, so that we can distinguish one from another. In our mind we put things that are similar to each other into a same groups, and when encountering new things with similar charateristics, we can recognize them and classify them into these groups. Furthermore, we are able to rely on the previously learned knowledge to make future predictions and guide future behaviors. Thus, it is reasonable to say, this clustering process, is a key part of the learning, and further a base of intelligence.

Cluster analysis is to separate a set of unlabeled objects into clusters, so that the objects in the same cluster are more similar, and objects belonging to different clusters are less similar, or dissimilar. Here is a mathematical description of hard clustering problem[12]:

Given a set of input data $X = \vec{x}_1, \vec{x}_2, \vec{x}_3, ...\vec{x}_N$,

Partition the data into K groups: $C_1, C_2, C_3, ...C_K, (K \leq N)$, such that

1) $C_i \neq \emptyset, i = 1, 2, ..., K$;

2) $\bigcup_{i=1}^{K} C_i = X$;

3) $C_i \bigcap C_j = \emptyset, i, j = 1, 2, ..., K, i \neq j$

Cluster analysis is widely applied in many fields, such as machine learning, pattern recognition, bioinformatics, and image processing. Currently, there are various clustering algorithms mainly in the categories of hierarchical clustering, centroid-based clustering, distribution-based clustering, density-based clustering, spectral clustering, etc. We will introduce some of these clustering methods in the following part.

1.3.1 Previous Work

Hierarchical clustering

There are two types of hierarchical clustering, one is a bottom up approach, also know as "Agglomerative", which starts from the state that every single data forms its own cluster and merges the small clusters as the hierarchy moves up; the other is a top down approach, also know as "Divisive", which starts from only one whole cluster and splits recursively as the hierarchy moves down. The hierarchical clustering algorithms intends to connect "objects" to "clusters" based on their distance. A general agglomerative clustering algorithm can be summarized as below:

1. Start with N singleton clusters.

   Calculate the proximity matrix for the N clusters.

2. Search the minimal distance

   $D(C_i, C_j) = min_{1 \leq m, l \leq N, l \neq m} D(C_m, C_l)$

   where D is a distance function adopted specified on certain dataset.

Then combine cluster C_i, C_j to form a new cluster.

3. Update the proximity matrix.

4. Repeat step 2 and 3.

However, the hierarchical clustering algorithms are most likely sensitive to outliers and noise. And once a misclassification happens, the algorithm is not capable of correcting the mistake in the future. Also, the computational complexity for most of hierarchical clustering algorithms is at least $O(N^2)$. Typical examples of hierarchical algorithms include CURE[13], ROCK[14], Chameleon[15], and BIRCH[16].

Centroid-based clustering

The Centroid-based clustering algorithms try to find a centroid vector to represent a cluster, although this centroid may not be a member of the dataset. The rule to find this centroid is to optimize a certain cost function, while on the other hand, the belongings of the data are updated as the centroid is repetitively updated. K-means[2] clustering is the most typical and popular algorithm in this category. The algorithm partition n data into k clusters in which each data belongs to the cluster with the nearest mean. Although the computation is NP-hard, there are heuristic algorithms that can assure convergence to a local optimum.

The mathematical model of K-means clustering is:

Given a set of input data $X = \vec{x}_1, \vec{x}_2, \vec{x}_3, ...\vec{x}_N$, Partition them into k sets $C_1, ...C_K, (K \leq N)$, so as to minimize the within cluster sum of squares:

$$argmin_C \sum_{i=1}^{k} \sum_{x_j \in C_i} \|x_j - \mu_i\|^2$$

where μ_i is the mean of points in C_i

Generally, the algorithm proceeds by alternating between two steps:

Assignment step: $C_i^{(t)} = \{x_p : \|x_p - \mu_i^{(t)}\| \leq \|x_p - \mu_j^{(t)}\| \forall 1 \leq j \leq k\}$

Update step: calculate the new means $m_i^{(t+1)} = \frac{1}{C_i^{(t)}} \sum_{x_j \in C_i^{(t)}} x_j$

However, the K-means algorithm has several disadvantages. It is sensitive to initial states, outliers and noise. Also, it may be trapped in a local optimum where the clustering result is not acceptable. Moreover, it is most effective working on hyper-spherical datasets, when faced with some dataset with concave structure, the centroid information may be not precise or even misleading.

Distribution-based clustering

Distribution based clustering methods are closely related to statistics. They tend to define clusters using some distribution candidates and gradually tune the parameters of the functions to fit the data better. They are especially suitable for artificial datasets, since they are originally generated from a predefined distribution. The most notable algorithm in this category is expectation-maximization algorithm. It assumes that the dataset is following the a mixture of Gaussian distributions. So the goal of the algorithm is to find the mutual match of the Gaussian models and the actual dataset distribution.

Given a statistical model consisting of known data $X = \vec{x}_1, \vec{x}_2, \vec{x}_3, ...\vec{x}_N,$, latent data Z, and unknown parameters $\vec{\theta}$, along with the likelihood function:

$$L(\vec{\theta}; X, Z) = p(X, Z|\vec{\theta})$$

and the marginal likelihood

$$L(\vec{\theta}; X) = p(X|\vec{\theta}) = \sum_Z p(X, Z|\vec{\theta})$$

The expectation-maximization algorithm then iteratively applies the following two steps:

Expectation step: calculate the expected value of log likelihood function $Q(\vec{\theta}|\vec{\theta}^{(t)}) = E_{Z|X,\vec{\theta}^{(t)}}[logL(\vec{\theta}; X, Z)]$

Maximization step: Find the parameter to maximize the expectation

$\vec{\theta}^{(t+1)} = argmax_{\vec{\theta}} Q(\vec{\theta}|\vec{\theta}^{(t)})$

However, the expectation-maximization algorithm is also very sensitive to the initial selection of parameters. It also suffers from the possibility of converging to a local optimum and the slow convergence rate.

Density-based clustering

Density based lustering algorithms define clusters as areas of higher density than the remainder area. The low density areas are most likely to be bonders or noise region. In this way, the algorithm should handle the noise and outliers quite well since their relevant density should be too low to pass a threshold. The most popular density based algorithm is DBSCAN[17](for density-based spatial clustering of applications with noise). DBSCAN defines a cluster based on "density reachability": a point q is directly density-reachable from point p if distance between p and q is no larger that a given radius ϵ. Further if p is surrounded by sufficiently many points, one may consider the two points in a cluster. So the algorithm requires two parameters: ϵ and the minimum of points to form a cluster (minPts). Then the algorithm can start from any random point measuring the ϵ neighborhood. The pseudo-code of the algorithm is presented below:

$DBSCAN(D, \epsilon, MinPts)$

$C = 0$

for each unvisited point P in dataset D

mark P as visited

$NeighborPts = regionQuery(P, \epsilon)$

if $sizeof(NeighborPts) < MinPts$

mark P as NOISE

else

$C = nextcluster$

$expandCluster(P, NeighborPts, C, \epsilon, MinPts)$

add P to cluster C

for each point P' in $NeighborPts$

if P' is not visited

mark P' as visited

$NeighborPts' = regionQuery(P', \epsilon)$

if $sizeof(NeighborPts') >= MinPts$

$NeighborPts = NeighborPts$ joined with $NeighborPts'$

if P' is not yet member of any cluster

add P' to cluster C

$regionQuery(P, \epsilon)$

return all points within P's ϵ-neighborhood

It is good for the DBSCAN that it requires no specific cluster number information and can find arbitrarily shaped clusters. It handles outliers and noise quite well. However, it depends highly on the distance measure and fails to cluster datasets with large differences in densities. Actually, by using a predefined reachability radius, the algorithm is not flexible enough. Also, it relies on some kind of density drop to detect cluster borders and it can not detect intrinsic cluster structures.

Spectral clustering

Spectral clustering techniques use the spectrum(eigenvalues) of the proximity matrix to perform dimensionality reduction to the datasets. One prominent algorithm of this category is Normalized Cuts algorithm[4], commonly used for image segmentation. We will introduce the mathematical model of the algorithm below:

Let $G = (V, E)$ be a weighted graph, and A and B as two subgroups of the vertices.

$A \bigcup B = V, A \bigcap B = \emptyset$

$cut(A, B) = \sum_{u \in A, v \in B} w(u, v)$

$assoc(A, V) = \sum_{u \in A, t \in V} w(u, t)$ and $assoc(B, V) = \sum_{v \in B, t \in V} w(v, t)$

then $Ncut(A, B) = \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(B,A)}{assoc(B,V)}$

Let $d(i) = \sum_j w_{ij}$ and D be an $n \times n$ diagonal matrix with d on the diagonal, then after some algebric manipulations, we get

$min_A Ncut(A, B) = min_{\vec{y}} \frac{\vec{y}^T(D-W)\vec{y}}{\vec{y}^T D\vec{y}}$, where

$y_i \in \{1, -b\}$ for some constant b, and $\vec{y}^T D\vec{1} = 0$

To minimize $\frac{\vec{y}^T(D-W)\vec{y}}{\vec{y}^T D\vec{y}}$, and avoid the NP-hard problem by relaxing the constraints on $\vec{y}$, the relaxed problem is to solve the eigenvalue problem $(D - W)\vec{y} = \lambda D\vec{y}$ for the second smallest generalized eigenvalue.

And finally bipartition the graph according to this second smallest eigenvalue.

The algorithm has solid mathematical background. However, calculating the eigenvalues and eigenvectors takes lots of computation. Especially if there are some tiny fluctuations in the data along time, the whole calculating process must be redone all over again, which makes calculations in the past useless.

1.3.2 Limitations of Current Classical Clustering Techniques

With the introduction and analysis on current existing clustering algorithms, we can conclude with some limitations of current classical clustering techniques:

1. Many algorithms require the input of cluster number. This is acceptable when the data is already well known, however, a huge problem otherwise.

2. The sensitivity to outliers and noise often influences the clustering results.

3. Some of the current algorithms can not adapt to clusters with different density or clusters with arbitrary shape.

4. Almost all algorithms adopt pure mathematical analysis, that are infeasible to be combined with dynamic systems and control theory. The lack of time dimension makes most of the algorithms a "one-time-deal", of which past calculations can not provide reference to future computations.

Actually, there are many examples in nature, like herds of animals, flocks of birds, schools of fish and colonies of bacteria, that seem to deal quite well with those prob-

lems. The robustness and flexibility of natural clusters far exceed our artificial algorithms. Consequently, it is natural to take a look at the nature to look for inspirations of a new algorithm that handles those problems well.

1.4 Inspirations from Nature

Nature is a great resource to take a look at when we want to find a bridge between computational algorithms and dynamic systems. On one hand, a lot of biological behaviors themselves are intrinsically dynamic systems. We can model these biological behaviors into physical relations and moreover into dynamic equations. Actually, many biomimic techniques such as sonar technology and flapping aircrafts are developed following this path. The close bonding between biology behaviors and dynamic systems makes it possible to not only simulate biology processes through dynamic models, but also control such processes toward a desired goal or result. Thus, it would be highly possible to develop control algorithms for biomimic systems in a biological sense, which is able to achieve efficient, accurate and flexible control performance.

On the other hand, computer science has been benefitting from learning from the nature for a long history. Similar mechanisms and requirements are shared by computational science and biology, which provide a base for cultivating various joint applications related to coordination, network analysis, tracking, vision and etc.[18]. Neuro networks[19][20], which has been broadly applied and examined in machine learning applications such as image segmentation, pattern recognition and even robot control, is a convincing example derived from ideas about the activities of neurons in the brain; Genetic algorithms[21], which is a search heuristic mimics the process of natural evolution, such as inheritance, mutation, selection, and crossover, have been widely applied over the last 20 years. Also, there have been many valuable researches motivated by the inspirations of social behaviors existing among social insects and particle swarms, which we will discuss in details in the following section 1.4.1.

There are several principles of biological systems that make it a perfect bridge connecting dynamic system control and machine learning algorithms[18]. First, dynamic

systems require real-time control strategy, which most modern machine learning algorithms fail to meet because that advanced and resource-demanding mathmetical tools are involved. However, a machine learning algorithm inspired by biology process is most likely to be feasible to be converted into a dynamically evolving process, such as ant colony algorithms and simulated annealing algorithms. Secondly, biological processes need to be able to handle failures and attacks successfully in order to survive and thrive. This property shares the same or similar concept with the robustness requirements for computation algorithms and also stability and resistance to noise for dynamic systems control strategy. So we may use the biological bridge connecting these properties in different fields and complement them with each other. Thirdly, biological processes are mostly distributed systems, such as molecules, cells, or organisms. The interaction, coordination, among the agents along with local decisions make each agent not only an information processing unit, but also a dynamic unit. This is very promising for connecting information techniques with dynamic control, and also for bringing in more intelligence into mechanical engineering field.

These shared principles indicate that bridging dynamic system control with machine learning through biology inpired algorithms is very promising. Such connection may improve understanding in both fields. Especially, the computational algorithms focus on speed, which requires utilizing advanced mathematical tools, but simultaneously makes the algorithm less flexible to time varying changes from both data side and environments. Since biological systems can inherently adapt to changing environments and constrains, and also they are able to make local decisions with limited knowledge, yet optimize towards a global goal, they are ideal examples to learn from for machine learning algorithm to realize large scale self organizing learning. In the following section 1.4.1, we will introduce some previous work utilizing this fact for building swarm intelligence.

1.4.1 Swarm Intelligence

Swarm intelligence is a mechanism that natural or artificial systems composed of many individuals behave in a collective, decentralized and self-organized way. In swarm

intelligence systems, a population of simple agents interact locally with each other and with the environment. Although there is no centralized control structure dictating individual behavior, local interactions between the agents lead to the emergence of global intelligent behavior. This kind of simple local decision policy hugely reduces the computational load on each information processing unit. Such mechanism pervades in nature, including social insects, bird flocking, animal herding, bacterial growth, fish schooling, and etc. Many scientific and engineering swarm intelligence studies emerge in the fields of clustering behavior of ants, nest building behavior of termites, flocking and schooling in birds and fish, ant colony optimization, particle swarm optimization, swarm-based network management, cooperative behavior in swarms of robots, and so on. And various computational algorithms have been inspired by the facts. For the swarm intelligence algorithms, we now have: Altruism algorithm[22], Ant colony optimization[23], Artificial bee colony algorithm[24], Bat algorithm[25], Firefly Algorithm[25], Particle swarm optimization[26], Krill Herd Algorithm[27], and so on.

The natural inspiration that we take in this thesis is Quorum Sensing, which will be described in details in section 1.4.2.

1.4.2 Quorum Sensing

Quorum Sensing[28][29][30][31][32][33][34][35] is a biological process, by which a community of bacteria cells interact and coordinate with their neighboring cells locally with no awareness of global information. This kind of local interaction is not achieved through direct cell-to-cell communication. Actually, each cell sends out signaling molecules called autoinducers that diffuse in the local environment and builds up local concentration. These auto inducers that carry introduction information can be captured by the receptors, who can activate transcription of certain genes that are equipped in the cell. In V. fisheri cells, the receptor is LuxR. There is a low likelihood of a bacterium detecting its own secreted inducer. When only a few cells of the same kind are present in the neighborhood, diffusion can reduce the density of the inducers to a low level, so that no functional behavior will be initiated, which is "energy

efficient". However, when the concentration of the surrounding area reaches a threshold, more and more inducers will be synthesized and trigger a positive feedback loop, to fully activate the receptors. Almost at the same time, specific genes begin being trascripted in all the cells in the local colony, and the function or behavior expressed by the genes will be performed collectively and coordinatedly in the swarm. For instance, if only one single Vibrio fischeri exists in the enviroment, then producing the bioluminescent luciferase would be a waste of energy. However when a quorum in vicinity is confirmed, such collective production can be useful and functional. Fig.1-1 gives a pictorial view of the process in Vibrio fischeri.

