ISSN: 2150-8194 Vol. 2, No. 6, June 2010

Scientific Research

Intelligent Information Management

ISSN: 2150-8194

9 772150 819009 06

www.scirp.org/journal/iim

Journal Editorial Board

ISSN: 2150-8194 (Print), 2150-8208 (Online)

Hhttp://www.scirp.org/journal/iim

Intelligent Information Management, **2010, 2, 343-390**
Published Online June 2010 in SciRes (http://www.SciRP.org/journal/iim/)

TABLE OF CONTENTS

Volume 2 Number 6 June 2010

Intelligent Information Management (IIM)

Journal Information

SUBSCRIPTIONS

The *Intelligent Information Management* (Online at Scientific Research Publishing, www.SciRP.org) is published monthly by Scientific Research Publishing, Inc., USA.

Subscription rates:

Print: $50 per issue.
To subscribe, please contact Journals Subscriptions Department, E-mail: sub@scirp.org

SERVICES

Advertisements
Advertisement Sales Department, E-mail: service@scirp.org

Reprints (minimum quantity 100 copies)
Reprints Co-ordinator, Scientific Research Publishing, Inc., USA.
E-mail: sub@scirp.org

COPYRIGHT

PRODUCTION INFORMATION

For manuscripts that have been accepted for publication, please contact:
E-mail: iim@scirp.org

Intelligent Information Management, **2010, 2, 343-353**
doi:10.4236/iim.2010.26042 Published Online June 2010 (http://www.SciRP.org/journal/iim)

Status of Developers' Testing Process

Gudrun Jeppesen[1], Mira Kajko-Mattsson[2], Jason Murphy[3]
[1]*Department of Computer and Systems Sciences, Stockholm University, Stockholm, Sweden*
[2]*School of Information and Communication Technology, Royal Institute of Technology, Stockholm, Sweden*
[3]*Nomadic Software, Nomadic City, Sweden*
E-mail: *gudrun@dsv.su.se, mekm2@kth.se, jasonleemurphy@hotmail.com*
Received March 12, 2010*; revised April* 15, 2010; *accepted May* 17, 2010

Abstract

Even if recent methodologies bring more recognition to developers' testing process, we still have little insight into its status within the industry. In this paper, we study the status of developers' testing process at Nomadic Software. Our results show that the process is not uniformly executed. The company suffers from lack of control over the methods used, lack of formal communication on requirements, lack of static testing practice, and lack of testing process documentation.

Keywords: Dynamic Testing, Static Testing, Peer Reviews, Inspections, Debugging, Test Cases, Testing Techniques

1. Introduction

Despite its importance, the overall testing process has for many years been neglected both within research and industry [1-3]. Most of the effort has been spent on creating testing processes on the system level. Hence, we have fairly good understanding of system testing and its industrial status. Regarding the other levels, such as unit (developer level testing), integration and acceptance testing, little, if almost nothing, has been done both within the academia and industry.

Recently, developers', integration and acceptance tests have received more recognition thanks to the agile methods [4-7]. Agile methods treat testing as an integral part of their processes. In these methods, no modification or refactoring of code is complete until 100% of unit tests have run successfully, no story is complete until all its acceptance tests have passed successfully, and additions and modifications to the code are integrated into the system on at least a daily basis. Despite this, we still have little insight into the status of these three types of tests. This insight is pivotal for providing feedback for process improvement and for making the overall development process more cost-effective [8,9].

In this paper, we study developers' testing process at *Nomadic Software*. Our goal is to establish its status within the company and identify areas for potential improvements. The study is based on a testing model [10], developed for a traditional heavyweight development context.

The remainder of this paper is as follows. Section 2 presents our research method and the organization studied. Section 3 describes the developers' testing model. Section 4 presents the status of the testing process and Section 5 makes final remarks.

2. Method

This section describes the research method taken in this study. Subsection 2.1 presents the company studied. Subsection 2.2 describes our research steps. Subsection 2.3 presents the questionnaire used in this study. Finally, Subsection 2.4 motivates the sampling method.

2.1. Nomadic Software

We have studied one large Swedish organization. Due to the sensitivity of the results presented herein, the company does not wish to disclose its name. For this reason, we call it *Nomadic Software*.

Nomadic Software is the IT provider of IT services within a larger group of companies, which we call *The Nomad Group*. This group serves the global market with world-leading products, services and solutions ranging from military defense to civil security products. It is operating in more than 100 countries with its headquarters in Sweden.

2.2. Research Steps

Our research consisted of three phases. As shown in **Figure**

1, these are 1) *Prefatory Study*, 2) *Pilot Investigation*, and 3) *Main Investigation*. Each of the phases consisted of three consecutive steps: *Planning*, *Investigation* and *Analysis*. Below, we briefly describe these phases.

In the *Prefatory Study* phase, we acquainted ourselves with Nomadic Software by investigating its processes and the roles involved in them. For this purpose, we studied the organization's internal documentation and made informal interviews with four developers. Our main purpose was to get background information about the company such as its employees, their working patterns and problems, and their opinions about their testing process. This helped us identify a preliminary status of the developers' testing process and generate questions to be used in later research phases.

After having acquainted ourselves with the company and its process, we decided to make a small survey in the *Pilot Investigation* phase. Here, we created a multiple choice questionnaire based on the results achieved in the former phase. The questionnaire was answered by the same four developers who participated in the *Prefatory Study* phase. Our purpose was to test questions and acquire feedback on their variability and expected answers. This helped us determine the appropriateness of the questions and their level of inquiry.

In the *Main Investigation* phase, we first designed a comprehensive questionnaire to be distributed to all the developers within *Nomadic Software*. Our purpose was to achieve a detailed description of the *AS - IS* situation of the company's developers' testing process, its inherent activities, information managed within the process, roles involved and the tools used.

Even if the questionnaire consisted of multiple choice questions, it became very detailed, and of considerable size. *Nomadic Software* estimated that it would take about one hour for each developer to answer it. Having as many as about eighty developers, it would be too expensive. For this reason, we had to cut out many of its questions and/or redesign others. We also had to split parts of the questionnaire into two sub-parts: one studying static testing and the other one studying dynamic testing. All in all, we received answers from fifteen developers, where seven developers answered the dynamic part and twelve developers answered the static part.

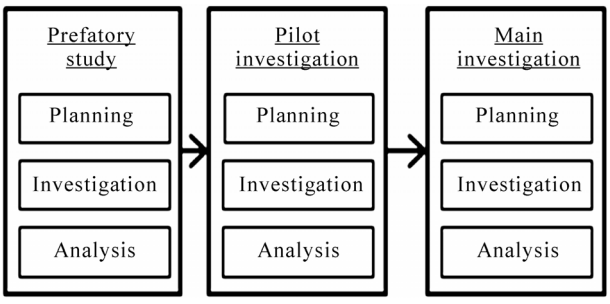


Figure 1. Research steps taken in this study.

2.3. Questionnaire

The questionnaire consists of two main sections: *Background* and *Testing Process*. It is shown in **Figure 2**.