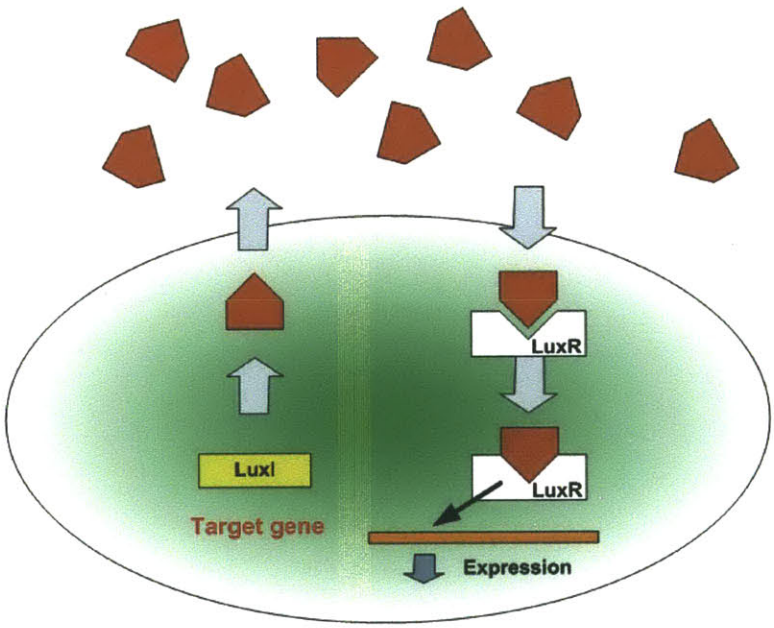

Figure 1-1: Quorum sensing model

In this context, quorum sensing is applicable in developing clustering algorithms, since the cells can sense when they are surrounded in a colony by measuring the autoinducer concentration with the response regulators. A clustering method inspired by quorum sensing would have the following advantages:

1. Since each cell makes decision based on local information, an algorithm could be designed to fit parallel and distributed computing, which is computationally efficient.

2. Since quorum sensing is a naturally dynamic process, such algorithm would be flexible for cluster shape-shifting or cluster merging, which means a dynamic clustering algorithm would be possible to develop.

3. Since the whole process can be modeled in a dynamic view, it would be much easier to incorporate the algorithm into real dynamic swarms control, such as groups of oscillators or swarms of robots, where most of current algorithms fail. So our thesis targets on the problem of developing a clustering algorithm inspired by quorum sensing and connects it with dynamic control strategies.

Chapter 2

Algorithm Inspired by Quorum Sensing

In this chapter, we are going to present the proposed algorithm inspired by quorum sensing. To design the algorithm, we need to first model the biological process of quorum sensing, including the pheromone distribution model, the local reacting policy, the colony establishments, the interactions between and inside colonies and colony branching and merging processes, which will be introduced respectively in section 2.1. For the algorithm inspired by nature, we provide adequate mathematical analysis on the diffusion tuning part and the colony interaction part. In the end of this chapter, we will compare with existing algorithms, to show that our algorithm is reasonable from both a biological view and a mathematical view in section 2.2.

2.1 Dynamic Model of Quorum Sensing

To model quorum sensing into a dynamic system, we need to have a review of the characteristics of the process, make some assumptions and also define our goal for the algorithm. For the quorum sensing, we can extract the following key principles by analyzing the process:

1. Every cell interacts not directly with other cells, but actually with its local

environment or medium. This kind of interaction is undertaken through tuning the ability to secrete pheromones that carry some information introducing themselves for species or some other genetic information.

2. For all the cells, they only know their local environment with no global awareness: where the other cells live and how they behave are unknown to any single cell. It is this global unawareness that makes it possible to make decisions locally and keep the whole process decentralized.

3. When the local density exceeds the threshold, it is confirmed that the cell currently lives in a colony, then relevant genes will be transcripted and all the cells in this colony will perform collective behavior.

4. Cells of certain species can sense not only pheromone molecules of their own kind, they can also receive autoinducers of other species to determine its local environment as enemy existing or other status.

Based on the principles introduced above about quorum sensing, we are proposing an algorithm to mimic the bio-behavior realizing dynamic clustering:

1. Every single cell expands its influence by increasing an "influence radius" as σ in its density distribution function. This can also be regarded as the exploration stage, since all the cells are reaching out to check whether there is a colony nearby.

2. When the density of some cells reaches a threshold, a core cell and a colony are established simultaneously, then it begins to radiate certain kinds of "pheromones" to influence its neighboring cells and further spread the recognition through local affinity. Any influence from an established colony would also stop the other cells from becoming new core cells, so that there are not too many core cells nor too many colonies.

3. Colonies existing in the environment interact with each other to minimize a cost function that can achieve optimized clustering results. In the mean time

some colonies may be merged into others and some may even be eradicated by others.

4. Finally get the clustering result by analyzing the colony vector of each cell.

We will introduce each part of the above proposal in the following sections 2.1.1-2.1.5.

2.1.1 Gaussian Distributed Density Diffusion

To simulate the decentralized process of quorum sensing, we are treating every data as a single cell in the environment. And to describe secretion of pheromones, we are using the Gaussian distribution kernel function as:

$$f(x, x_i) = e^{-\frac{(x-x_i)^2}{{\sigma_i}^2}} \tag{2.1}$$

We are using the Gaussian distribution here because:

1. The Gaussian distribution constrains the influence of any single data in a local environment, which is an advantage for developing algorithms since outliers would have limited impact on the whole dataset.

2. The Gaussian kernels have also been broadly used in supervised learning methods like Regularized Least Squares, and Support Vector Machines, for mapping the data into a higher dimension space, in which clusters are linearly separable. While in these algorithms, learning is achieved by tuning a coefficient vector corresponding to each data, in our algorithm, we are proposing to tune the σ_i's for each data which is a deeper learning into the topology structure of the dataset.

3. Gaussian distribution is a proper model for quorum sensing. The σ_i's can be considered as "influence radius" representing the secreting ability of each cell. Also, the influence from one cell to another is naturally upper bounded at 1, even if two cells are near each other and the influence radius is very large.

By using the Gaussian kernel function, we can map all the data into a kernel matrix $M_{n\times n}$, where for $i \neq j$,

$$m_{ij} = f(x_i, x_j) = e^{-\frac{(x_j - x_i)^2}{{\sigma_j}^2}} \tag{2.2}$$

So m_{ij} is the influence of pheromones from cell j to cell i, while we make $m_{ii} = 0$ since cells tend to ignore their self secreted auto-inducers. Also, by treating $f(x, x_j) = e^{-\frac{(x - x_j)^2}{{\sigma_j}^2}}$ as cell i's influence on environment, m_{ij} is the indirect influence on cell j from the environment owing to the impact from cell i. Moreover,

$$\sum_{j\neq i}^{n} m_{ij} = \sum_{j\neq i}^{n} e^{-\frac{(x_j - x_i)^2}{{\sigma_j}^2}} \tag{2.3}$$

is the influence on cell i from all other cells, or in another word, the local density of cell i in the environment. We use a density vector $\vec{d} = M \times \vec{1}_{n\times 1}$, where $d_i = \sum_{j\neq i}^{n} m_{ij}$ to represent the density of all the cells. The density of a cell describes the "reachability" of a cell from others. In a directed network graph scenario, it's the total received edge weights of a single node. Also it represents the local connectivity of a cell which is quite meaningful if we think of a cluster as a colony of locally and continually connected data. If the density of a certain cell is high, we can say this cell is "well recognized" by its neighbors, and for the cell itself, it can make the judgment that it is located in a well established colony. One thing also worth noting is that, since we only want the cells connected with their local neighbors, it will be wiser to set a threshold on m_{ij} to make the matrix M much sparser, such as if $m_{ij} < 0.1$ then update m_{ij} as zero in the M matrix.

So the upcoming problem is how to develop the local policy for tuning the σ_i's, which will influence the density vector, to simulate quorum sensing and eventually realize data clustering. This problem will be introduced in the following section.

2.1.2 Local Decision for Diffusion Radius

We propose the local tuning policy as the following evolution equation:

$$\dot{\vec{\sigma}} = M(a \cdot \vec{1}_{n\times 1} - M \cdot \vec{1}_{n\times 1}) + \beta(M - D)\vec{\sigma} + \vec{f}_{init} \tag{2.4}$$

As introduced in the previous section, $M \cdot \vec{1}_{n\times 1}$ represents the local density of all the cells. So if we set a density goal vector as $a \cdot \vec{1}_{n\times 1}$, then $a \cdot \vec{1}_{n\times 1} - M \cdot \vec{1}_{n\times 1}$ is the difference vector, which can also be considered as the "hunger factor" vector. If the hunger factor of a certain cell is zero, above zero or below zero, we can say this cell is well recognized, over recognized or poorly recognized by the environment respectively. The "hunger factor" information can be carried in the secreted molecules since all the needed inputs are the local density which can be sensed from local environment. Moreover, $M(a \cdot \vec{1}_{n\times 1} - M \cdot \vec{1}_{n\times 1})$ is the local hunger factor vector accumulated in the location of all the cells, which is the actuation force tuning the $\vec{\sigma}$. When the local environment appears to be "hungry" (local hunger factor is positive), the cell tends to increase its influence radius to satisfy the demand, and on the hand when the local environment appears to be "over sufficient" (local hunger factor is negative), the cell tends to decrease its influence radius to preserve the balance.

For the second component in the evolution equation, $\beta(M - D)\vec{\sigma}$, we add in the term based on the assumption that cells in a neighborhood should share similar secreting ability. The D matrix here is a diagonal matrix, with the entries $D_{ii} = \sum_{j\neq i}^{n} m_{ij}$. So actually, the ith term in the vector $\beta(M - D)\vec{\sigma}$ equals to $\sum_{j\neq i}^{n} m_{ij}(\sigma_j - \sigma_i)$, which is diffusive bonding that is not necessarily symmetric. Adding this term here would help keep influence radius in a neighborhood similar with each other. In the nature, cells belonging to the same colony share similar biological characteristics, which in our case, can be used to restrain the influence radius to be similar with the neighborhood. Also, in the nature, any cell has an upper bound for the influence radius. No cell can secrete enough auto-inducers to influence all the cells in the whole environment, due to its capability and environment dissipation. Since we are not aware of this upper bound, or in a statistics view, the variance in a local environment,

it is hard to find a proper parameter to describe the upper bound. Actually, this is also a huge problem for many other Gaussian kernel based algorithms, since there is no plausible estimation for the σ_i's in the Gaussian kernel. In our algorithm, we use this mutual bonding between cells to provide a local constraint on the upper bound of the σ_i's. It also helps to bring the dynamic system to stability, which will be explained later in the mathematical part.

The third part of the equation provides initial perturbation or actuation to the system, and it will disappear when the system enters into a state that we can consider most of the cells living in a local colony. When the system starts from $\vec{\sigma} = \vec{0}$, or a very close region around the origin, it is obvious that for all the entries in the M matrix, $m_{ij} \approx 0$, and also $\beta(M - D)\vec{\sigma} \approx \vec{0}$, so that the system will stay around the origin or evolve slowly, which is not acceptable for an algorithm. So we add in this initial actuation term $\vec{f}_{init} = 0.5a \times \vec{1}_{n\times 1} - M \times \vec{1}_{n\times 1}$, which is used to measure whether a cell is already recognized by other cells, so that for the neighbors around cell i, the incoming density $\sum_{j\neq i}^{n} m_{ij}$ reaches a certain level. When $\vec{f}_{init}$ reaches around 0, the former two components in the evolving equation have already begun impacting, so the initial actuation can disappear from then on. Consequently, we can choose a standard when the initial actuation disappears, such as if $\frac{\sum_{i}^{n} d_i}{n} > 2$, we can regard most cells recognized by the environment and cancel the initial actuation. The explanation of initial actuation also can be found in nature: when cells are transferred into an entirely new environment, they will secrete the pheromones to see whether they currently live in a quorum. We can think of this process as an exploration stage of quorum sensing when stable interactions between cells and colonies have not yet been established. The natural exploration stage is restrained by cells' secreting capability. Also in our algorithm, it is constrained by the early stopping rule, so that no cell can explore indefinitely, and thus outliers data would regard themselves as "not surrounded", and impact little on well established colonies.

Finally, the combination of all the three components above forms the local evolving rule modeling the quorum sensing policy. It is worth noting that, no matter what extra information is included in the virtual pheromones, such as the "hunger factor" or the

local influence radius, for the receptors, they can sense the information just from the environment without knowing the origin of any pheromones nor their locations. This policy modulates cell-to-cell communications into cell-to-environment interactions, which makes more sense for distributed computation. Based on the results we can achieve as a sparsely connected M matrix, and in the following section, we will introduce details about colony interactions.

2.1.3 Colony Establishments and Interactions

In quorum sensing, when the concentration surpass a predefined threshold, cells in the colony will begin to produce relevant functional genes to perform group behavior. In our algorithm, we use this trait as the criterion for establishing a colony. When the density of a cell d_i surpasses a predefined threshold $b \leq a$, and the cell is not recognized by any colony, then we can establish a new colony originating from this cell and build a $n \times 1$ colony vector, with the only non-zero entry as 1 in the ith term. The origin cell will be regarded as a core cell. For example, if the jth colony derives from the ith cell, then a new vector $\vec{c_j}$ will be added into the colony matrix C, where $C = [\vec{c_1}, \vec{c_2}, ..., \vec{c_{i-1}}, \vec{c_i}]$. The only non-zero entry in $\vec{c_j}$ is its ith term, which is also C_{ij} that is initialized as 1. In the meantime, the colony vectors keep evolving following the rules:

$$\dot{\vec{c_i}} = -\sum_{j \neq i}(M + M^T)\vec{c_j} + \gamma(M + M^T)\vec{c_i} \tag{2.5}$$

which can also be written as: $\dot{\vec{c_i}} = -(M + M^T)(\vec{c_e} - \vec{c_i}) + \gamma(M + M^T)\vec{c_i}$, where $\vec{c_e} = \sum \vec{c_i}$

Summing all the colony vectors to achieve the environmental colony vector $\vec{c_e}$ also follows the idea of quorum sensing to simplify the calculation through using global variable updates instead of calculating every component. All entries in the colony vectors are saturated in the $[0, 1]$ range, which means that they will not increase after reaching 1, nor decrease after reaching 0. It is also worth noting that $\vec{c_e}$ is the criterion judging whether a potential core cell candidate has been recognized by existing colonies or not. Moreover, for the interaction equations, we can also write

the series of interaction equations in a matrix view:

$$\dot{C} = -(M + M^T)(\vec{c_e}\vec{1}_{1\times k} - C) + \gamma(M + M^T)C \tag{2.6}$$

where k is the number of existing colonies. As we can see from the equation, the interaction of colonies is consisted of two parts: the first part is a single colony to environments interaction; and the second part is a single colony to self evolving. The equation is designed to realized a continuous optimization on a Normalized Cuts alike cost function. We will introduce the mathematical analysis on the two parts and also the parameter γ in more details in the later section 2.2.2.

2.1.4 Colony Merging and Splitting

After the cell interactions and colony interactions above, we will have a sparsely and locally connected weighted graph, described by matrix M, and some distributed core cells along with related colonies. Among these distributed colonies, some may be well connected to each other; some may share a small amount of cells in each colony; or maybe a part of one colony moved from the previous colony into another one; or maybe there are new colonies appear. These scenarios require rules for merging the colonies parts and updating existing colony numbers. The criterion we are using is similar with the Normalized Cuts method by calculating a ratio between inter colony connections and inner colony connections, which is meant to describe the merging potential for one colony into another one:

$$r_{ij} = \frac{\vec{c_i}^T(M + M^T)\vec{c_j}}{\vec{c_i}^T(M + M^T)\vec{c_i}} \tag{2.7}$$

We can set a threshold, such that if there exists any $r_{ij} > 0.2, i \neq j$, then we can merge the colony i into colony j. Also if we have a predefined number of clusters and current cluster number is larger than the expected one, we can choose the largest r_{ij} and merge colony i into colony j.

On the other hand, there will be occasions that new clusters are split from previous

colonies, or a previous cluster is not continuously connected any more. As we mentioned previously in section 2.1.1, we can set a threshold to make the M matrix much sparser. This threshold can also make sure that inter colony connections between two well segmented clusters are all zero, which also means that certain pheromones from one colony can not diffuse into other colonies. Thus, to detect whether a new cluster appears, we can set a continuity detecting vector $\vec{s_i}$ for each colony with the only non-zero entry corresponding to the colony's core cell as 1. The evolution of continuity detecting vectors also follows the rules of colony interactions:

$$\dot{\vec{s_i}} = -(M + M^T)(\vec{s_e} - \vec{s_i}) + \gamma(M + M^T)\vec{s_i} \tag{2.8}$$

where $\vec{s_e} = \sum \vec{s_i}$ and also in a matrix view,

$$\dot{S} = -(M + M^T)(\vec{s_e}\vec{1}_{1\times k} - S) + \gamma(M + M^T)S \tag{2.9}$$

When the saturated continuity detecting process reaches a stable equilibrium, that $\dot{S}$ has no non-zero entry, we restart the process all over again. Actually, this detecting process is to redo the colony interaction part again and again, while preserving the current clustering results in C matrix. Cells indentified as outliers in every continuity detecting iteration will be marked as "not recognized by any colony" and will be available for forming new colony following the colony establishment process introduced in section 2.1.2. Since the calculation for interactions between colonies is only a relatively small amount of the whole algorithm, such update will not influence much on the overall computing speed.

2.1.5 Clustering Result

Finally, we can get the clustering by analyzing the colony vectors. By choosing the maximum entry of each row in matrix C, and finding out to which column it belongs to for each cell, we can determine the belongings of each cell among the existing colonies. If for some cell, the related row in C is a zero vector, then the cell or rep-

resenting data can be regarded as an outlier. The scenario that there are more than one non zero entries in a row in C will not happen if we tune the parameter γ to 1, which we will explain later in section 2.2.2. Thus, we can achieve the clustering results by simply counting the C matrix, without further K-means process used by other algorithms such as Power Iteration Clustering.

Pseudo Code:

---

1. Initialize $\vec{\sigma}$ as $\vec{0}$, form the M matrix, set the parameters a, b, β, γ

2. Begin the process:

$\dot{\vec{\sigma}} = M(a \cdot \vec{1}_{n\times 1} - M \cdot \vec{1}_{n\times 1}) + \beta(M - D)\vec{\sigma} + \vec{f}_{init}$

Detect new cluster:

if $\exists d_i > b(b \leq a)$ and cell i not recognized by any colony

create a new colony using cell i as core cell

end

$\dot{C} = -(M + M^T)(\vec{c_e}\vec{1}_{1\times k} - C) + \gamma(M + M^T)C$

$\dot{S} = -(M + M^T)(\vec{s_e}\vec{1}_{1\times k} - S) + \gamma(M + M^T)S$

Cluster segmented detection:

if in the stable state, $S \neq C$

update C = S and accept new born clusters

end

$r_{ij} = \frac{\vec{c_i}^T(M+M^T)\vec{c_j}}{\vec{c_i}^T(M+M^T)\vec{c_i}}$

Cluster merging:

if $\exists r_{ij} > 0.2, i \neq j$

then we can merge the colony i into colony j

end

3. Achieve the clustering results by counting the C matrix

2.2 Mathematical Analysis

In this section, we will introduce the mathematical background of the proposed algorithm. We will show that the influence radius tuning policy is an approximation for the optimization of a goal function $\|a \cdot \vec{1}_{n\times 1} - M \cdot \vec{1}_{n\times 1}\|$. For the colony interactions, the designed rule is meant to optimize a Normalized Cuts alike cost function, while the merging policy follows the same goal. Finally, we provide some analysis on analogies to other algorithms, such as spectral clustering, power iteration clustering and normalized cuts.

2.2.1 Convergence of Diffusion Tuning

As we introduced in section 2.1.2, the local tuning policy is:

$$\dot{\vec{\sigma}} = M(a \cdot \vec{1}_{n\times 1} - M \cdot \vec{1}_{n\times 1}) + \beta(M - D)\vec{\sigma} + \vec{f}_{init}$$

Through the tuning of influence radius vector, finally we can get the local density of each cell around a predefined value a, while we can also allow some errors, so that outliers or local dense groups won't harm the overall result. Actually, if we want to make sure that every cell's local reaches exactly at a, we can minimize the $\|a \cdot \vec{1}_{n\times 1} - M \cdot \vec{1}_{n\times 1}\|^2$.

$$\begin{aligned}
\frac{d}{dt}\|a \cdot \vec{1}_{n\times 1} - M \cdot \vec{1}_{n\times 1}\|^2 &= \frac{d}{dt}(a \cdot \vec{1}_{n\times 1} - M \cdot \vec{1}_{n\times 1})^T(a \cdot \vec{1}_{n\times 1} - M \cdot \vec{1}_{n\times 1}) \\
&= -2(a \cdot \vec{1}_{n\times 1} - M \cdot \vec{1}_{n\times 1})^T \frac{d}{dt}(M \cdot \vec{1}_{n\times 1}) \\
&= -2(a \cdot \vec{1}_{n\times 1} - M \cdot \vec{1}_{n\times 1})^T (\frac{\partial}{\partial \vec{\sigma}} M \cdot \vec{1}_{n\times 1})\dot{\vec{\sigma}} \qquad (2.10)
\end{aligned}$$

Thus, if $\dot{\vec{\sigma}} = (\frac{\partial}{\partial \vec{\sigma}} M \cdot \vec{1}_{n\times 1})^T(a \cdot \vec{1}_{n\times 1} - M \cdot \vec{1}_{n\times 1})(*)$, then

$$\begin{aligned}
\frac{d}{dt}\|a \cdot \vec{1}_{n\times 1} - M \cdot \vec{1}_{n\times 1}\|^2 &= -2(a \cdot \vec{1}_{n\times 1} - M \cdot \vec{1}_{n\times 1})^T (\frac{\partial}{\partial \vec{\sigma}} M \cdot \vec{1}_{n\times 1})(\frac{\partial}{\partial \vec{\sigma}} M \cdot \vec{1}_{n\times 1})^T \\
&\quad \cdot (a \cdot \vec{1}_{n\times 1} - M \cdot \vec{1}_{n\times 1}) \leq 0
\end{aligned}$$

We name the Jacobian matrix as $J = (\frac{\partial}{\partial \vec{\sigma}} M \cdot \vec{1}_{n\times 1})$, then $J_{ij} = \frac{2(x_i - x_j)^2}{\sigma_j^3} e^{-\frac{(x_i-x_j)^2}{\sigma_j^2}}, i \neq j$, and $J_{ii} = 0$. So we can have:

$$\dot{\sigma_i} = 2 \sum_{j \neq i} \frac{(x_i - x_j)^2}{\sigma_i^3} e^{-\frac{(x_i-x_j)^2}{\sigma_i^2}} (a - d_j) \tag{2.11}$$

In the equation, every term is composed of two parts: the $(a - d_j)$ term represents the "hunger factor" of surrounding cells, and the $\frac{(x_i-x_j)^2}{\sigma_j^3} e^{-\frac{(x_i-x_j)^2}{\sigma_j^2}}$ represents the ability of cell i's influence on other cells to satisfy their needs. The latter part reaches to nearly zero when either σ_i is too small to influence on some cells, or when it is already large enough that the influence from cell i already reaches to almost 1, that increasing the influence radius wont help much about solving the "hunger problem".

However, the equation shown above is not a perfect candidate for the tuning policy. On one hand, such optimization is easy to cause the "over-fitting" problem. To achieve the goal that every cell's local density reaches a certain value, we may get some ill-posed results, such as some "super cells" with infinite large influence radius and keeps increasing, while all the other cells' influence radius are reduced to 0. On the other hand, even if we can regularize with the $\beta(M - D)\vec{\sigma}$ to avoid the over-fitting problem, but the whole process is not suitable for distributed computation especially for swarms of robots since for any single agent to make decisions, it will need the complete information of all the other agents including locations. This is all due to the second term which provides useful information on how changing of one agent's radius can influence on the others, which is required for every agent to make decisions based on cell-to-cell feedbacks instead of cell-to-environment feedbacks. Such agent-to-agent communication would form a much more complex network, that makes any algorithm lack of efficiency.

Consequently, in our final proposed algorithm, we are using

$$\dot{\vec{\sigma}} = M(a \cdot \vec{1}_{n\times 1} - M \cdot \vec{1}_{n\times 1}) + \beta(M - D)\vec{\sigma} + \vec{f}_{init}$$

instead as an approximation. By comparing to equation (*), we replace $(\frac{\partial}{\partial \vec{\sigma}} M \cdot \vec{1}_{n\times 1})^T$

with matrix M, and keep the "hunger factor" term. By doing so, our algorithm is much more reasonable in biological view: the pheromones secreted by cells carry the information of hunger degree, and also follows the influence density distribution. In this way, the hunger factor represents the amplitude of a Gaussian distribution function with the same influence radius. This replacement ignores the information of capability that changing cell i's radius will influence on satisfying other cells' demands. It will cause problem in the occasions, such as there are only two cells in a local environment who can sense each other's hunger factor quite well, however, the influence radius will keep increasing because if $a > 1$, their demand will never be satisfied. To deal with this problem, we add in the $\beta(M - D)\vec{\sigma}$ to make sure that the influence radius distribution is more unified within colonies. Also, an early stopping rule of initial actuation is also helpful for avoiding the problem. Also, we can set a dynamic upper bound for σ_i's or add a inhibiting term $-\alpha\vec{\sigma}$ at the end of the tuning equation, so that at any time, no "super cell" will happen despite of local needs.
For the proposed tuning policy:

$$\dot{\vec{\sigma}} = M(a \cdot \vec{1}_{n\times 1} - M \cdot \vec{1}_{n\times 1}) + \beta(M - D)\vec{\sigma} + \vec{f}_{init}$$

The convergence and stability of the system is relatively hard to prove directly. This is because that the Jacobian matrix associated with variable $\vec{\sigma}$ is not feasible to analyze about the eigenvalues for such a complex and nonlinear dynamic system. There is also no typical way to segment the large matrix into feedback, parallel, or hierarchical structures that could possibly reduce the complexity. Also, since this is not typical problems like synchronization or trajectory tracking, powerful tools like contraction analysis and partial contraction analysis are not suitable for theoretical analysis. So for the stability problem, we will provide several supportive theoretical analysis combined with further experiments in Chapter 3 to show that the system can arrive in an acceptable state space region and stay there.
First of all, the system $\dot{\vec{\sigma}} = (\frac{\partial}{\partial\vec{\sigma}}M \cdot \vec{1}_{n\times 1})^T(a \cdot \vec{1}_{n\times 1} - M \cdot \vec{1}_{n\times 1})(*)$ is a stable system.

If we use the Lyapunov function

$$V = \|a \cdot \vec{1}_{n\times 1} - M \cdot \vec{1}_{n\times 1}\|^2 \tag{2.12}$$

then,

$$\dot{V} = -2(a\cdot\vec{1}_{n\times 1} - M\cdot\vec{1}_{n\times 1})^T(\frac{\partial}{\partial\vec{\sigma}}M\cdot\vec{1}_{n\times 1})(\frac{\partial}{\partial\vec{\sigma}}M\cdot\vec{1}_{n\times 1})^T\cdot(a\cdot\vec{1}_{n\times 1} - M\cdot\vec{1}_{n\times 1}) \leq 0 \tag{2.13}$$

and since $\ddot{V}$ is bounded, $\dot{V} \to 0$ asymptotically, and thus the system is stable, that $\vec{\sigma}$ will converge to a certain equilibrium. Although our real algorithm replaces $(\frac{\partial}{\partial\vec{\sigma}}M\cdot\vec{1}_{n\times 1})^T$ with matrix M, we have all other parts remained the same. And also, if we assume that influence radius in the same colony are mostly similar, then $M \approx M^T$, thus all entries in $(\frac{\partial}{\partial\vec{\sigma}}M\cdot\vec{1}_{n\times 1})^T$ only adds a proportional term $\frac{(x_i-x_j)^2}{\sigma_j^3}$ before m_{ij}. So inherently, M should be a good approximation, so that the stability of the real system can be partially inferred from the its approximating equation.

Secondly, adding the term $\beta(M-D)\vec{\sigma}$ would help stabilize the system. As we know that the matrix $(M-D)$ is a semi-negative definite matrix. Although it is impossible to determine the eigenvalue distribution of the first part of the Jacobian matrix $\frac{\partial}{\partial\vec{\sigma}}M(a\cdot\vec{1}_{n\times 1} - M\cdot\vec{1}_{n\times 1})$, by tuning the *beta* into a large enough value, it is plausible to bring the system to a stable equilibrium. Assume that β is so large that all σ_i's in the same cluster are constrained to be the same value. Then $M = M^T$, then for the single variable in a cluster, it is obvious that the system will converge to an equilibrium that averages the demands of the belonging cells and erases this average "hunger" to zero. By tuning β down to an acceptable value, $\beta(M-D)\vec{\sigma}$ provides a soft constraint that cells in a colony have similar influence secreting ability that can be referred to nature. It also helps to stabilize the system by counteracting any positive eigenvalue in the first part of the Jacobian matrix. Moreover, what is more obvious is that we can add the term $-\alpha\vec{\sigma}$ at the end of the tuning equation, not only to help restrain "super cells", but also stabilize the system in a much more direct way. Consequently, the stability and convergence of the tuning equation, although not feasible to be proved directly, can be ensured by tuning relevant parameters and

won't cause any problem to the whole algorithm. Actually, existence of some minor fluctuations in an acceptable range, yet not converging to a static equilibrium, is reasonable from biological view.

2.2.2 Colony Interaction Analysis

In the Normalized Cuts algorithm[4], it is designed to minimize the function:

$$Ncuts(A,B) = \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(B,A)}{assoc(B,V)} \tag{2.14}$$

where

$$cut(A,B) = \sum_{i \in A, j \in B} m_{ij} \tag{2.15}$$

and

$$assoc(A,V) = \sum_{i \in A} m_{ij} \tag{2.16}$$

By normalizing the previously MinCut problem, the unnatural bias for partitioning out small sets of points is avoided. While in our algorithm, we are having a set of distributed core cells, thus by weakening some small colonies down to 1 cell and finally eliminating them, we can also avoid the problem. Also, if we use the Normalized Cuts function, minimizing the function through dynamic evolutions by taking time derivatives would cause more complex problems. Thus we simply use a modified version of MinCut, by adding a term trying to maximize inner colony bonding. This added term not only performs the role of normalization in normalized cuts, but also simulates the process of diffusing the colony recognition from the core cell to its local environment and naturally all connected neighbors. Moreover, we changed the binary clustering problem into a multi cluster interaction problem using continuous colony vectors.

The Colony interactions are meant to minimize a cost function similar to the Nor-

malized Cuts and MinCut:

$$V_{colony} = \sum_{i \neq j} \vec{c_i}^T (M + M^T)\vec{c_j} - \frac{\gamma}{2} \sum_i \vec{c_i}^T (M + M^T)\vec{c_i} \tag{2.17}$$

$$\begin{aligned} \dot{V}_{colony} &= \sum_{i \neq j} \dot{\vec{c_i}}^T (M + M^T)\vec{c_j} - \gamma \sum_i \dot{\vec{c_i}}^T (M + M^T)\vec{c_i} \qquad (2.18) \\ &= \sum_i \dot{\vec{c_i}}^T (M + M^T)(\vec{c_e} - \vec{c_i}) - \gamma \sum_i \dot{\vec{c_i}}^T (M + M^T)\vec{c_i} \\ &= \sum_i \dot{\vec{c_i}}^T (M + M^T)(\vec{c_e} - (\gamma + 1)\vec{c_i}) \end{aligned}$$

Consequently, by making $\dot{\vec{c_i}} = -(M + M^T)\vec{c_e} + (\gamma + 1)(M + M^T)\vec{c_i}$

$$\begin{aligned} \dot{V}_{colony} &= -\sum_i (\vec{c_e} - (\gamma + 1)\vec{c_i})^T (M + M^T)^2 (\vec{c_e} - (\gamma + 1)\vec{c_i}) \\ &\leq 0 \end{aligned}$$

We can also transform the interaction equations into a matrix view:

$$\dot{C} = -(M + M^T)\vec{c_e}\vec{1}_{1 \times k} + (\gamma + 1)(M + M^T)C$$

Also need to remember here that all entries in C are strictly restricted in the range of $[0, 1]$. The interactions between colonies are composed of two parts, one part is the mutual inhibitions between colonies, and the other part is self promotion of one colony into the whole environment through diffusion. When initially the colonies have not yet been well developed, but just established on distributed core cells, it is obvious that there will be no inter colony interactions, but only colony self-expanding through $\dot{C} = (\gamma + 1)(M + M^T)C$. This simulates a neighbor-to-neighbor infection, and this infection realizes that cells recognized already can activate their neighbors and pass on the colony identities. So it is natural that if one cell is already recognized by one colony, then it can not establish a new cluster as a core cell, because it can already be well connected to an existing core cell directly or indirectly. When the colonies have been expanding for a certain period of time, some colonies that are not well separated become neighboring to each other, and the mutual inhibition comes into effect. So finally, it will be a natural balance between inhibitions from other colonies and self

expanding forces, which can be further tuned by the parameter γ.