2.3.1. Background Section

The *Background* section inquires about the respondents and the underlying testing conditions within the company studied. It covers the following:

1) *Developers*' *Background*: This part inquires about the developers, their testing experience and current responsibilities. It covers *Questions* 1-3. Our aim is to get to know the developers' background and provide a basis for analyzing the results of this study.

2) *Methods Used*: This part inquires about the development methods defined and used within the company. It covers *Questions* 4-6. Our goal is to find out whether the developers use the methods defined within the company and the reasons behind using or not using them.

3) *Scope and Effort of Testing*: This part inquires about the scope of the developers' testing activities and the effort spent on them. It covers *Questions* 7 and 8. The goal is to find out what testing levels and activities the developers are involved in, what is the effort spent on them, and its distribution on manual and automatic testing.

2.3.2. Testing Process Section

In the Testing Process section, we inquire about the status within the testing phases such as Preparatory, Write Code/ Change Code, Testing, Debugging, Evaluation and Sign-Off (see **Figure 2**).

1) Preparatory Phase: This part inquires about the planning of the developers' work. It includes the following parts:

- Documentation Part: This part inquiring about the documents providing input to the implementation phase. It covers Questions 9-11. Our goal is to find out what documents are studied before the implementation, what measures are taken in cases when they are defective, and elicit examples of the defects.
- Testing Plan: This part inquiring about the developers' test plans. It covers Questions 12 and 13. The goal is to find out what activities are included in the implementation and testing phases and when they are carried out.
- Testing Environment: This part inquiring about the activities the developers conduct to create a testing environment. It covers Question 14. The goal is to find out what activities that the developers create when setting up their own testing environments.

2) Write Code/Change Code Phase: This part inquires about the basic code implementation activities. It covers Question 15. Our aim is to assure that all the respondents write and/or change code, and test it.

Background Section

1) Does your current role entail programming of software?
2) Please state the number of years that you have been working with testing in your current role (at your current employer and in total).
3) What type of activities are you currently involved in?
   ● Development of a completely new system;
   ● Development of a new feature in an existing system;
   ● Defect correction; ● Testing of your own code;
   ● Testing of other developers' code; ● Writing your own test cases; ● Writing other developers' test cases; ● Other, please specify;
4) What test processes/methods do you follow?
   ● Rational Unified Process (RUP); ● Software Development Process (*NomadicRUP*) in all your development; ● Software Development Process, not in your testing; ● Software Test Process (a separate test process linked to *NomadicRUP*; ● Method that you have brought in with you from your former company; ● An old method that has been used earlier at *Nomadic Software*;
   ● Your own method; ● No method at all; ● Other, please specify;
5) If you use use the *Software Development Process (NomadicRUP)*, does it contain enough information regarding unit and unit integration test activities?
6) If you follow the *Software Test Process*, do you believe it to be useful for increasing the code quality?
7) What is your weekly effort (in percentage) spent on manual and dynamic testing?
8) Which types of testing are you involved in?
   ● Unit tests; ● Integrations test of you own units;
   ● Continuous integration of components; ● Functional tests;
   ● Security tests; Regression tests; ● Integrity tests;
   ● Other tests, please specify.

Testing Process Section

9) What documents do you study before coding and testing?
   ● Requirements specification; ● Design specification
   ● Program specification; ● Change request; ● Problem report;
   ● Nothing, everything is communicated orally;
   ● Other test, please specify;
10) If some of the documents have inconsistencies, need further clarification or is missing information, do you report that it needs updating?
11) Please, state which documents need updating and list some of the problems identified in those documents.
12) Which of the activities do you include in your testing plan?
   ● Coding; ● Testing; ● Creating stubs and drivers;
   ● Preparing your testing environment; ● Modifying of your regression test cases; ● Others, please specify;
   ● None, you do not create your own testing plan;
13) When exactly do you carry out those activities?
   ● Before coding; ● During coding; After coding;
   ● Never;
14) Do you create your own testing environment? If yes, what exactly do you do?
15) Which activities do you perform in the *Write Code/Change Code* phase?
   ● Write code; ● Change code; ●Link and compile code;
   ● Other, please specify;

Dynamic Testing Part

16) Which types of dynamic testing do you perform?
   ● Black-box tests; ● White-box tests; ● Grey-box tests;
17) Which role(s) does/do write your test cases?
   ● Another developer writes your test cases; ● An integrator writes your test cases; ● A software architect writes your test cases,
   ● Other role writes your test cases, please specify which role it is;
18) Do your document your own test cases?
   ● Always; ● Very often; ● Half of them; ● Rarely;
   ● Never;
19) What are your testing coverage goals?
20) If you have not achieved your testing goals, what do you do?
21) Which method do you use when designing your input data?
   ● Equivalence partitioning; ● Boundary-value analysis;
   ● Cause-effect graphing; ● Error guessing; ● Statement coverage;
   ● Decision coverage; ● Conditions coverage;
   ● Decision/condition coverage; ● Multiple-condition coverage;
22) When comparing the received testing results with the expected ones, what do you do when you find discrepancies?
   ● You make your own notes; ● You hand in a trouble report;
   ● Other, please specify.
23) Which roles do you get in contact with in the following situations?
   ● Who informs you about the functionality that you have to develop? ● Who informs you about that you have to test other developer's code? ● Who informs you about inconsistencies, needs for clarification and missing information? ● Who do you inform that you that you have updated your own test cases? ● Who do your inform that requirements need to be updated? ● Who do you inform about your testing results? ● Who do you inform that your tests have been completed?

Debugging Part

24) If you are debugging code, what do you when you find discrepancies
   ● Study the source code and correct it, if relevant;
   ● Study test cases(s) and correct it/them, if relevant;
   ● Study the requirements specification and take relevant measures, if relevant; ● Study the design specification and take relevant measures, if relevant; ● Others, please specify;
25) What tool(s) are you using for debugging?

Static Testing Part

26) Are you involved in reviews?
   ● You do reviews of your own code; ● You do walkthroughs (review of a peer's code); ● You do formal inspections; ● You do not do any reviews, walkthroughs or inspections;
27) What is the purpose of your reviews?
   ● Code follows; ● Programming guidelines;
   ● Organizational standards; ● Other criteria, please specify;
28) Do you make notes documenting the results of the reviews?
29) Do you report discrepancies encountered during the reviews?

Sign-off Part

30) Do you sign-off your development and test results before delivering your code for integration or system testing?
31) Do you sign-off your development and test result, exactly which artifacts do you sign-off?

Evaluation Part

32) If you look at how developer's testing is being done today, please state what can be improved and motivate your suggestions.
33) If you look at the way the developer's testing is done today, please state what is the best in today's testing procedures and motivate way.

Figure 2. Our questionnaire.

3) Testing Phase: This part inquires about the status within the test execution phase. It includes the following parts:

- Dynamic Test Execution Phase: This part inquires about the status of the dynamic testing process, its timing and results. It covers Questions 16-23. Our aim is to find out whether dynamic testing activities are conducted, how and when they are conducted, what is their outcome and how they are communicated among the roles involved.
- Static Test Execution Phase: This part inquires about the status of the static testing process and its results. It covers Questions 26-29. Here, we wish to find out whether static testing activities are conducted, how and when they are conducted, and what their outcome is.

4) Test debugging phase: This part inquires about the practice of debugging. It covers Questions 24 and 25. Our aim is to find out what is done when defects are discovered and what tools are used for localizing these defects.

5) Sign-Off Phase: This part finds out whether the developers sign off their results. It covers Questions 30 and 31. Our aim is to hear about whether the developers sign-off their code and what artifacts they sign-off.

6) Evaluation Phase: This part inquires about the developers' opinion about the testing process and their suggestions for the process improvement. Our aim is to elicit the developer's recommendations on how to improve their testing process and/or how to preserve its good elements. It covers Questions 32 and 33.

2.4. Sampling Method

Initially, we intended to achieve a full sampling coverage of our respondents. However, as already mentioned, this was considered to be too expensive by the company's management. Hence, only fifteen developers were involved in this study. These individuals belonged to different projects, and they were chosen by their respective project managers. We had no opportunity to influence their selection. For this reason, we have no other choice than to classify the sampling method used in this study as a convenience sampling method.

The convenience sampling method does not allow us to generalize our results with respect to the status of the organization studied. However, it provides an indication of what the status of the developers' testing process looks like.

3. Testing Model

There are not so many process models delineating the developers' testing process. One of the current ones is illustrated in **Figure 3** [10]. It provides a framework for developers' testing phases and their constituent activities. It was developed in a traditional heavyweight context. However, it is even relevant in the context of agile development. By framework, we mean that it covers most of the activities necessary for conducting unit and unit integration tests.

As shown in **Figure 3**, the phases of the developers' testing process are 1) *Preparatory Phase*, 2) *Write Code/ Change Code Phase*, 3) *Testing Phase*, 4) *Debugging Phase*, 5) *Evaluation Phase* and finally, 6) *Sign-off Phase*.

1) Preparatory Phase

The Preparatory Phase consists of two alternative phases. Their choice depends on whether one writes new code or changes an existing one. The changes may concern changes requested by external customers or changes to be conducted due to discovered defects in any of the testing process phases. The activities for these two phases are almost the same. One makes a new low-level design or checks whether or how to make changes to the existing one. One plans for the next testing iteration, that is, one creates/modifies test cases, specifies/checks inputs and expected outputs, and creates stubs and drivers, if necessary. The only difference is that one revises regression test case base in cases when the code is changed.

2) Write Code/Change Code Phase

During the *Write Code/Change Code Phase*, developers write or change their code and compile it.

3) Testing Phase

The *Testing Phase* consists of unit and unit integration testing which, in turn, may be conducted dynamically and statically. Dynamic testing implies testing software through executing it. One starts by checking if the test cases fulfil the given requirements, one creates additional test cases, if needed, links the units and tests them. The test results are then documented and compared to the expected ones. Static testing, on the other hand, implies testing software through reading it. It ranges from informal code reviews conducted by the developers themselves, to reviews conducted by peers, to formal inspections performed by a group of dedicated roles.

4) Debugging Phase

The *Debugging Phase* is conducted in parallel with the other testing phases. Using the testing results, one localizes defects and removes them. It partly overlaps with the activities within problem management process.

5) Evaluation Phase

The *Evaluation Phase* is conducted on two levels. The first level is performed by developers. They evaluate code before sending it for system integration. The second level evaluates the development and testing routines with the purpose of providing feedback for process improvement.

6) Sign-off Phase

Due to the importance of unit and unit integration tests, the developers should sign off that all the components

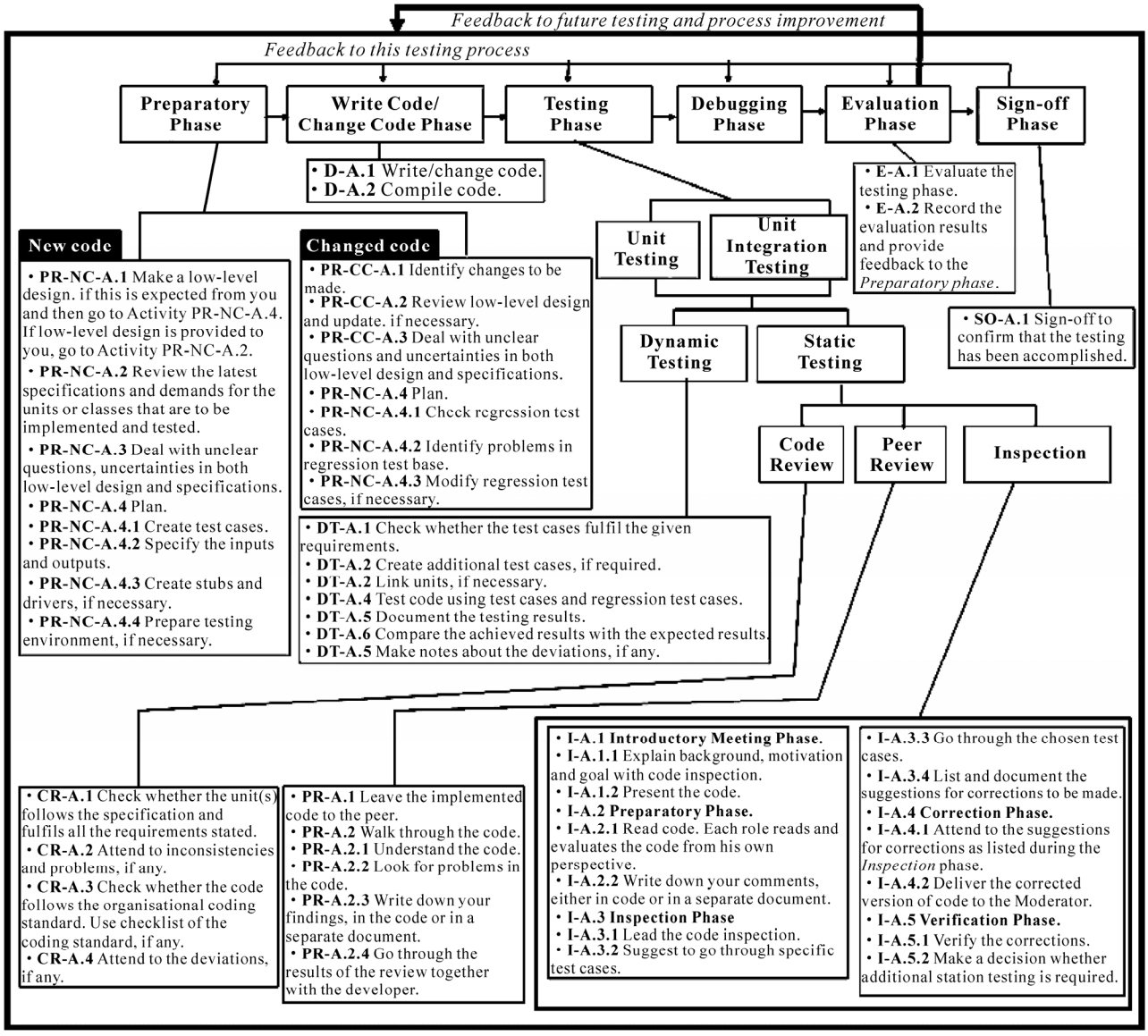


Figure 3. Developers' testing process.

delivered for integration have been successfully tested. We consider the *Sign-off Phase* important because it finalizes the developers' tests. It adds pressure on the developers and hinders them from delivering untested code. It also promotes higher level of accountability among the developers.