Furthermore, we can explain the interaction in a micro view at the boundary of two stable neighboring colonies: suppose that we have two colonies A and B closely neighboring with each other, with colony vector as $\vec{c_A}$ and $\vec{c_B}$ respectively. For a single cell i in the boundary area between the two colonies, following the interaction rules,

$$\dot{c}_{Ai} = -\sum_j (m_{ij} + m_{ji}) c_{Bj} + \gamma \sum_j (m_{ij} + m_{ji}) c_{Aj}$$

$$\dot{c}_{Bi} = -\sum_j (m_{ij} + m_{ji}) c_{Aj} + \gamma \sum_j (m_{ij} + m_{ji}) c_{Bj}$$

When $\gamma = 1$, it is obvious that $\dot{c}_{Ai} = -\dot{c}_{Bi}$, so if accumulated bonding from colony A is larger than accumulated bonding from colony B, as $\sigma_j(m_{ij} + m_{ji})c_{Aj} > \sigma_j(m_{ij} + m_{ji})c_{Bj}$, then finally $\dot{c}_{Ai} = 1, \dot{c}_{Bi} = 0$, otherwise, $\dot{c}_{Ai} = 0, \dot{c}_{Bi} = 1$. So if we set $\gamma = 1$, then in the final result every row in matrix C will have at most one non-zero entry as 1, on the column, whose colony has most accumulated bonding weights towards the cell.

So what if $\gamma \neq 1$? Actually, γ is the parameter tuning the relative strength of inhibition and expanding forces, when $\gamma < 1$, the inhibition force is relatively enhanced, naturally, there might exist some blank boundaries between clusters, as the inhibition force from neighboring colonies are so strong that expanding from neither colony could reach the cell. On the other hand, when $\gamma > 1$, the expanding force is relatively enhanced, so it is more easy for the colony factors to spread into well connected neighboring colonies despite of mutual inhibition. As shown in Fig.2-1 γ here measures the ability to "cross the chasm", since as long as the expanding force is over $\frac{1}{\gamma}$ of the inhibition force, the penetration can happen to make the neighboring colonies more connected to each other. So at the beginning, it would be wise to tune up γ, to not only speed up new born colony growing, but also enhance distributed small colonies merging. By strengthening the ability to "cross the chasm", it will also help the algorithm avoid being trapped in local optimum. At the late stage of the process, when all colonies have become stable, we can tune γ back to 1 to achieve a distinct

clustering result.

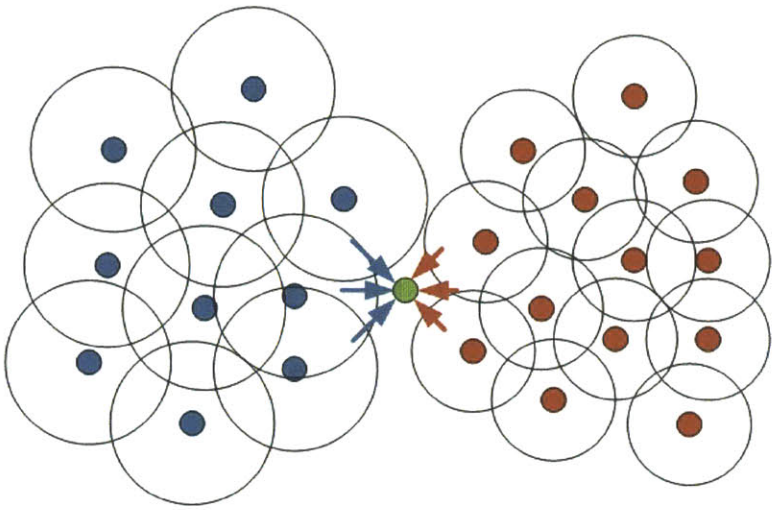

Figure 2-1: The interactions between two colonies

Thus, the algorithm have successfully optimized a cost function similar to Normalized Cuts and MinCut, to coordinate the interactions between colonies so that we can achieve clustering result that is reasonable both in a biological view and a mathematical view.

2.2.3 Analogy to Other Algorithms

Although a purely biology inspired algorithm, our algorithm still has various connections to some other existing algorithms, like density based clustering algorithms(DBSCAN, DENCLUE[36]), and spectral clustering algorithms(Normalized Cuts, Power Iteration Method[37]):

Like DBSCAN algorithm, we also have the concept of growing radius, and also we both regard a cluster as locally well connected groups of data. However, the radius in our algorithm is not used to define a connected territory, yet to influence the local environment in order to tune the M matrix to a sparsely connected graph. Also, our algorithm performs distributed computation and is much more flexible to time varying data.

Similar to the spectral clustering algorithms, like normalized cuts and diffusion maps,

we also utilize the M matrix alone for analysis, and we even borrow the idea from normalized cuts to build a cost function. However, we take different paths to segment the clusters. The spectral clustering algorithms dig deep into the mathmatical analysis on eigenvalue and eigenvector distributions of the M matrix or its modifications, while actually, it can also be explained in the markov process theory. However, we focus more on the local connectivity and diffusion properties that can be inspired by nature. We use similar cost functions to value the quality of clustering results, and it is reasonable to say, the spectral clustering methods provide great theoretical support to our method, and the solution in our way is most likely to be the same when using the second largest eigenvector to segment clusters in normalized cuts. Nonetheless, our algorithm is self organizing, which makes it more adaptive to problems, where σ can not be predefined. Also, we don't need to calculate the eigenvalues again and again if the data is time varying continuously. Actually, when dealing with shifting data, we can think of our clustering result as an integral process where historical results can provide reference information to future result, so that calculations in the past are not wasted. Consequently, in the long run, our algorithm is more efficient and flexible.

Further, we will test our algorithm in various benchmark datasets in Chapter 3.

Chapter 3

Experimental Applications

In this chapter, we will introduce the experiment results of the proposed algorithm. First we will provide clustering results on some typical synthetic benchmarks, whose data structures are not linearly separable, so that cannot be solved by K-means or distribution based algorithms. Then we will test on some real benchmarks applications, and compare with clustering results with current popular algorithms including Normalized Cuts[4], Ng-Jordan-Weiss algorithm[3] and Power Iteration Clustering[37]. Then we will test the algorithm on some novel applications, such as using the algorithm on alleles classifications. And finally, we will discuss the possibility of applying the algorithm on clustering dynamic systems.

3.1 Synthetic Benchmarks Experiments

We provide the clustering results on four occasions that are commonly considered as difficult clustering scenarios: the two-chain model, the double-spiral model, the two moon model and the island model. The parameters setting we are using here are: $a = 5, b = 5, \beta = 2, \gamma = 5$

Two-chain model

The two chain dataset is to test the capability of the algorithm to diffuse the influence in a chain pattern. In the dataset, we have 224 samples forming two chain-shaped data cluster, as shown in Fig.3-1. After tuning the influence radius σ_i's, we can have

two well separated colonies with the density distribution shown in Fig.3-2. We can have the clustering results shown as Fig.3-3.

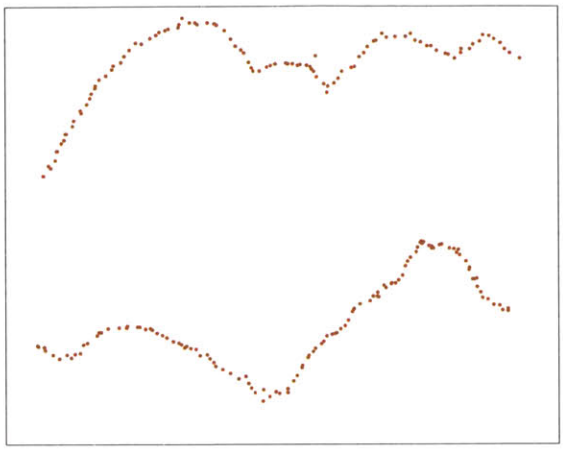

Figure 3-1: The two-chain shaped data model

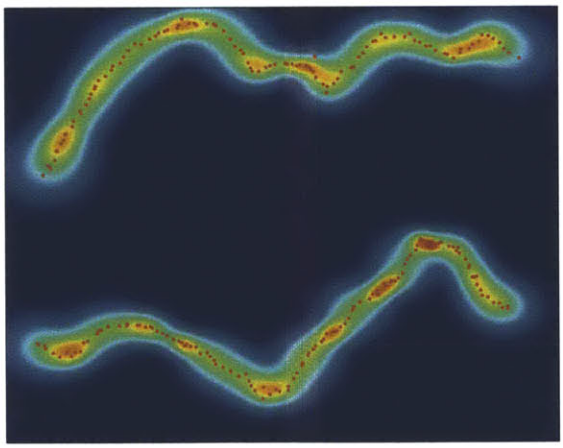

Figure 3-2: Density distribution of two-chain shaped data model

Two-spirals model

Then we try to test the algorithm on more entangled datasets, such as two spirals-shaped data model. In this way, we can make sure that the tuning rules guarantee

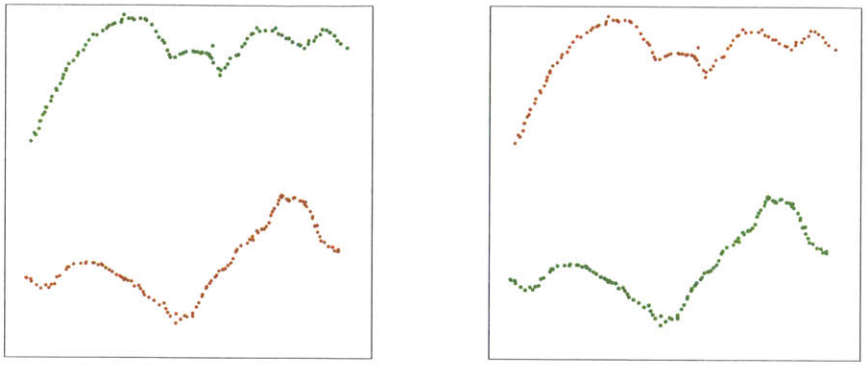

Figure 3-3: Clustering result of the two-chain shaped data model

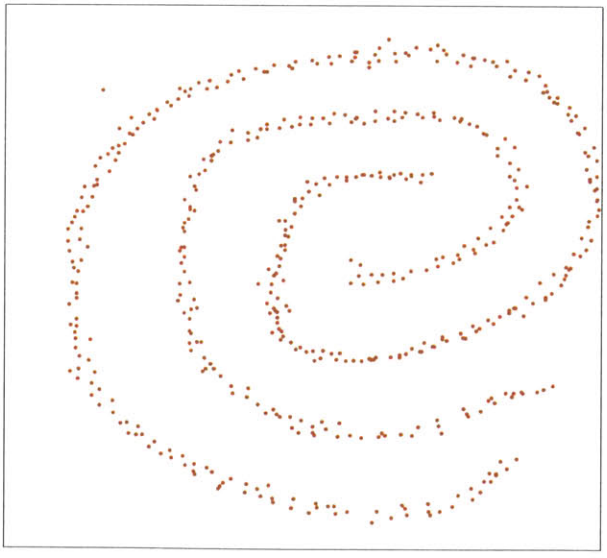

Figure 3-4: The two-spiral shaped data model

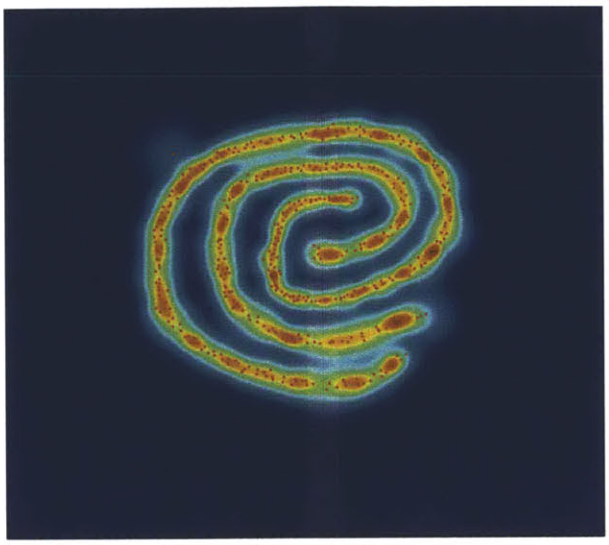

Figure 3-5: Density distribution of two-spiral shaped data model

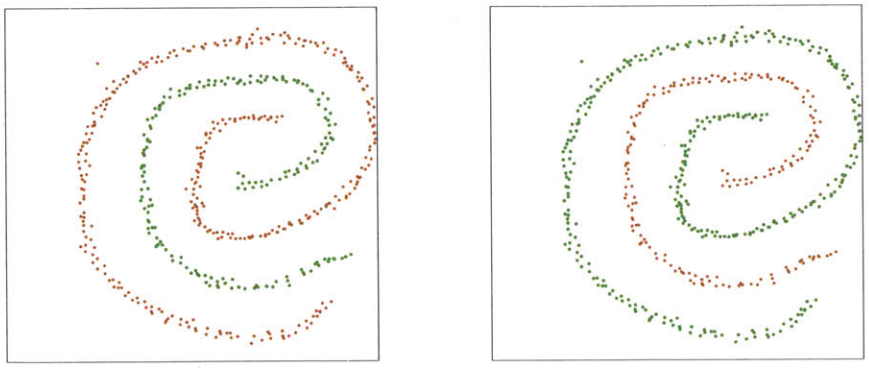

Figure 3-6: Clustering result of the two-spiral shaped data model

that cells are only connected to their local neighbors, and also see whether the colony factors diffuse faster along dense area than sparse area. In the dataset, we have 525 samples distributed in a two-spiral way, as shown in Fig.3-4. After the tuning process, the density distribution map of the double spirals is shown in Fig.3-5. The clustering result of the two clusters is shown in Fig.3-6

Figure 3-7: The two-moon shaped data model

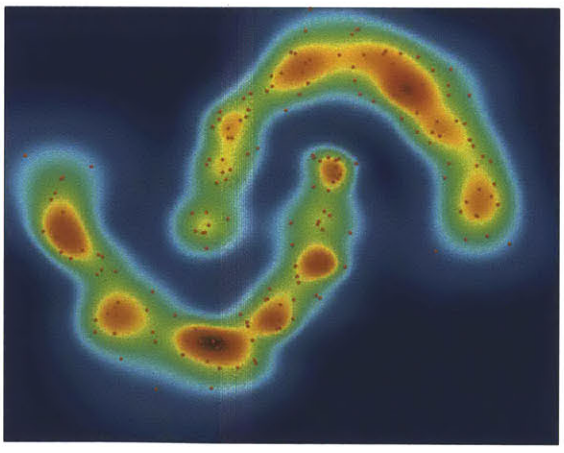

Figure 3-8: Density distribution of two-moon shaped data model

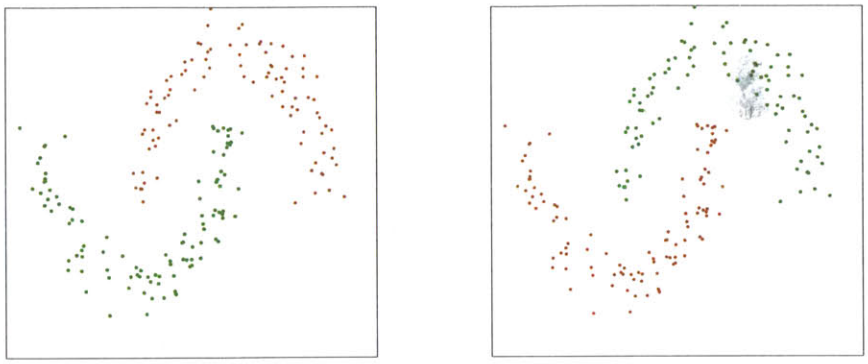

Figure 3-9: Clustering result of the two-moon shaped data model

Two-moon model

To test the performance on more dense datasets, we are using the two-moon data model which is widely used in supervised learning and semi-supervised learning researches. As we can see from Fig.3-7, the dataset is composed of two clusters with 200 samples that can not be linearly separated. Also, it can be segmented into four clusters if considering the separation within the two clusters, which is a result that can also be achieved with our algorithm with other parameter settings. We can see from Fig.3-8, the influence tuning make sure that the final density distribution closely fits the original data distribution, which can be quite useful for further classification and outliers detection. And the clustering result is shown in Fig.3-9.