The framework does not impose any particular sequence. Developers are free to adapt it to their own context. Usually, before sending their components for integration, they may have to repeat many of its phases or their parts. This is illustrated with a non-bold line in **Figure 3**. In addition, the framework suggests that the developers evaluate the testing process in the *Evaluation* phase and provide feedback for process improvement. This is illustrated with a bold arrow line in **Figure 3**.

4. Status within Nomadic Software

In this section, we present the results of the survey. When reporting on them, we follow the order of the questionnaire as defined in Subsection 2.3.

4.1. Respondents and their Background

All the respondents (100% of response coverage) are involved in programming. As illustrated in **Figure 4**, in average, they have been working with programming and testing for 3.2 ± 3.2 years at *Nomadic Software* and for 7.4 ± 7.1 years in their career lives.

The respondents are involved in various lifecycle phases; 53.3% are involved in developing new systems,

93.3% enhance existing systems with new features and 86.7% attend to software problems. Irrespective of the phase, all the respondents are involved in writing and testing their own code. Out of them, 46.7% write their own test cases and 13.3% write test cases to be used by other developers. Some of them (6.7%) also conduct other unspecified activities.

4.2. Method Used

Nomadic Software has defined and established their own development method. This method is based on RUP [11] and it is called *NomadicRUP*. All the developers are required to follow it either standalone or in combination with the *Software Test Process*, a process that has been defined and established by *Nomadic Software*. Despite this, as shown in **Figure 5**, only 33.3% of the respondents follow it within all their development activities (including testing).

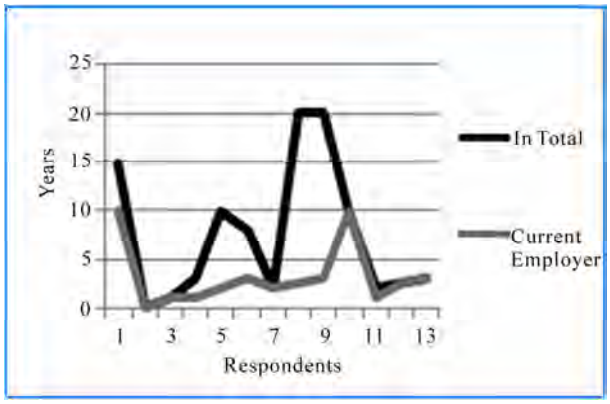


Figure 4. Experience in testing.

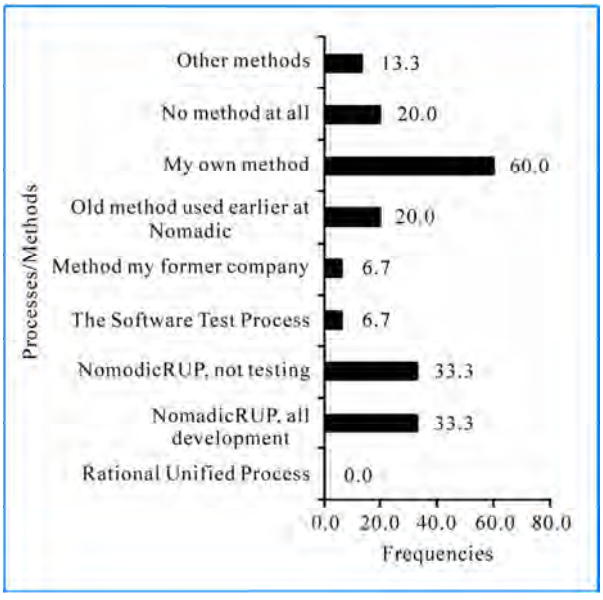


Figure 5. Process/method followed.

Regarding the remaining respondents, 33.3% of them, follow *NomadicRUP* within development but not within testing; 6.7% utilize the *Software Test Process*, 6.7% use a method that they have brought with them from an earlier employer, and 20.0% use an old *Nomadic* method. As many as 60 % use their own method and as many as 20.0% do not use any method at all. Finally, 13.3% of the respondents use methods such as Scrum, XP and ITM Process [12-14].

It is easy to recognize in **Figure 5** that the majority of the respondents follow more than one method. This is proved by calculating the accumulated frequency which is 193.3%.

Our respondents have admitted that they conduct developers' tests in an ad hoc manner. They mainly use common sense when testing their code. However, they claim that they are more disciplined when performing higher-level tests with respect to planning, testing and follow up.

There are many reasons to why *NomadicRUP* is not used by all the developers. Some of: 1) the developers have not even made an effort to get acquainted with the method; hence, they do not use it, 2) the methods are too general and it does not support their specific development needs while the use of Scrum has substantially increased progress and code quality, 3) the developers have gone over to Scrum because they feel that by using *NomadicRUP*, they produce a lot of meaningless and quickly outaging documentation instead of writing code, 4) the developers continue with the *NomadicRUP's* forerunner that was used to develop and that is still used to maintain some of the existing applications, 5) the developers wish to decide by themselves on how to carry out their own testing work.

As shown in **Figure 5**, 66.6% (33.3% + 33.3%) of the respondents follow the *NomadicRUP* method but only 33.3% of them use it for testing purposes. Still, however, 63.7% of them are of the opinion that the method includes sufficient information about developers' testing process.

Regarding the *Software Test Process*, only 6.7% of the respondents follow it (see **Figure 5**). Just as with the *NomadicRUP* method, some of the respondents are of the opinion that even this method includes sufficient information about and guidelines for conducting developers' tests and that it generates better code quality. Some other respondents claim that the very abstract presentation level of the method allows them to state that they follow the method. In reality, however, they use common sense when testing their components.

Irrespective of whether the developers follow the software test process, some of them are of the opinion that is it useful to have a formal testing process on a developers' level. It forces the developers to create test cases on different levels, imposes traceability among them and facilitates future development and change.

However, the main obstacle hindering the developers to do the testing is time. They have little time assigned to do the unit and unit integration tests.

4.3. Scope and Effort of Testing

Testing is mainly done manually at *Nomadic Software*. The respondents had difficulties to estimate the effort spent on the manual and automatic testing. This is because the effort varies from week to week or it depends on the complexity of code. In average, however, as shown in **Figure 6**, the respondents spend $0 \leq 30.9\% \leq 69.2\%$ of their weekly working time (40 hours) on doing manual tests and only $0 \leq 2.4\% \leq 9.1\%$ of their time on doing automatic tests.

Developers conduct various tests. As shown in **Figure 7**, their testing activities range from unit tests (92.9%), through unit integration tests (92.9%), functional tests (71.4%), system regression tests (42.9%), and testing of other developers' integrated components (50.0%). In addition, some of them are involved in tests such as usability tests (28.6%), integrity tests (21.4%), and security tests (21.4%). It is worth mentioning that not all the respondents were familiar with all the test types mentioned in the question.