Island model

Finally, we test our algorithm on a dataset shaped like an island, to make sure that the island cluster can be isolated from the ring like cluster. The 236 samples and the corresponding density maps along with clustering results are shown respectively in Fig.3-10, Fig.3-11, and Fig.3-12.

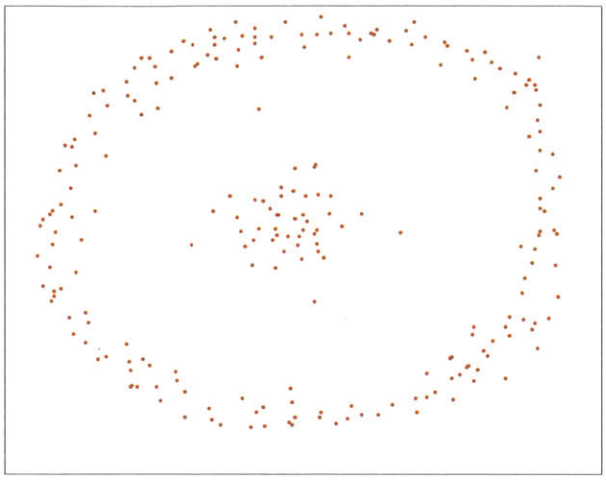

Figure 3-10: The island shaped data model

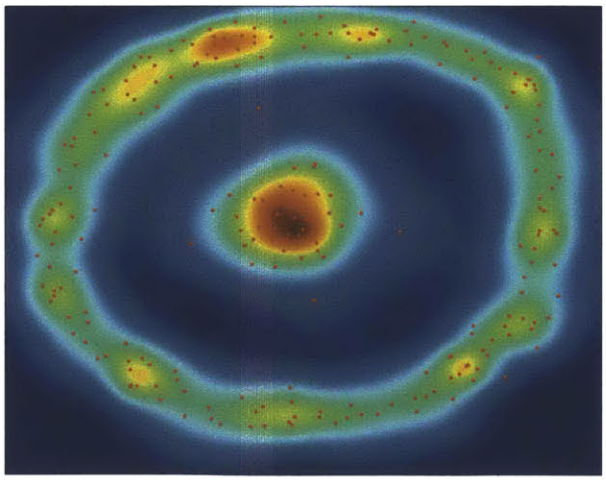

Figure 3-11: Density distribution of island shaped data model

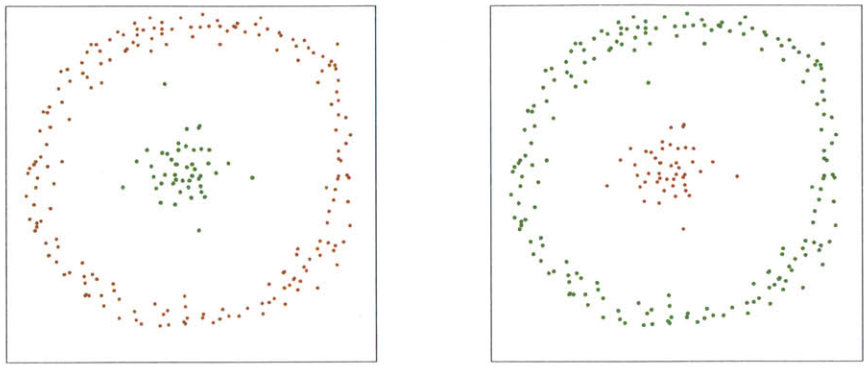

Figure 3-12: Clustering result of the island shaped data model

3.2 Real Benchmarks Experiments

In this section, we will present the experimental results of our algorithm on a variety of real datasets including the Iris datasets and the Pendigits datasets.

Iris flower dataset

The Iris flower dataset or Fisher's Iris dataset[38], introduced by Sir Ronald Fisher(1936), consists of 150 instances forming 3 clusters, of which two are only nonlinearly separable. The dataset is to quantify the morphologic variation of Iris flowers of three species(Iris setosa, Iris virginica and Iris versicolor) on four features, the length and the width of the sepals and petals. The dataset is widely used for measuring supervised clustering algorithms, however, not so popular in the clustering analysis field, since two of the clusters are not easily separable. Actually, in our algorithm, it is also normal that with some parameter settings, we can only detect two clusters out of the dataset. Or sometimes we can get results of three clusters while the segmentation, although not accurate, that are still good segmentations if we just look at the data structure and forget the labeled information. Finally, with the parameters set as: $a = 3, b = 2.5, \beta = 2, \gamma = 2$, we can successfully cluster the dataset into three clusters with 4 errors out of 150, a 97.3% correctness rate. One thing worth noting here is that, the reason we are tuning down a, b and γ is due to the concern that overly diffusing and merging may cause the problem that a third cluster can not be separated

from the dataset. So we tune down a to 3, so that cells will seek less recognition from neighborhood, and we tune down γ, so that clusters will merge less. The clustering result is shown in Fig.3-13

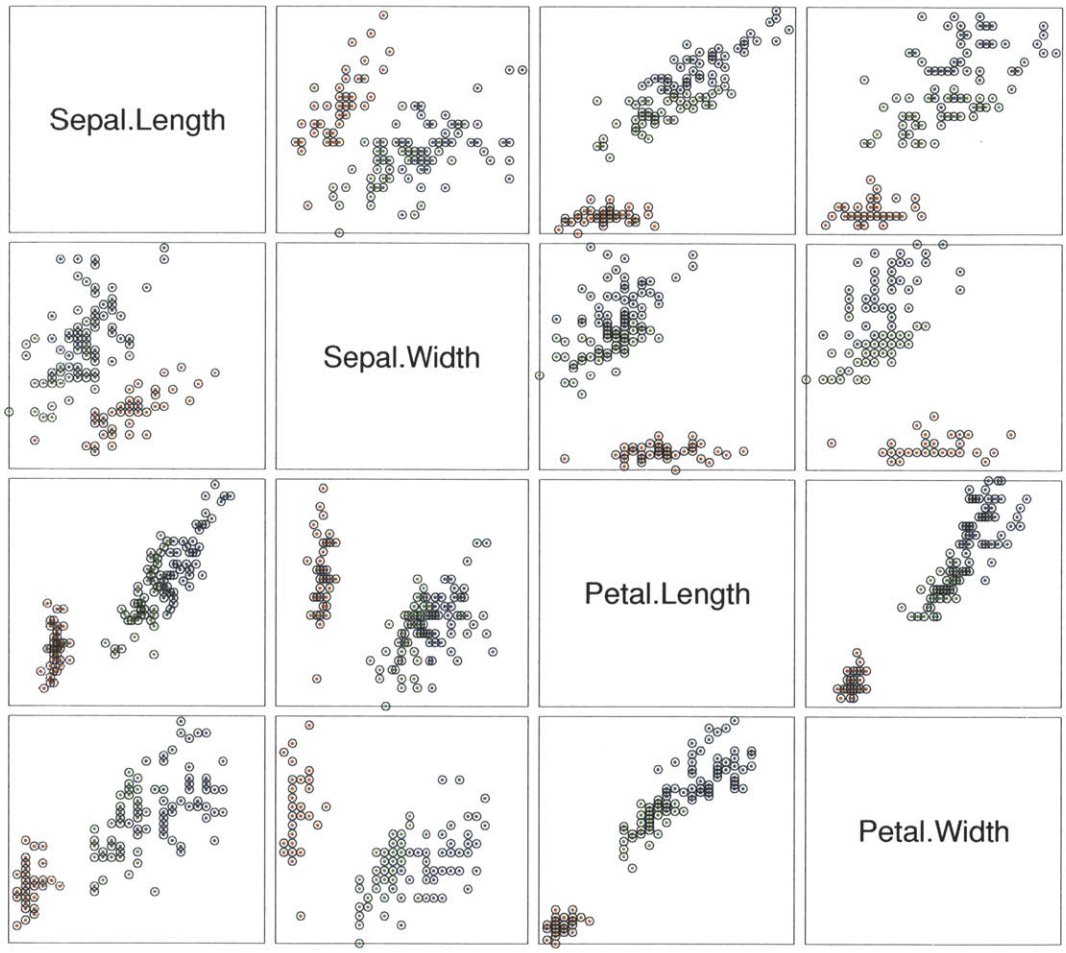

Figure 3-13: The clustering result of Iris dataset

Pen digits dataset

The Pen-based recognition of hand written digits datasets[39] (Alimoglu & Alpaydin, 1997) is a multivariate dataset of 10992 instances, each with 16 attributes. The dataset is a handwritten dataset of ten digits written by 44 writers. Consequently, the dataset is a good benchmark for testing the ability of the algorithm to cluster the data into much more than 2 clusters simultaneously in a high dimensional space. We randomly choose 1000 instances and set the parameters as: $a = 3, b = 3, \beta =$

$2, \gamma = 1.5$. The clustering results are shown in Table 3.1. The overall correctness rate is 86.6%. This level of accuracy at clustering is not yet achieved in many existing algorithms, such as Normalized cuts.

Also, we have tested the algorithm on two subsets of the dataset, PenDigits01 and PenDigits17, containing the digits "0, "1 and "1, "7, respectively. Each dataset contains 200 instances, 100 per digit. PenDigits01 is an easier dataset since the writing of "0" and "1" is more differentiable and PenDigits17 represents a more difficult dataset since the latter two digits are more difficult to differentiate in hand writing. For the set of parameters, we are using $a = 5, b = 5, \beta = 2, \gamma = 5$, and $a = 3, b = 3, \beta = 2, \gamma = 2$ respectively. The comparison results are shown in Table 3.2.

Table 3.1: Clustering result of Pendigits dataset

Cluster	Number of data	0	1	2	3	4	5	6	7	8	9	Hit rate(%)
1	84	82	0	0	0	0	0	0	0	2	0	97.6
2	81	0	76	0	0	0	0	0	5	0	0	93.8
3	31	31	0	0	0	0	0	0	0	0	0	100.0
4	81	0	3	0	0	0	0	78	0	0	0	96.3
5	143	0	25	115	0	0	0	0	2	1	0	80.4
6	107	0	0	0	0	102	0	0	1	0	4	95.3
7	63	0	0	0	0	0	0	0	63	0	0	100.0
8	134	0	8	0	90	0	9	0	3	6	18	67.2
9	60	0	0	0	0	0	7	0	0	0	53	88.3
10	11	0	0	0	0	0	0	0	11	0	0	100.0
11	61	0	0	0	0	0	61	0	0	0	0	100.0
12	18	0	0	0	0	0	0	0	0	0	18	100.0
13	7	7	0	0	0	0	0	0	0	0	0	100.0
14	21	0	0	0	0	0	0	0	0	21	0	100.0
15	20	0	0	0	0	0	0	0	0	20	0	100.0
16	14	0	0	0	0	0	1	0	0	0	13	92.8
17	25	0	0	0	0	0	0	0	0	25	0	100.0
Overall	961 (39 as outliers)											86.6

Polbooks dataset

We have also tested our algorithm on Polbooks dataset for network segmentation

tasks, although our algorithm works better on numeric clustering problems. Yet, the segmentation result is still satisfying comparing to cutting edge algorithms. For the experiment here, we are using $a = 3, b = 2, \beta = 2, \gamma = 1$. The comparison results are shown in Table 3.2.

Table 3.2: Clustering result comparison with NCut, NJW and PIC

Dataset	Instances	Clusters	NCut (%)	NJW (%)	PIC (%)	Ours (%)
Iris	150	3	67.3	80.7	98.0	97.3
Pendigits	1000	10				86.6
PenDigits01	200	2	100.0	100.0	100.0	100.0
PenDigits17	200	2	75.5	75.5	75.5	81.5
Polbooks	105	3	84.8	82.3	86.7	83.8

3.3 Novel Experiment on Application for Alleles Clustering

This section specifies the application of classifying the supertypes of the major histocompatibility complex (MHC, also called HLA in humans) alleles, especially the DRB (HLA-DR chain) alleles. We utilize the BLOSUM62 matrix and kernel matrix between alleles series in [40] to conduct data clustering. As introduced in [40], it is very important to understand the similarities of DRB for the designation of high population coverage vaccines. There are only no more than 12 HLA II alleles in each HLA gene, and an HLA gene has the ability to produce a large amount of allelic variants. It is difficult to design a vaccine that can be effective when dealing with a large population. Yet, the HLA molecules have overlapping peptide binding sets, so by grouping them into clusters, or supertypes, molecules in the same supertype will have similar peptide binding specificity. Although the Nomenclature Committee of the World Health Organization havs given extensive tables on serological type assignments to DRB alleles, the information is not yet complete. So the work of grouping the alleles into groups and compare them with the WHO labels would be

helpful to the understanding of similarities of alleles and also provide predictions to the unlabeled DRB alleles.

In the work of [40], Wen-Jun et al analyzed on 559 DRB alleles, and proposed a kernel matrix based on BLOSUM62 matrix:

$K^1 : A \times A \to R$ as

$$K^1(x,y) = (\frac{Q(x,y)}{p(x)p(y)})^\beta, \beta > 0$$

where

$$p(x) = \sum_{y \in A} Q(x,y), \forall x \in A$$

Based on this, for two amino acid strings of the same length k, $u = (u_1, u_2, ..., u_k)$, $v = (v_1, v_2, ..., v_k)$

$$K_k^2(u,v) = \prod_{i=1}^{k} K^1(u_i, v_i)$$

and further, for two amino acid with different length f and g

$$K^3(f,g) = \sum_{\substack{u \subset f, v \subset g \\ \|u\| = \|v\| = k \\ all k = 1,2,...}} K_k^2(u,v)$$

and the normalized kernel $\hat{K}$

$$\hat{K}(x,y) = \frac{K(x,y)}{\sqrt{K(x,y)K(y,y)}}$$

And finally, based on the kernel matrix, for the alleles set N with 559 components, the distance between the alleles is

$$D_{L^2}(a,b) = (\frac{1}{\|N\|} \sum_{c \in N} (\hat{K}_N^3(a,c) - \hat{K}_N^3(b,c))^2)^{\frac{1}{2}}$$

So we can utilize this distance to replace the Euclidean distance $(x_i - x_j)^2$ in the Gaussian kernel, while still use the tuning policy introduced before. Finally, we can have the clustering result shown in Table 3.3, with the parameter setting $a = 3, b = 2, \beta = 5, \gamma = 4$

Table 3.3: Clustering result of the alleles data

Cluster	Allele	Allele	Allele
1	DRB1*0107	DRB1*0109	DRB1*0129
	DRB1*0127	DRB1*0106	DRB1*0136
	DRB1*0119	DRB1*0110	DRB1*0130
	DRB1*0118	DRB1*0117	DRB1*0105
	DRB1*0131	DRB1*0112	DRB1*0108
	DRB1*0128	DRB1*0135	DRB1*0134
	DRB1*0114	DRB1*0111	DRB1*0125
	DRB1*0126	DRB1*0104	DRB1*0124
	DRB1*0137	DRB1*0121	DRB1*0120
	DRB1*0103	DRB1*0102	DRB1*0122
	DRB1*0115	DRB1*0101	
	DRB1*0123	DRB1*0132	

2	DRB1*1528	DRB1*1518	DRB1*1549
	DRB1*1514	DRB1*1546	DRB1*1508
	DRB1*1526	DRB1*1537	DRB1*1539
	DRB1*1542	DRB1*1535	DRB1*1530
	DRB1*1551	DRB1*1502	DRB1*1512
	DRB1*1524	DRB1*1501	DRB1*1529
	DRB1*1557	DRB1*1520	DRB1*1545
	DRB1*1503	DRB1*1536	DRB1*1548
	DRB1*1523	DRB1*1544	DRB1*1511
	DRB1*1554	DRB1*1531	DRB1*1507
	DRB1*1556	DRB1*1552	DRB1*1516
	DRB1*1509	DRB1*1515	DRB1*1555
	DRB1*1506	DRB1*1553	DRB1*1532
	DRB1*1547	DRB1*1533	DRB1*1558
	DRB1*1522	DRB1*1541	DRB1*1540
	DRB1*1538	DRB1*1543	DRB1*1505
	DRB1*1504		
3	DRB1*0910	DRB1*0903	DRB1*0905
	DRB1*0913	DRB1*0915	DRB1*0904
	DRB1*0911	DRB1*0916	DRB1*0912
	DRB1*0914	DRB1*0909	DRB1*0907
	DRB1*0906	DRB1*0901	DRB5*0112
	DRB1*0908		