4.4. Preparatory Phase

Various documents provide basis for starting the coding activity. As shown in **Figure 8**, our respondents mainly use 1) requirement specifications (80.0%), 2) design specifications (73.3%), 3) change requests (86.7%), 4) program specifications (53.3%), and 5) problem reports (60.0%). The use of problem reports supports developers in recreating reported problems and in finding deficiencies in the development and maintenance. However, as many as 13.3% of the respondents use oral communication as a basis for their coding activities. This is because the above-mentioned documents do not always exist. Another reason is the fact that many of the above-mentioned documents are not always of satisfactory quality. Hence, the respondents find it easier to use oral communication as a basis for starting their coding activities.

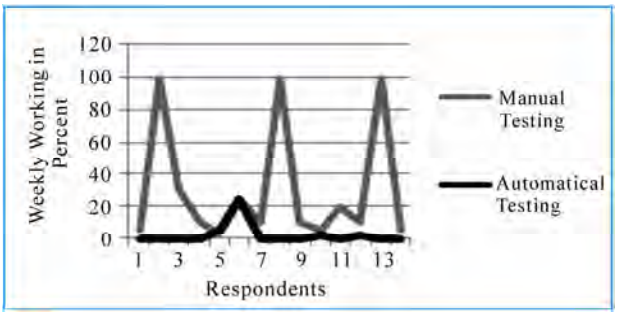


Figure 6. Effort spent on testing.

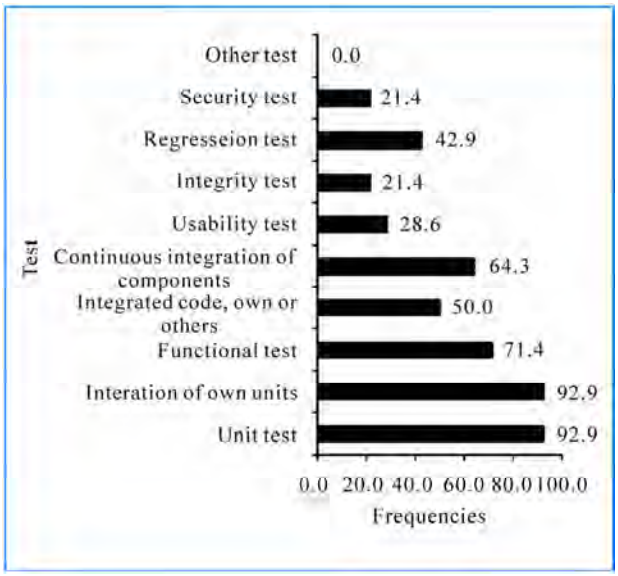


Figure 7. Test conducted.

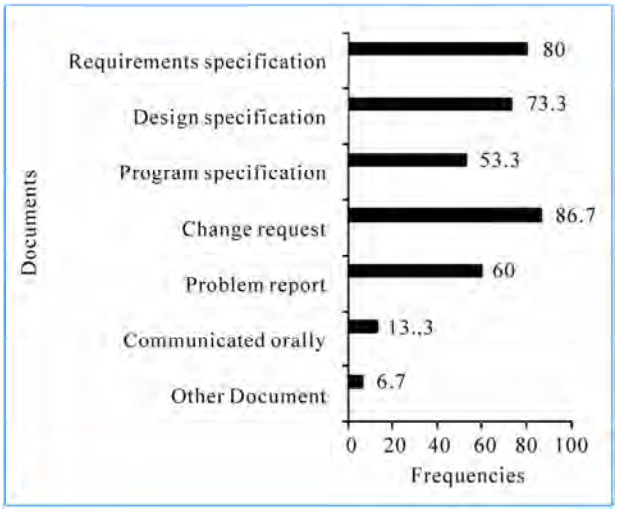


Figure 8. Document.

When studying the above-mentioned documents, the respondents often discover various defects concerning inconsistencies and/or uncertainties. As shown in **Table 1**, the respondents that have answered this question find defects ranging from missing information in design specification to missing or outdated information in various documents. These defects are then reported for corrective measures by 92.9% of the respondents. The reporting is done for the purpose of updating the documents and not for the purpose of providing a basis for improving the testing process. The remaining respondents (7.1%) do not do any reporting at all.

Some of the respondents have not provided any information on what documents they use as a basis for starting their coding and testing activities. They have however provided us with the following opinions: 1) when

Table 1. Defect examples.

Document name	Defect
Requirement specification	A conditions has to few exits Use Cases are on a too high-level Missing business rules
Design specification	Missing information or documents
Change report	Common functionality is not common
Problem report	Outdated information

Table 2. Timing of some testing activities in percentage.

Activity	When			
	Before coding	During coding	After coding	Another time
Stubs and drivers	33.3	66.7	33.3	0.0
New functionality test cases	41.6	41.6	75.0	16.7
Regression test cases	0.0	16.7	83.3	0.0

designing the system I do not add any descriptions about how to conduct unit and integration tests since it is not requested by the organization, 2) I cannot remember a document that does not include inconsistencies and/or uncertainties, and 3) documentation is generally a bad way of communicating information to the implementation process and to keep information about how things work. Therefore, documented tests are a lot better if they are combined with documentation easily extractable from code.

The respondents were asked to list the activities that they included in their testing plans. Only 71.4% of the respondents plan their implementation and testing. In their plans, they include 1) coding (90.0% of the respondents), 2) testing (100%), 3) preparation of their own testing environments (80.0%), and 4) modification of regression test cases (40.0%). Very few of the respondents (20.0%) include creation of stubs and drivers in their testing plans. These plans, however, are made on an informal basis. This is because the organization does not promote planning of and documenting tests.

Regarding the activities included in the testing plan, we inquired about the point in time when they were conducted. Our aim was to find out whether they were conducted 1) before coding, 2) during coding, 3) after coding, or 4) never. As shown in **Table 2**, the timing of these activities varies in the following:

1) Stubs and drivers are created before and during coding.

2) Regression test cases are modified during and after coding. However, the greater majority of the respondents (83.3%) modify them after they have finished coding.

3) New functionality test cases are written throughout the whole implementation process. The majority of the respondents (75%), however, create them after coding.

We also inquired whether the respondents created their own testing environments and exactly what they did when doing it. Eighty percent of the respondents do create their own testing environments. When doing it they (1) test project code and run functional testing of their own components, (2) change test data by making a copy of production data, (3) use remote automatic tests whenever they are checking something in, (4) create a number of settings to point out the resources required to run and execute a build. In addition, the respondents have commented that they have three environments: development, test, and production. However, they only use the development environment when conducting developer's tests (unit tests).

4.5. Write Code/Change Code Phase

All the respondents (100%) write new code and change an existing code. However, 61.6% of the respondents have to compile their code manually. The remaining respondents get it automatically done via tools which both check syntax and compile the code.

4.6. Dynamic Testing

The respondents were requested to list the dynamic testing practices they used. The majority of them (85.7%) conduct black-box and white box tests. Although grey-box testing is not promoted at *Nomadic Software*, 42.9% of the respondents have answered that they conduct grey-box tests as well.

We inquired whether the respondents wrote test cases by themselves or whether they got them written by some other role. We also inquired if they documented their own test cases. Our results show that all the respondents claim that they write their own test cases, but on some occasions, 14.3% of them use test cases written by other developers. No other role than a developer role is involved in writing test cases for our respondents.

We inquired whether the respondents documented their own test cases. Our results show that 42.9% of the respondents always document their test cases, 28.6% do it very often, 14.3% do it rarely, and 14.3% never do it. Some of the respondents have pointed out that one mainly puts effort into documenting the integration test cases instead. Other respondents have mentioned that documentation is only in JavaDoc but that they can generate a report on all tests when they run them.

Developers' tests are the most efficient tests to conduct. Because the cost of coverage is low, one should strive to set a testing coverage goal as high as possible.

We inquired about the developers' coverage goals. Our results are illustrated in **Figure 9**. As shown there, 1) 33.3% of the respondents test all main parts of code, 2) 50% test all code, 3) 16.7% test 60%-80% of all code 4) 16,7% test all features, and 5) 16.7% test all the architectural decisions. These results do not specify testing coverage for any specific testing technique. This is because the coverage goals are not determined by the organization but by the developers themselves. However, as **Figure 10** shows, the white-box testing techniques used by the respondents are 1) multiple-conditions coverage, 2) decision/conditions coverage, 3) condition coverage, 4) decision coverage, and 5) statement coverage.

Regarding the test cases involving input data, most of the respondents (80% of them) use the boundary analysis method, and 60% use error guessing. Some of them also use equivalence partitioning (40%) and cause-effect graphing technique (20%). In cases when the coverage goals are not achieved, the respondents take measures such as 1) discuss cost and revenue of further testing, 2) ask the project manager for further measures, 3) decide by themselves what to do next, or 4) they just checked in code to the repository.

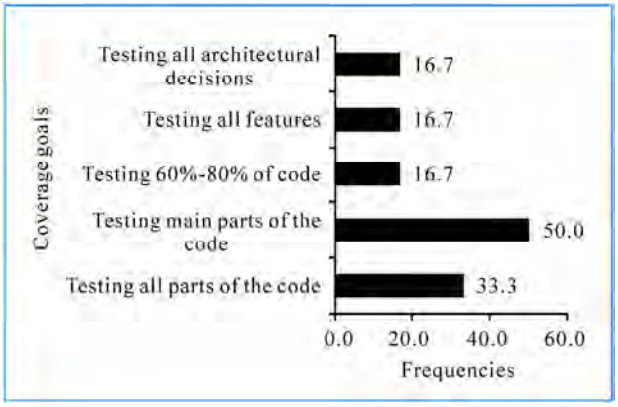


Figure 9. Test coverage goals.

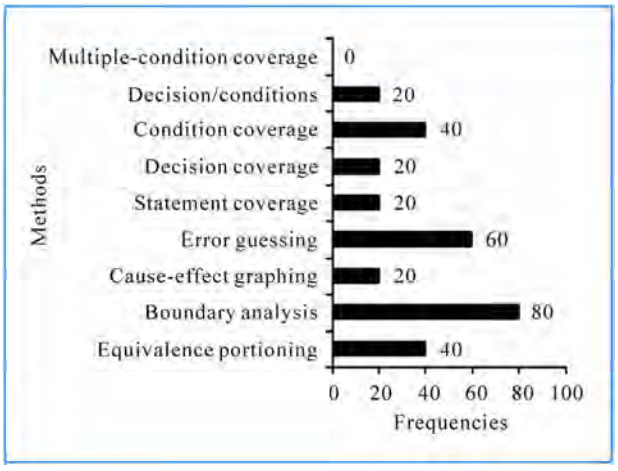


Figure 10. Data input methods used.

Test results ought to be documented. For this reason, we inquired whether the respondents recorded their testing outcome and how they did it. Our results show that 57.1% of the respondents make their own informal notes about the discrepancies between the expected and achieved results and 28.6% hand in trouble reports, if necessary. Some of the respondents (42.9%) not only make notes or hand in trouble reports but also correct the code by themselves. Finally, some of the respondents just fix code without making either formal or informal notes.

Developers come in contact with various roles in different situations. These are:

1) *System Analysts* and *System Architects* to discuss new functionality to be developed, suggestions for their updates and reports on inconsistencies in them, if any.

2) *Business System Manager* and *End User* to discuss maintenance tasks and inconsistencies in them, if any.

3) *Test Managers* requiring that the respondents test other developers' tests. The respondents may also inform the *Test Managers* about the completion of their tests and their testing results.

4.7. Test debugging Phase

We inquired about how the developers tracked defects in the *Debugging* phase and what tool support they used. Our results show that all the respondents debug their code, if needed. If they find defects, 100% of them correct them in source code and requirements, and 85.7% correct them in design specifications and test cases. The tools used during the *Debugging* phase are, for instance, *Visual Studio, IntelliJ, JProfiler* and *Jboss*.

4.8. Static Testing

Given a set of static testing practices, the respondents were requested to identify the ones they used. They had a choice of 1) own reviews implying that they checked their own code, 2) walkthroughs of peer code, and 3) formal inspections. Our results show that 100% of the respondents review their own code, 9.1% do walkthroughs of peer code, and 18.1% are involved in inspections. The inspections, however, are very seldom performed.

We inquired about the purpose of the reviewing activities. Irrespective of how the developers review their code (own review or walkthroughs), at least 90.1% of the respondents review it for the consistency with the requirements. When conducting own reviews, 63.6% of the respondents also review for organizational standards, and 9.1% review for other criteria such as, for instance, international standards. In the context of walkthroughs, 18.2% of the respondents review for organizational standards only. In situations when the respondents con-

duct inspections, they only review for consistency with requirements.

In static testing, it is imperative to document the testing results. Hence, we inquired whether the respondents recorded them. Our results show that very few respondents, only 18.2% of all of them, document the results of their own code reviews and no one documents walkthrough and inspection results.

We also inquired how the respondents documented the discrepancies discovered during static testing. As illustrated in **Table 3**, the results span between 54.5% of the respondents making own notes to 9.1% of the respondents handing in trouble reports. The remaining discrepancies are communicated on an oral basis.

4.9. Sign off

We asked the respondents whether they finalized their implementation and testing activities in a formal or informal way, for instance, by signing off their code. Only 9.1% of the respondents sign-off their work after they have completed their tests. The artifact that is used for signing-off is mainly a version management tool complemented by an informal hand-shake among the developers, testers and managers.

4.10. Evaluation

We also inquired about the best of the today's testing process. According to the respondents, the best parts of the process are 1) the ability to conduct test review, 2) freedom to use, for instance, Scrum/XP instead of, for instance, *NomadicRUP*, 3) the ability to import production data to be used as test data, 4) the ability to test your own code, 5) automatic test framework, 6) the opportunity to start testing early in the development cycle. Their motivations are 1) system development using Scrum/XP generates less defects, 2) test data can always be up to date since it is possible to copy production data, 3) the framework automatically conducts regression tests, 4) by placing testing early in the development cycle, focus is set on the actual problems and assures that test cases are written, and finally, 5) the transfer of documents is substantially reduced.