4	DRB1*0715	DRB1*0717	DRB1*0712
	DRB1*0716	DRB1*0705	DRB1*0704
	DRB1*0709	DRB1*0707	DRB1*0720
	DRB1*0719	DRB1*0708	DRB1*0701
	DRB1*0713	DRB1*0714	DRB1*0711
	DRB1*0703	DRB1*0721	DRB1*0718
5	DRB1*1220	DRB1*1234	DRB1*1225
	DRB1*1213	DRB1*1202	DRB1*1233
	DRB1*1208	DRB1*1216	DRB1*1218
	DRB1*1215	DRB1*1219	DRB1*1211
	DRB1*1205	DRB1*1222	DRB1*1221
	DRB1*1214	DRB1*1201	DRB1*1229
	DRB1*1230	DRB1*1203	DRB1*1226
	DRB1*1228	DRB1*1207	DRB1*1232
	DRB1*1223	DRB1*1212	

	DRB1*0342	DRB1*0326	DRB1*0361
	DRB1*0345	DRB1*0337	DRB1*0328
	DRB1*0354	DRB1*0310	DRB1*0362
	DRB1*0321	DRB1*0313	DRB1*0308
	DRB1*0324	DRB1*0359	DRB1*0346
	DRB1*0307	DRB1*0358	DRB1*0312
	DRB1*0349	DRB1*0319	DRB1*0306
	DRB1*0340	DRB1*0311	DRB1*0318
	DRB1*0317	DRB1*0355	DRB1*0304
	DRB1*0341	DRB1*0320	DRB1*0344
6	DRB1*0329	DRB1*0348	DRB1*0325
	DRB1*0327	DRB1*0363	DRB1*0331
	DRB1*0302	DRB1*0305	DRB1*0309
	DRB1*0303	DRB1*0301	DRB1*0323
	DRB1*0364	DRB1*0356	DRB1*0352
	DRB1*0353	DRB1*0351	DRB1*0339
	DRB1*0360	DRB1*0314	DRB1*0332
	DRB1*0322	DRB1*0315	DRB1*0357
	DRB1*0336	DRB1*0334	DRB1*0335
	DRB1*0343	DRB1*0347	DRB1*0365
	DRB1*0330	DRB1*0333	DRB3*0115

7	DRB1*0466	DRB1*0488	DRB1*0416
	DRB1*0449	DRB1*0437	DRB1*0415
	DRB1*0445	DRB1*0458	DRB1*0461
	DRB1*04102	DRB1*0472	DRB1*0440
	DRB1*0441	DRB1*0401	DRB1*0426
	DRB1*0477	DRB1*0413	DRB1*0457
	DRB1*0446	DRB1*0444	DRB1*0480
	DRB1*0421	DRB1*0408	DRB1*0486
	DRB1*0419	DRB1*0423	DRB1*0412
	DRB1*0468	DRB1*0404	DRB1*0409
	DRB1*0406	DRB1*0470	DRB1*0467
	DRB1*0465	DRB1*0474	DRB1*0490
	DRB1*0478	DRB1*0495	DRB1*0405
	DRB1*0484	DRB1*0450	DRB1*0410
	DRB1*0491	DRB1*0427	DRB1*0487
	DRB1*0485	DRB1*0407	DRB1*0417
	DRB1*0433	DRB1*0452	DRB1*0411
	DRB1*0439	DRB1*0403	DRB1*0448
	DRB1*0435	DRB1*0451	DRB1*0424
	DRB1*0443	DRB1*0431	DRB1*0482
	DRB1*0442	DRB1*0455	DRB1*0476
	DRB1*0459	DRB1*0497	DRB1*0430
	DRB1*0428	DRB1*0475	DRB1*0489
	DRB1*0479	DRB1*04101	DRB1*0496
	DRB1*0429	DRB1*0414	DRB1*0483
	DRB1*0462	DRB1*0493	DRB1*0463

	DRB1*0471	DRB1*0402	DRB1*0460
	DRB1*0434	DRB1*0453	DRB1*0447
	DRB1*0464	DRB1*0418	DRB1*0436
	DRB1*04100	DRB1*0473	DRB1*0454
	DRB1*0456	DRB1*0438	DRB1*0425
8	DRB1*1612	DRB1*1615	DRB1*1607
	DRB1*1608	DRB1*1604	DRB1*1617
	DRB1*1609	DRB1*1605	DRB1*1611
	DRB1*1610	DRB1*1602	DRB1*1534
	DRB1*1603	DRB1*1618	DRB1*1527
	DRB1*1601	DRB1*1616	DRB1*1521
	DRB1*1614		
9	DRB1*0338	DRB3*0220	DRB3*0226
	DRB3*0102	DRB3*0203	DRB3*0206
	DRB3*0111	DRB3*0223	DRB3*0218
	DRB3*0101	DRB3*0216	DRB3*0208
	DRB3*0112	DRB3*0225	DRB3*0221
	DRB3*0109	DRB3*0219	DRB3*0207
	DRB3*0110	DRB3*0217	DRB3*0210
	DRB3*0106	DRB3*0211	DRB3*0209
	DRB3*0113	DRB3*0227	DRB3*0205
	DRB3*0108	DRB3*0214	DRB3*0213
	DRB3*0107	DRB3*0202	DRB3*0212
	DRB3*0103	DRB3*0201	DRB3*0302
	DRB3*0105	DRB3*0215	DRB3*0301
	DRB3*0114	DRB3*0222	DRB3*0303
	DRB3*0204		
10	DRB4*0104	DRB4*0101	DRB4*0107
	DRB4*0105	DRB4*0108	DRB4*0102

11	DRB5*0101	DRB5*0111	DRB5*0103
	DRB5*0104	DRB5*0113	DRB5*0204
	DRB5*0109	DRB5*0105	DRB5*0202
	DRB5*0107	DRB5*0114	DRB5*0203
	DRB5*0106	DRB5*0102	DRB5*0205
12	DRB1*0821	DRB1*0802	DRB1*0810
	DRB1*0814	DRB1*0804	DRB1*0848
	DRB1*0826	DRB1*0830	DRB1*0825
	DRB1*0809	DRB1*0813	DRB1*0834
	DRB1*0835	DRB1*0841	DRB1*0807
	DRB1*0836	DRB1*0819	DRB1*0833
	DRB1*0838	DRB1*0847	DRB1*0823
	DRB1*0816	DRB1*0805	DRB1*0827
	DRB1*0842	DRB1*0822	DRB1*0815
	DRB1*0828	DRB1*0839	DRB1*0824
	DRB1*0845	DRB1*0801	DRB1*0820
	DRB1*0817	DRB1*0806	DRB1*1415
	DRB1*0837	DRB1*0843	DRB1*14116
	DRB1*0844	DRB1*0840	DRB1*1477
	DRB1*0808	DRB1*0818	DRB1*1440
	DRB1*0811	DRB1*0803	DRB1*14102
	DRB1*0829	DRB1*0846	DRB1*1484
	DRB1*0812		

	DRB1*1446	DRB1*1455	DRB1*1406
	DRB1*1410	DRB1*14104	DRB1*1451
	DRB1*1457	DRB1*14101	DRB1*14106
	DRB1*1439	DRB1*1434	DRB1*1489
	DRB1*1461	DRB1*1408	DRB1*1481
	DRB1*1473	DRB1*1414	DRB1*1418
	DRB1*1479	DRB1*1423	DRB1*1413
	DRB1*14107	DRB1*1462	DRB1*1494
	DRB1*1431	DRB1*1460	DRB1*1483
	DRB1*1493	DRB1*1436	DRB1*1482
	DRB1*1450	DRB1*1464	DRB1*1495
	DRB1*1428	DRB1*1421	DRB1*1496
	DRB1*1468	DRB1*1430	DRB1*1437
13	DRB1*1471	DRB1*1417	DRB1*1445
	DRB1*1404	DRB1*1433	DRB1*14105
	DRB1*1476	DRB1*1459	DRB1*1416
	DRB1*1475	DRB1*1409	DRB1*1474
	DRB1*1411	DRB1*1480	DRB1*14111
	DRB1*1452	DRB1*1497	DRB1*14100
	DRB1*14117	DRB1*1422	DRB1*1443
	DRB1*14112	DRB1*1441	DRB1*1444
	DRB1*1426	DRB1*1449	DRB1*1405
	DRB1*1470	DRB1*1420	DRB1*14103
	DRB1*14110	DRB1*14109	DRB1*1456
	DRB1*1435	DRB1*14108	DRB1*1491
	DRB1*1472	DRB1*1424	DRB1*1432
	DRB1*1465	DRB1*1419	DRB1*1487

	DRB1*1438	DRB1*1486	DRB1*1447
	DRB1*1499	DRB1*1490	DRB1*1429
	DRB1*1407	DRB1*1488	DRB1*1402
	DRB1*1401		
	DRB1*14115	DRB1*1498	DRB1*1463
14	DRB1*1427	DRB1*1403	DRB1*1478
	DRB1*1467	DRB1*1412	DRB1*1485
	DRB1*0113	DRB1*0420	DRB1*1209
	DRB1*0116	DRB1*0499	DRB1*1204
	DRB1*1003	DRB1*0469	DRB1*0832
	DRB1*1001	DRB1*0498	DRB1*1442
Outliers	DRB1*1002	DRB1*0482	DRB1*1425
	DRB1*1510	DRB1*0706	DRB1*1469
	DRB1*1525	DRB1*0902	DRB1*1458
	DRB1*0316	DRB5*0112	DRB1*1448
	DRB1*1227		

As we can see by comparing our result to the result in [40], we can have a match table as Table 3.4. Also, detailed comparison with results in each supertype is shown in Table 3.5, we can see that our result has no misclassification errors if we treat their results as correct. All the difference between our result and theirs, is that the 25 alleles classified as outliers in our result have been put into some clusters in their result. However, as they also suggest that some outliers in our result such as DRB5*0112, DRB1*1525, DRB1*1425, DRB1*1442, DRB1*1469 and DRB1*0832 are exceptional, the combination of both results make the classification of these alleles doubtful. We also share same results on the clustering of several other exceptions, such as DRB1*0338, DRB3*0115. Moreover, we support their results on classification of , DRB1*1440 , DRB1*1469, DRB1*1477, DRB1*1484, DRB1*14116 and DRB1*14102 into the ST8 supertype. The experiment on this problem suggests that our algorithm is effective on clustering multiple clusters simultaneously for alleles

clustering data, and also our results support the conclusions of Wen-jun et al's work on the mathematical foundation analysis of amino acid chains. The ability of detecting outliers may lead further analysis on the controversial results in the clustering and provide potential directions on biological researches.

Table 3.4: Clustering result match-up of alleles clustering

Supertype	Serotype	Cluster
ST52	DR52	9
ST3	DR3, DR17, DR18	6
ST6	DR14, DR1404, DR1403, DR6	13, 14
ST8	DR8	12
ST4	DR4	7
ST2	DR15, DR16, DR2	2, 8
ST5	DR12	5
ST53	DR53	10
ST9	DR9	3
ST7	DR7	4
ST51	DR51	11
ST1	DR1, DR103	1
ST10	DR10	DRB1*1003 (outlier) DRB1*1001(outlier) DRB1*1002(outlier)

3.4 Experiments on Dynamic System Grouping

In this section, we will introduce the applications of our algorithm on clustering dynamic systems or dynamic data. The applications in this part show the capability of our algorithm to deal with time varying data in a continuous way. The clustering result that our algorithm can achieve is flexible and changing according to the distribution of data, which can be regarded as an integration of distribution over a certain period of time. We will introduce the applications of mobile robots clustering, adaptive system self grouping and potential multi-model system embedding in the following sections.

Table 3.5: Clustering result comparison of alleles clustering

Supertype	Not included	Misclassified	Outliers
ST52	0	0	0
ST3	6	0	6
ST6	2	0	2
ST8	2	0	2
ST4	6	0	6
ST2	1	0	1
ST5	1	0	1
ST53	0	0	0
ST9	1	0	1
ST7	1	0	1
ST51	0	0	0
ST1	2	0	2
ST10	3	0	3
Overall	25	0	25

3.4.1 Motivation

It is fascinating to see the occasion that tens of thousands of fish, birds, insects or buffalos moving together with perfect coordination and flexible formation. There are no absolute leaders or regional coordinators in the herd, yet the members coordinate and interact with local neighbors, that eventually realizes global coordination. There are already researchers working on the projects of swarm intelligence and swarm robotics, such as Vijay Kumar's group in University of Pennsylvania and The Kilobot Project from Radhika Nagpal's group in Harvard University. However, current researches mostly focus on the grouping behavior of tens of robots. When considering communications among thousands of robot agents, we need to design a novel mechanism for agents to communicate and interact within neighborhoods efficiently and quickly. Using bio-inspired strategy on the study of this problem is highly attractive, since we have already seen the fact that flocks of birds and schools of fish can move elegantly with flexibility and ability to avoid obstacles. It would be significantly meaningful, if we can combine research on clustering, synchronization and control strategy to achieve real-time learning and decision making. So we test our algorithm inspired by quorum sensing on three applications related to dynamic systems to show the potential and real advantage of our algorithm comparing to other existing ones, the ability

to make real machines more intelligent.

3.4.2 Application I. Clustering of mobile robots

The first application is to cluster working mobile robots into groups so that in further tasks, they can have better coordination and synchronization. We design the scenario as 200 robots distributed at the locations of the previous introduced two-moon dataset. The 200 robots are moving around their center locations with the radius of 0.5, random angular speed in the range of $[2, 4]$, and random initial angular position in the range of $[0, 2\pi]$. The whole algorithm remains the same as in the applications in previous applications. It is only the distance matrix D and influence matrix M, that are updated continually. Although it seems to be a lot of calculation, actually, in the real world application, we can attach electromagnetic field emitters, and electromagnetic field intensity sensors on each of the robot agent. The intensity of electromagnetic field at a certain location is the superposition of all neighboring electromagnetic fields influence. In this way, by tuning the influence radius of electromagnetic fields, which is most likely to be Gaussian distributions, we can change the communications mechanism from agent-to-agent interactions to agent-environment-agent interactions, which is the basic idea of quorum sensing.

Back to our experiment, in Fig.3-14, it is shown that even if the locations of robots are changing over time, our influence radius tuning process is capable of dealing with local density variations. Over the simulation, the density distribution map fits closely with the data distribution which is helpful for segmenting clusters and detecting outliers. Also in Fig.3-15, we can see that the influence radius is tuning down regarding to a local high density, and tuning up corresponding to a local low density, to achieve balance. Finally, in Fig.3-16, we can see that the cluster number is merged all the way down to 2 clusters, and the final result is exactly what we want, same as in the static two-moon model clustering.

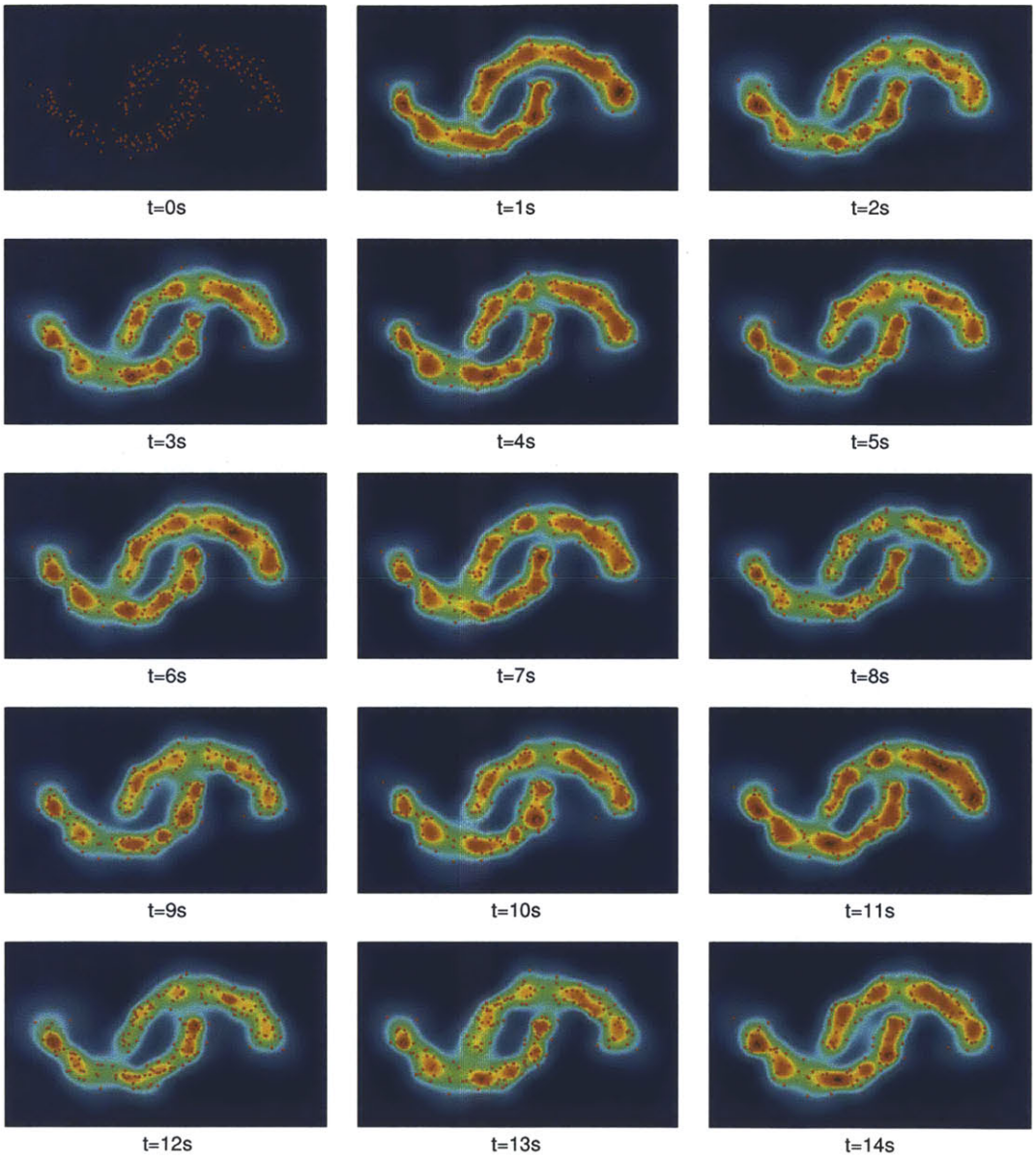

Figure 3-14: The distribution map of data and density in 14 seconds

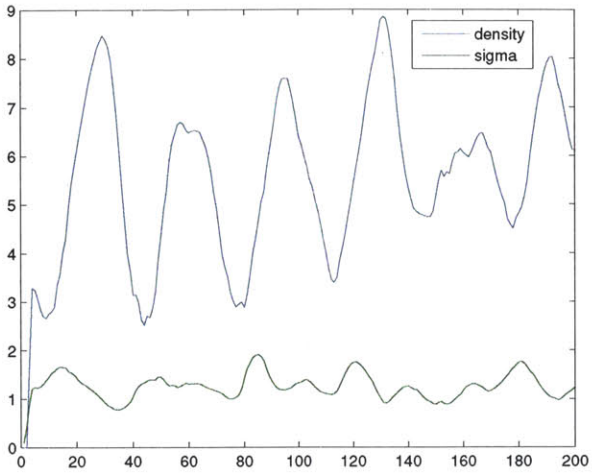

Figure 3-15: Variation of density and influence radius of a single cell

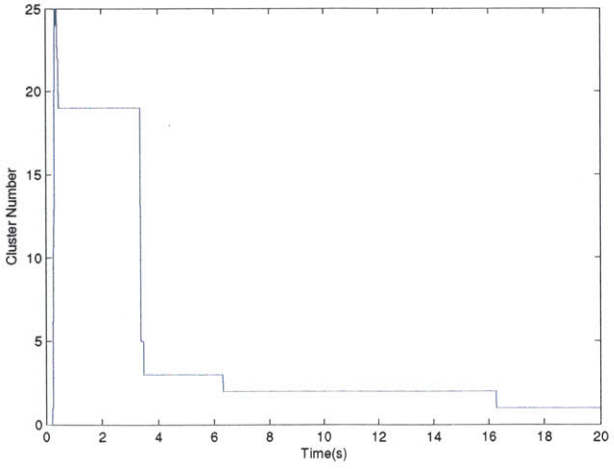

Figure 3-16: Cluster numbers of over the simulation time

3.4.3 Application II. Clustering of adaptive systems

One problem on adaptive control is that the adapting parameters cannot be guaranteed to converge to the true value. However, it is important to know the true sets of parameters to enhance transient response and improve further performance in the long run. Suppose we have many unknown dynamic systems to control, while we have the information that the parameters configurations can only fall into certain limited choices. So it would be possible to cluster the dynamic systems and figure out who share the same parameters configuration as one group. Further, we can even couple the systems in the same group, to achieve synchronization by forming contracting Lagrangian systems as introduced in [41] and [42]. As explained in [43], synchronization will enhance the ability to resist noise and improve robustness of the whole network. Consequently, such grouping and coupling would be beneficial for better control performance, and can be analyzed in further research.

Also, as we see in many natural colonies, the group merging and splitting is so smooth and elegant, which gives the colonies flexibility to deal with obstacles and danger. So we want to also realize this dynamic clustering in a natural way by merging similar clusters and split clusters that are already not connected. We design our experiment as follow.

Suppose we have 60 dynamic systems as $m_i\ddot{x}_i + b_i\dot{x}_i + k_i x_i = u_i, m_i, b_i, k_i$ are unknown constants. The 60 dynamic systems are in mainly 3 groups. The parameters m_i, b_i, k_i of the first 20 systems follows the Gaussian distribution around $[2, 4, 3]$ with the standard variation as 0.2. The parameters m_i, b_i, k_i of the 20-40 systems follows the Gaussian distribution around $[4, 3, 2]$ with the standard variation as 0.2. And the parameters of the last 20 systems follows the Gaussian distribution around $[3, 2, 4]$ with the standard variation as 0.2. The initial distribution of the parameter space is shown in Fig.3-17. During the simulation time between 25s and 35s after the systems is stable, the parameters of the last 20 systems are changed to follow the Gaussian distribution around $[4, 3, 2]$ with the standard variation as 0.2, so that there are actually 2 clusters during that time. And after 35s, the parameters of the last

20 systems are changed to follow the Gaussian distribution around $[6, 6, 6]$ with the standard variation as 0.2, when there will be again 3 clusters existing then. We believe such experiments would be adequate to test the dynamic clustering by merging and splitting colonies.

The goal of controlling the system would be to track the trajectory of $x_d(t) = sin(2\pi t)$, so obviously $\dot{x}_d(t) = 2\pi cos(2\pi t), \ddot{x}_d(t) = -4\pi^2 sin(2\pi t)$. Generally to make sure a system like $m\ddot{x} + b\dot{x} + kx = u$ follows the predefined trajectory, we can design adaptive control strategy as follows: Define $s = \dot{x} - \dot{x}_d + \lambda(x - x_d)$
and the Lyapunov function as

$$V = 0.5ms^2 + 0.5(\hat{m} - m)^2 + 0.5(\hat{b} - b)^2 + 0.5(\hat{k} - k)^2 \geq 0 \tag{3.1}$$

then

$$\begin{aligned} \dot{V} &= ms\dot{s} + \dot{\hat{m}} * \tilde{m} + \dot{\hat{b}} * \tilde{b} + \dot{\hat{k}}\tilde{k} \\ &= \dot{\hat{m}}\tilde{m} + \dot{\hat{b}} * \tilde{b} + \dot{\hat{k}}\tilde{k} + ms(\ddot{x} - \ddot{x}_d + \lambda(\dot{x} - \dot{x}_d)) \\ &= \dot{\hat{m}}\tilde{m} + \dot{\hat{b}} * \tilde{b} + \dot{\hat{k}} * \tilde{k} + s(u - m\ddot{x}_d - b\dot{x} - kx + \lambda m(\dot{x} - \dot{x}_d)) \end{aligned}$$

So by designing

$$u = \hat{m}(\ddot{x}_d - \lambda(\dot{x} - \dot{x}_d)) + \hat{b}\dot{x} + \hat{k}x - k_1 s \tag{3.2}$$

where k_1 is a positive number, and

$$\dot{\hat{m}} = -s(\ddot{x}_d - 4(\dot{x} - \dot{x}_d)) \tag{3.3}$$

$$\dot{\hat{b}} = -s\dot{x} \tag{3.4}$$

$$\dot{\hat{k}} = -sx \tag{3.5}$$

We can get $\dot{V} = -k_1 s^2 \leq 0$. Also since $\ddot{V}$ is bounded, by using Barbalat's lemma, we

can have $\dot{V}$ converging to 0 asymptotically. And thus s and $x - x_d$ will converge to 0 asymptotically since s can be considered as a first order filter for $x - x_d$. So this it the rule we use to control the system by using

$$u = \hat{m}(\ddot{x}_d - \lambda(\dot{x} - \dot{x}_d)) + \hat{b}\dot{x} + \hat{k}x - k_1 s$$

Also, we assume that systems with similar parameter configurations would need similar inputs since we know that

$$\tilde{m}(\ddot{x}_d - \lambda(\dot{x} - \dot{x}_d)) + \tilde{b}\dot{x} + \tilde{k}x$$

will converge to 0 asymptotically in adaptive control theory and thus

$$u \approx m(\ddot{x}_d - \lambda(\dot{x} - \dot{x}_d)) + bdx + kx$$

. Consequently, systems with similar parameter settings are likely to get similar inputs. By conducting fast Fourier transform of the input signal over 1 second which is the period time of the sinusoidal trajectory, we can use the Fourier transform vector to calculate distance matrix between agents and then put into our algorithm for clustering the systems. In our experiment, we have $\lambda = 4$, and we use the normalized 20 point Fourier transform vector as the feature vector. The final clustering result is shown in Fig.3-18. As we can see, during 0-25 seconds, the cluster number is merged down to 3, and further to 2 approximately only 1 second after the parameters configuration is changed. And moreover, a third cluster is split out from the merged 40 systems again only 1 second after the parameters configuration variation. With the correct clustering result of the 60 systems during all parameter variations, we are confident to say that our system is capable of dealing with time varying data by using the accumulating information to do dynamic clustering, while also capable of handling variations of cluster numbers as well. The results also suggest that using our algorithm to group dynamic system is feasible and further applications on synchronization and coordination are very promising.

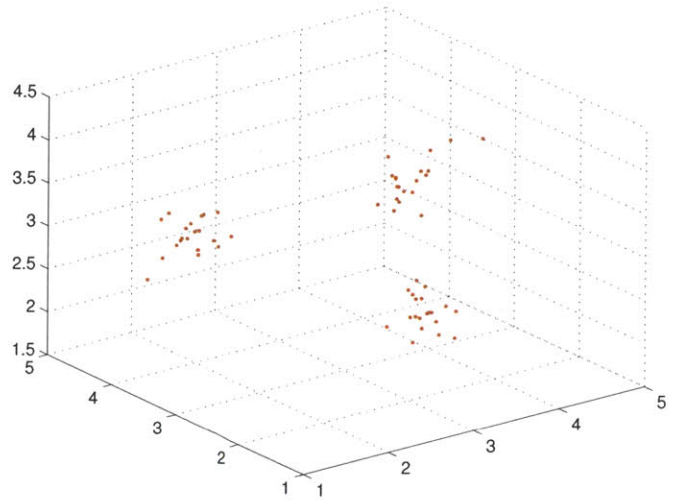

Figure 3-17: Initial parameters configuration of the 60 systems

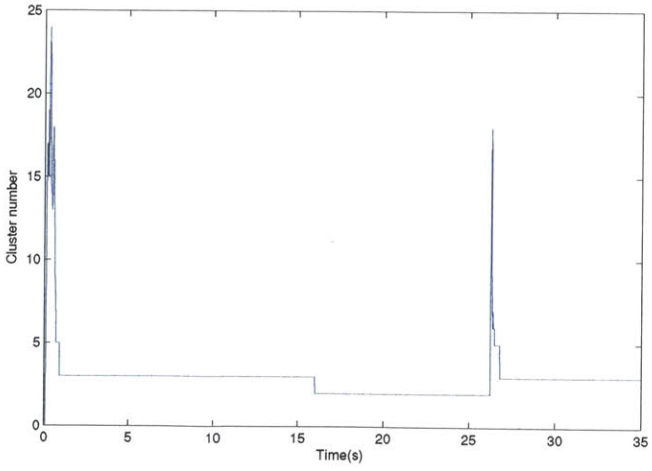

Figure 3-18: Cluster numbers during the simulation

3.4.4 Application III. Multi-model switching

As introduced in [7][8], multi-model switching control could help the system with better transient control performance and higher precision. Suppose we have a system with unknown parameters settings, however we know that there are only three choices of parameters configuration. We are suggesting a new method for multi-model switching control as follows:

1. Initially, we use adaptive control to make sure the system is working with accpetable performance. While at the same time, we have tens of virtual systems simulating with the same control input and parameters scattering around the pre-known choices.

2. After the density map is stable by tuning the influence radius of the virtual systems, we can get the local density of the real system as

$$d_r = \sum_i^n e^{-\frac{(f_r - f_i)^2}{\sigma_i^2}} \tag{3.6}$$

   where f_r and f_i are Fourier transform vectors of the input of the real system and virtual systems.

3. If d_r exceed a predefined threshold, we can consider that the real system belongs to a virtual cluster, and by analyzing the virtual cluster's parameters settings, we will be able to realize the parameters choice of the real system.

4. Further, if the parameters of the system vary again, by detecting d_r dropping down, we can resume adaptive control and wait for the next time that d_r supasses the threshold.

In our experiment, we are still using the same 60 dynamic systems and control laws as introduced in the previous section. And we have a new "real" system, whose parameters $m_r, b_r, k_r$are set as [4,3,2] initially, and then changed to [2,4,3]. When the system is recognized by a cluster, we will set the estimation parameters to the true value of the cluster belonging parameters. As we can see from the simulation results