Table 3. Recording Testing Results

Methods	I make own notes	I hand in trouble reports	Other, please specify	I do not document
Own Review	54.5	18.2	9.1	18.2
Walkthroughs	9.1	0	9.1	64.7
Inspections	27.2	9.1	9.1	45.5

5. Final Remarks

In this paper, we have studied developers' testing process at *Nomadic Software*. Our goal is to establish its status within the company and identify areas for potential improvements. The study is based on a traditional testing model elicited in [10]. The respondents involved in this study are developers with solid programming background and experience.

Our results show that the developers' testing process is not uniformly performed within *Nomadic Software*. Right now, the company suffers from the following problems:

1) *Lack of control over the methods used*: Even if *Nomadic Software* has put effort into defining and establishing a development and testing process, the majority of the developers still use other methods and they conduct their tests in an ad hoc manner. Irrespective of the reasons behind, *Nomadic Software* did not have insight into what methods were used within the company before this study. Neither did it have control over the status of the developers' testing process. Regarding the developers, some of them are hardly acquainted with the company's testing method.

2) *The organization does not assign enough time for conducting developers' tests*. This leads to the fact that developers' tests get neglected. Developers are too much in a hurry to deliver code for integration and system tests.

3) *Testing coverage goals are not clearly stated by the organizations studied*. Neither are they determined for any specific testing technique. This implies that each developer sets his own goals. This, in turn, may lead to strongly varying code quality as delivered by various developers.

4) *Important requirements and defects in requirements specifications are communicated orally*: Quite a big portion of requirements and problems are communicated orally. These requirements and problems do not get documented even after being implemented. This is a severe problem that may substantially degrade the system maintainability and contribute to quick software ageing and lack of control over the development and maintenance process [15].

5) *Not all test cases get documented*: This implies that the company cannot determine whether the developers' testing has been sufficiently performed. This also implies that regression testing on the developers' level practically does not exist.

6) *Static testing is not practiced enough*: Static testing is performed on an informal basis. At its most, developers review their own code and sometimes their peers' code. Formal inspections of critical code parts are conducted very seldom.

7) *Lack of testing guidelines*: Lack of testing guide-

lines makes developers decide by themselves on how to conduct their testing activities. This, in turn, leads to the non-uniformity of the testing process execution.

8) *Insufficient education within testing*: The employees at *Nomadic Software* get a very short education on development method, where testing is one of its parts. Hence, they have not acquired sufficient knowledge. This is clearly evident from the fact that the respondents are not acquainted with some basic testing terms such as integrity tests or they use the terms differently. A similar phenomenon has been observed in our former study in [16].

9) *Lack of testing strategy*: *Nomadic Software* lacks a strategy aiding them in defining how to test in a cost-effective and qualitative manner and designating test types to be part of the testing process.

Due to the sampling method used in this study, we cannot generalize the results presented herein. However, we may still claim that our results strongly indicate that just as *Nomadic Software*, many software companies are in great need to revise their developers' testing process, put it in the context of its overall testing process and make effort into improving it.

When studying the developer's testing process at *Nomadic Software*, we have identified several problem areas related to the education of developers and the management and execution of the testing process. Specific pains that we have observed are lack of control over the testing methods used, lack of testing strategies and lack of directives of what is expected from the developers. To attend to these problem areas is not an easy task. It requires many different measures ranging from creating appropriate overall testing strategies in which developer's testing strategy is clearly identified and specified, defining testing processes in which developers' tests play an essential role, and monitoring that they are followed by the developers. To realize them can be a long and complex process. However, as an initial step towards improving the developers' testing process, we suggest the software community create guidelines providing instructions and recommendations specifying what and how developers' tests should be done and what sort of actions should be taken in particular testing circumstances.

6. References

[1] J. W. Cangussu, R. A. DeCarlo and A. P. Mathur, "A Formel Model of the Software Test Process," *IEEE Transactions on Software Engineering*, Vol. 28, No. 8, 2002, pp. 782-796.

[2] L. Groves, R. Nickson, G. Reeves, S. Revves and M. Utting, "A Survey of Software Pratices in the New Zeeland Software Industry," *Proceedings of Australian Software Engineering Conference*, Queensland, 28-29 April 2000, pp. 189-201.

[3] S. P. Ng, T. Murnane, K. Reed, D. Grant and T. Y. Chen, "A Preliminary Survey on Software Practices in Australia," *Proceedings of Australian Software Engineering Conference*, Melbourne, 13-16 April 2004, pp. 116-125.

[4] "Agile Software Development," 2009. http://en.wikipedia.org/wiki/Agile_software_development

[5] H. Gallis, E. Arisholm and T. Dyka, "An Initial Framwork for Research on Pair Programming," *Proceedings of ISESE International Symposium on Empirical Software Engineering*, Rome, 30 September-1 October 2003, pp. 132-142.

[6] E. M. Guerra and C. T. Fernandes, "Refactoring Test Code Safely," *Proceedings of ICSEA International Conference on Software Engineering Advances*, Cap Esterel, 25-31 August 2007, p. 44.

[7] P. J. Schroeder and D. Rothe, "Teaching Unit Testing using Test-Driven Development," 2005. http://www.testingeducation.org/conference/wtst4/pjs_wtst4.pdf

[8] S. Koroorian and M. Kajko-Matsson, "A Tale of Two Daily Build Projects," *Proceedings of International Conference on Software Engineering Advances*, Porto, 20-25 September 2009, pp. 245-251.

[9] G. J. Meyers, T. Badgett, T. M. Thomas and C. Snadler, "The Art of Software Testing," 2nd Edition, John Wiley & Sons, Inc., Hoboken, 2004.

[10] M. Kajko-Mattsson and T. Björnsson, "Outlining Developer's Testing Mode," *Proceedings of EUROMICRO Conference on Software Engineering and Advanced Applications*, Lübeck, 27-31 August 2007, pp. 263-270.

[11] B. Henderson-Sellers, G. Collins and I. Graham, "UML-Compatible Process," *Proceedings of* 34*th Annual Hawaii International Conference on System Sciences*, Maui, Vol. 3, 3-6 January 2001, p. 3050.

[12] "ITM Process (IT-Product Maintenance Process)," Internal Documentation at Nomadic Software, 2009.

[13] R. Juric, "Extreme Programming and its Development Practices," *Proceedings of* 22*nd ITI International Conference Information Technology Interfaces*, Pula, 13-16 June 2000, pp. 97-104.

[14] L. Rising and N. S. Janoff, "The Scrum Development Process for Small Teams," 2000. http://members.cox.net/risingl1/Articles/IEEEScrum.pdf

[15] M. Kajko-Matsson, "Corrective Maintenance Maturity Model: Problem Management," Ph.D. Dissertation, Stockholm University and Royal Institute of Technology, Stockholm, 2001.

[16] M. Kajko-Mattsson, "Common Concept Apparatus within Corrective Software Maintenance," *Proceedings of International Conference on Software Maintenance*, Los Alamitos, 30 August-3 September 1999, pp. 287-297.