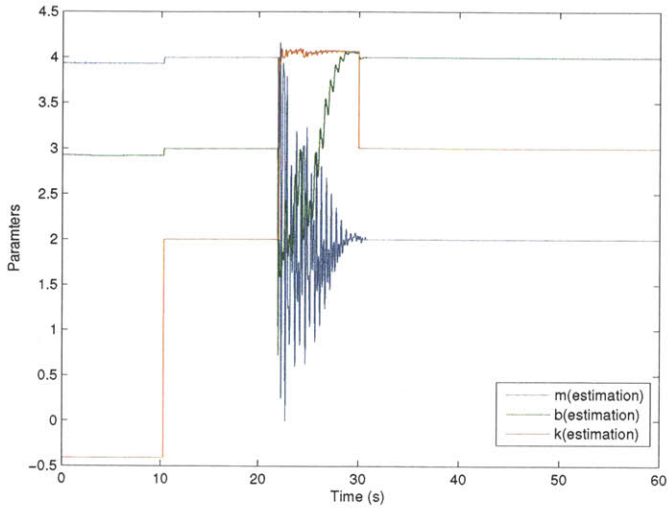

Figure 3-19: Parameter estimations of the real system

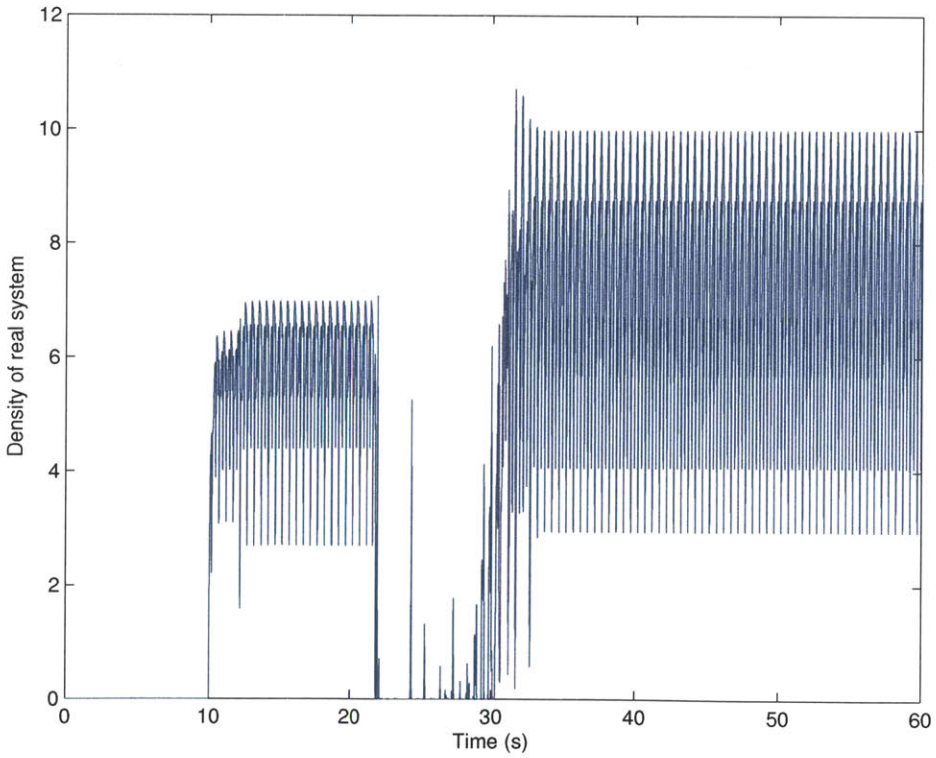

Figure 3-20: Density of the real system

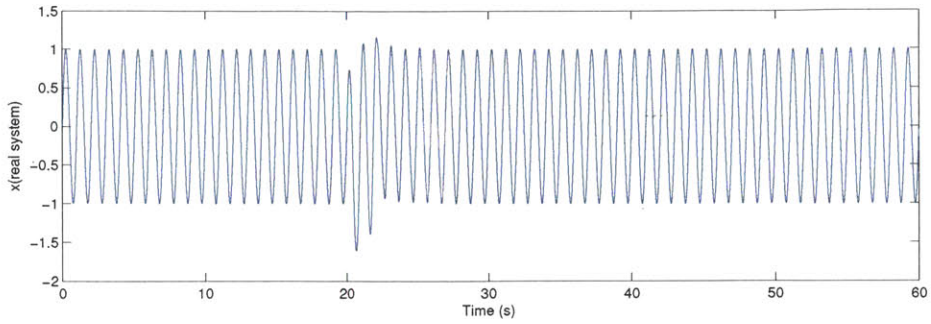

Figure 3-21: Trajectory of the real system

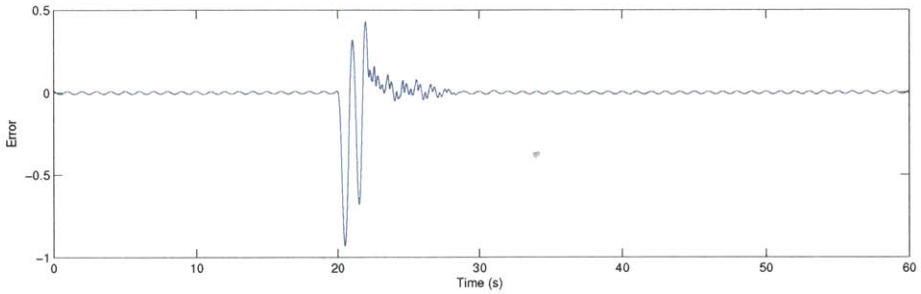

Figure 3-22: Error of the real system

Fig.3-19 and Fig.3-20, soon after the multi-model switching starts at $t = 10$ seconds, the density of the real system surpasses the threshold 5, and the parameter estimations is tuned to the true value. The real system parameters are changed at $t = 20$ seconds, and in a short period of time the density drops along with the control mode changed to adaptive control. After about another 10 seconds, the density is high again, and the system is recognized by the new correct cluster with true parameters set. Also, as we can see from Fig.3-21 and Fig.3-22, the system quickly adopts adaptive control when the robust control with parameter prediction fails and the over all performance is quite good during the transition.

With the three applications above, we show the ability of combining our algorithm with dynamic systems in order to give mechanical control "real intelligence". The overall strategy mimics the idea of smooth variations in natural colonies and the results prove the reliability of the proposed algorithm.

Chapter 4

Conclusions

4.1 Summary

As we introduced at the beginning the whole thesis, our goal of this thesis is to present our clustering algorithm inspired by quorum sensing as a potential bridge between modern machine learning technologies and adaptive estimation and control. With the background of huge improvements in calculation power, it is now reasonable to take another look at the adaptive control theory, whose main body was developed around 1980s. However, most current machine learning algorithms are not suitable to be used on "machines", since these algorithms take mathematical methods to realize optimizations, which is difficult to be combined with dynamic systems in real-time use. So we choose the clustering problem as a breakthrough point, attempting to bridge the two parts. When considering online clustering problems, it reminds us that many natural colonies such as schools of fish, flocks of birds, and hordes of buffalos have great coordination and synchronizations with in the colony. It is natural to search answers and solutions from inspirations in nature. The natural phenomenon that inspires us is quorum sensing, which is a decentralized process among colonies of cells. In quorum sensing, cells do not communicate with each other directly, yet they influence and sense from the local environment, which assures simple local decision making policy and global optimization simultaneously.

Inspired by quorum sensing, we design a clustering algorithm mainly consisted

of two parts. The first part is an influence radius tuning process which helps to build and stabilize a sparsely connected networks. When density of some certain cells surpasses a predefined threshold, we can establish new colonies originating from the corresponding "core cell". And further, the colonies interact with each other following a rule that tends to optimize a cost function similar with the one used in Normalized Cuts. Finally when the whole system becomes stable, we can extract clustering results by counting the colony vectors, which is rather simple.

In the experiments, we first test our designed algorithm on some synthetic datasets, such as two-moon model, two-spiral model, two-chain model and the island model. The results show that our algorithm performs nicely on both linearly separable datasets and non-linearly separable datasets, which is not available for K-means clustering and Distribution-based clustering methods. Then we test the algorithms on some real datasets like Iris dataset, Pendigits dataset and Polbooks dataset. Our results show that the proposed algorithm is capable of clustering multiple clusters simultaneously and the performance is no worse or even better than cutting-edge algorithms on these static datasets. When using our algorithm on some novel applications such as alleles clustering, we achieve highly similar results comparing to the results of Smale's group. We not only get most alleles labeled the same as theirs, but also support their conclusions on some exceptional classification. Moreover, we detect several outliers that are also exceptional samples in their result, which could lead further biological research to emphasize on these specific alleles. Finally, we put our algorithm into the applications of mobile robots clustering, adaptive systems clustering and multi-model switching control. The performances have shown that our algorithm is capable of dealing with data variation and cluster merging and splitting efficiently, just like the smooth and elegant group coordination in nature. Our algorithm realizes dynamic clustering and can be easily incorporated into control strategy, which is a breakthrough in bridging adaptive estimation and control with modern machine learning techniques.

Our algorithm's advantage over existing algorithms can be concluded as the following points:

1. Our algorithms defines cluster as a local region where density is high and continually distributed. Thus our algorithm is capable of clustering datasets that are not linearly separable.

2. By mimicking quorum sensing, we make our algorithm decentralized, so that it is suitable for parallel and distributed computation, which may further improve algorithm speed. Especially if we use the algorithm on real mobile robots with individual onboard processors, the local decision making policy would be really easy since the agents are only interacting with their local environments.

3. Since the whole system is decentralized, our algorithm handles noise and outliers well, and can detect outliers easily by choosing the cells with their density close to zero.

4. We can cluster multiple colonies simultaneous during the process. And such segmentation is dynamically adjusted due to the optimization of a reasonable cost function.

5. Our algorithm can adapt to clusters with different sizes and variations, since the influence radius is tuned to preserve local connectivity.

6. Though cluster merging and splitting, our algorithm can even adapt to cluster variations during any period of time with quick response time.

7. The biggest advantage of our algorithm is that the clustering process itself is dynamically evolved, which makes the algorithm easy to be conbined with real mechanical systems with no modifications. Such combination could shed light upon new control theory development and new ideas that could make the controlled systems more flexible, coordinated and robust.

Also inevitably, our algorithm has several drawbacks. First, the overall computation complexity would be $O(n)^3$ if all the calculations are undertaken by one single processor. However, this is due to the reason that the advantage of quorum sensing as a decentralized process hasn't been utilized. Actually, if we use the algorithm

in real robots clustering scenarios, with distributed computation, the computation complexity of any single robot would be hugely reduced to linear time. Secondly, the algorithm tends to be sensitive to the parameter settings, and cluster segmentation through colony interaction could be trapped in local optimum. Through the analysis in Chapter 2 for colony interaction, we know that γ defines the ability of mutual penetration and crossing density gaps. So with larger γ, coloniesare more inclined to merge into each other. The parameter β measures how much we want the σ_i's in a cluster to be similar with each other. Larger β would result in smoother σ_i's distribution, yet potentially rugged density distribution, since the cells may not be so homogeneously distributed. Parameters a and b measure how sparse do we want for the connection graphs. With more connected graphs, we tend to have fewer clusters, and vice versa. Hence, we have some basic rules to tune the parameters: if the result suggests fewer clusters than we expect, we can tune down a, and b, and tune up γ, and if the influence radius of some cells become too large, we can tune up β. It is also worth noting that, even though the system is somehow sensitive to parameter settings, all the possible clustering results make sense in different views, for example, the two-moon dataset can be segmented into two, three or four clusters, that all seem plausible. Further, we can design some rules tuning parameters dynamically according to our expectations, which can be part of the future work.

4.2 Future Work

For future work, we will develop the algorithm in several directions. First, we will conduct further research on stabilizing the algorithm and reduce sensitivity of the parameters. Secondly, for describing dynamic systems, we may need various better methods for extracting feature vectors rather than just analyzing the fast Fourier transform vector of the inputs. Thirdly, we will look into the possibilities of using our clustering algorithm on much more application scenarios involved with dynamic systems. And last but not least, we will develop new control theories to utilize the information that such modern machine learning techniques could further improve

control performance, synchronization and more natural self-organizing behaviors.

Bibliography

[1] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[2] J.A. Hartigan and M.A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[3] A.Y. Ng, M.I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002.

[4] Shi Jianbo. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:888–905, 2000.

[5] J. J. E. Slotine and Weiping Li. *Applied nonlinear control.* Prentice Hall, 1991.

[6] R.M. Sanner and J.J.E. Slotine. Gaussian networks for direct adaptive control. *IEEE Transactions on Neural Networks*, 3(6):837–863, 1992.

[7] K.S. Narendra and J. Balakrishnan. Adaptive control using multiple models. *IEEE Transactions on Automatic Control*, 42(2):171–187, 1997.

[8] K.S. Narendra, J. Balakrishnan, and M.K. Ciliz. Adaptation and learning using multiple models, switching, and tuning. *Control Systems, IEEE*, 15(3):37–51, 1995.

[9] D.M. Wolpert and M. Kawato. Multiple paired forward and inverse models for motor control. *Neural Networks*, 11(7-8):1317–1329, 1998.

[10] A. Paul and M.G. Safonov. Model reference adaptive control using multiple controllers and switching. In *Proceedings of 42nd IEEE Conference on Decision and Control*, volume 4, pages 3256–3261 vol. 4, 2003.

[11] L. Li, M. Gariel, R.J. Hansman, and R. Palacios. Anomaly detection in onboard-recorded flight data using cluster analysis. In *Digital Avionics Systems Conference*, pages 4A4–1–4A4–11. IEEE, 2011.

[12] R. Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.

[13] S. Guha, R. Rastogi, and K. Shim. Cure: an efficient clustering algorithm for large databases. volume 27, pages 73–84. ACM SIGMOD Record, 1998.

[14] S. Guha, R. Rastogi, and K. Shim. Rock: A robust clustering algorithm for categorical attributes. In *15th International Conference on Data Engineering*, pages 512–521. IEEE, 1999.

[15] G. Karypis, E.H. Han, and V. Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *Computer*, 32(8):68–75, 1999.

[16] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases, 1996.

[17] M. Ester, H.P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. volume 1996, pages 226–231. AAAI Press, 1996.

[18] Y. Afek, N. Alon, O. Barad, E. Hornstein, N. Barkai, and Z. Bar-Joseph. A biological solution to a fundamental distributed computing problem. *Science's STKE*, 331(6014):183, 2011.

[19] C.M. Bishop. Neural networks for pattern recognition. 1995.

[20] J.J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554, 1982.

[21] David E. Goldberg and John H. Holland. Genetic algorithms and machine learning. *Machine Learning*, 3(2):95–99, 1988.

[22] M. Waibel, D. Floreano, and L. Keller. A quantitative test of hamilton's rule for the evolution of altruism. *PLoS Biology*, 9(5):e1000615, 2011.

[23] M. Dorigo and G. Di Caro. Ant colony optimization: a new meta-heuristic. In *Proceedings of the Congress on Evolutionary Computation*, volume 2. IEEE, 1999.

[24] D. Karaboga and B. Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *Journal of Global Optimization*, 39(3):459–471, 2007.

[25] X.S. Yang. *Nature-inspired metaheuristic algorithms*. Luniver Press, 2011.

[26] K.E. Parsopoulos and M.N. Vrahatis. Recent approaches to global optimization problems through particle swarm optimization. *Natural computing*, 1(2):235–306, 2002.

[27] A.H. Gandomi and A.H. Alavi. Krill herd: a new bio-inspired optimization algorithm. *Communications in Nonlinear Science and Numerical Simulation*, 2012.

[28] M.B. Miller and B.L. Bassler. Quorum sensing in bacteria. *Annual Reviews in Microbiology*, 55(1):165–199, 2001.

[29] C.M. Waters and B.L. Bassler. Quorum sensing: cell-to-cell communication in bacteria. *Annual Review of Cell and Developmental Biology*, 21:319–346, 2005.

[30] T.D. Seeley and P.K. Visscher. Group decision making in nest-site selection by honey bees. *Apidologie*, 35(2):101–116, 2004.

[31] S.C. Pratt. Quorum sensing by encounter rates in the ant temnothorax albipennis. *Behavioral Ecology*, 16(2):488–496, 2005.

[32] K.H. Nealson, T. Platt, and J.W. Hastings. Cellular control of the synthesis and activity of the bacterial luminescent system. *Journal of bacteriology*, 104(1):313–322, 1970.

[33] E. Mallon, S. Pratt, and N. Franks. Individual and collective decision-making during nest site selection by the ant leptothorax albipennis. *Behavioral Ecology and Sociobiology*, 50(4):352–359, 2001.

[34] K.G. Chan, S.D. Puthucheary, X.Y. Chan, W.F. Yin, C.S. Wong, W.S.S. Too, and K.H. Chua. Quorum sensing in aeromonas species isolated from patients in malaysia. *Current microbiology*, 62(1):167–172, 2011.

[35] B.M.M. Ahmer. Cell-to-cell signalling in escherichia coli and salmonella enterica. *Molecular microbiology*, 52(4):933–945, 2004.

[36] Alexander Hinneburg and Hans-Henning Gabriel. *DENCLUE 2.0: Fast Clustering Based on Kernel Density Estimation*, volume 4723, pages 70–80. Springer Berlin / Heidelberg, 2007.

[37] F. Lin and W.W. Cohen. Power iteration clustering. *ICML (to appear)*, 2010.

[38] R.A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Human Genetics*, 7(2):179–188, 1936.

[39] F. Alimoglu and E. Alpaydin. Combining multiple representations and classifiers for pen-based handwritten digit recognition. In *Proceedings of the Fourth International Conference on Document Analysis and Recognition*, volume 2, pages 637–640 vol. 2. IEEE, 1997.

[40] W.J. Shen, H.S. Wong, Q.W. Xiao, X. Guo, and S. Smale. Towards a mathematical foundation of immunology and amino acid chains. *Arxiv preprint arXiv:1205.6031*, 2012.

[41] Quang-Cuong Pham and Jean-Jacques Slotine. Stable concurrent synchronization in dynamic system networks. *Neural Networks*, 20(1):62–77, 2007.

[42] S.J. Chung and J.J.E. Slotine. Cooperative robot control and concurrent synchronization of lagrangian systems. *IEEE Transactions on Robotics*, 25(3):686–700, 2009.

[43] N. Tabareau, J.J. Slotine, and Q.C. Pham. How synchronization protects from noise. *PLoS Computational Biology*, 6(1):e1000637, 2010.