Intelligent Information Management, **2010, 2, 354-364**
doi:10.4236/iim.2010.26043 Published Online June 2010 (http://www.SciRP.org/journal/iim)

Intelligent Optimization Methods for High-Dimensional Data Classification for Support Vector Machines

Sheng Ding[1,2], Li Chen[1]
[1]*College of Computer Science and Technology, Wuhan University of Science and Technology, Wuhan, China*
[2]*School of Remote Sensing and Information Engineering, Wuhan University, Wuhan, China*
E-mail: *dingwhu@gmail.com*
Received March 2, 2010; *revised April* 3, 2010; *accepted May* 4, 2010

Abstract

Support vector machine (SVM) is a popular pattern classification method with many application areas. SVM shows its outstanding performance in high-dimensional data classification. In the process of classification, SVM kernel parameter setting during the SVM training procedure, along with the feature selection significantly influences the classification accuracy. This paper proposes two novel intelligent optimization methods, which simultaneously determines the parameter values while discovering a subset of features to increase SVM classification accuracy. The study focuses on two evolutionary computing approaches to optimize the parameters of SVM: particle swarm optimization (PSO) and genetic algorithm (GA). And we combine above the two intelligent optimization methods with SVM to choose appropriate subset features and SVM parameters, which are termed GA-FSSVM (Genetic Algorithm-Feature Selection Support Vector Machines) and PSO-FSSVM(Particle Swarm Optimization-Feature Selection Support Vector Machines) models. Experimental results demonstrate that the classification accuracy by our proposed methods outperforms traditional grid search approach and many other approaches. Moreover, the result indicates that PSO-FSSVM can obtain higher classification accuracy than GA-FSSVM classification for hyperspectral data.

Keywords: Support Vector Machine (SVM), Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Feature Selection, Optimization

1. Introduction

Support vector machine (SVM) was first proposed by Vapnik [1] and has recently been applied in a range of problems including pattern recognition, bioinformatics and text categorization. SVM classifies data with different class labels by determining a set of support vectors that are members of the set of training inputs that outline a hyperplane in the feature space. When using SVM, two issues should be solved: how to choose the optimal input feature subset for SVM, and how to set the best kernel parameters. Traditionally, the two issues are solved separately ignoring their close connections, this always leads low classification accuracy. These two problems are crucial, because the feature subset choice influences the appropriate kernel parameters and vice versa [2]. Therefore, obtaining the optimal feature subset and SVM parameters must occur simultaneously.

Feature selection is used to identify a powerfully predictive subset of fields within a database and reduce the number of fields presented to the mining process. By extracting as much information as possible from a given data set while using the smallest number of features, we can save significant computational time and build models that generalize better for unseen data points. Feature subset selection is an important issue in building an SVM-based classification model.

As well as feature selection, the proper setting of parameters for the SVM classifier can also increase classification accuracy. The parameters that should be optimized include penalty parameter C and the kernel function parameters such as the gamma (γ) for the radial basis function (RBF) kernel. To design a SVM classifier, one must choose a kernel function, set the kernel parameters and determine a soft margin constant C (penalty parameter). As a rule, the Grid algorithm is an alternative to finding the best C and gamma (γ) when using the RBF kernel function. However, this method is time consuming and does not perform well [3]. Moreover, the Grid algorithm can not perform the feature selection task.

Both feature subset selection and model parameter setting substantially influence classification accuracy. The optimal feature subset and model parameters must be determined simultaneously. Since feature subset and model parameters greatly affects the classification accuracy.

To simultaneously optimize the feature subset and the SVM kernel parameters, this study attempts to increase the classification accuracy rate by employing two evolutionary computing optimization-based approaches: genetic algorithm (GA) and particle swarm optimization (PSO) in SVM. These novel approaches are termed PSO-FSSVM (Particle Swarm Optimization-Feature Selection Support Vector Machines) and GA-FSSVM (Genetic Algorithm-Feature Selection Support Vector Machines). The developed approaches not only tune the parameter values of SVM, but also identify a subset of features for specific problems, maximizing the classification accuracy rate of SVM. This makes the optimal separating hyperplane obtainable in both linear and non-linear classification problems.

The remainder of this paper is organized as follows. Section 2 reviews pertinent literature on SVM and the feature selection. Section 3 then describes basic GA concept and GA-FSSVM model of feature selection and parameter optimization. Also, Section 3 then describes in detail the developed PSO-FSSVM approach for determining the parameter values for SVM with feature selection. Section 4 compares the experimental results with those of existing traditional approaches. Conclusions are finally drawn in Section 5, along with recommendations for future research.

2. Literature Review

Approaches for feature selection can be categorized into two models, namely a filter model and a wrapper model [4]. Statistical techniques, such as principal component analysis, factor analysis, independent component analysis and discriminate analysis can be adopted in filter-based feature selection approaches to investigate other indirect performance measures, most of which are based on distance and information. Chen and Hsieh [5] presented latent semantic analysis and web page feature selection, which are combined with the SVM technique to extract features. Gold [6] presented a Bayesian viewpoint of SVM classifiers to tune hyper-parameter values in order to determine useful criteria for pruning irrelevant features.

The wrapper model [7] applies the classifier accuracy rate as the performance measure. Some researchers have concluded that if the purpose of the model is to minimize the classifier error rate, and the measurement cost for all the features is equal, then the classifier's predictive accuracy is the most important factor. Restated, the classifier should be constructed to achieve the highest classification accuracy. The features adopted by the classifier are then chosen as the optimal features. In the wrapper model, meta-heuristic approaches are commonly employed to help in looking for the best feature subset. Although meta-heuristic approaches are slow, they obtain the (near) best feature subset. Shon [8] employed GA to screen the features of a dataset. The selected subset of features is then fed into the SVM for classification testing. Zhang [9] developed a GA-based approach to discover a beneficial subset of features for SVM in machine condition monitoring. Samanta [10] proposed a GA approach to modify the RBF width parameter of SVM with feature selection. Nevertheless, since these approaches only consider the RBF width parameter for the SVM, they may miss the optimal parameter setting. Huang and Wang [11] presented a GA-based feature selection and parameters optimization for SVM. Moreover, Huang *et al*. [12] utilized the GA-based feature selection and parameter optimization for credit scoring.

Several kernel functions help the SVM obtain the optimal solution. The most frequently used such kernel functions are the polynomial, sigmoid and radial basis kernel function (RBF). The RBF is generally applied most frequently, because it can classify high-dimensional data, unlike a linear kernel function. Additionally, the RBF has fewer parameters to set than a polynomial kernel. RBF and other kernel functions have similar overall performance. Consequently, RBF is an effective option for kernel function. Therefore, this study applies an RBF kernel function in the SVM to obtain optimal solution. Two major RBF parameters applied in SVM, C and γ, must be set appropriately. Parameter C represents the cost of the penalty. The choice of value for C influences on the classification outcome. If C is too large, then the classification accuracy rate is very high in the training phase, but very low in the testing phase. If C is too small, then the classification accuracy rate is unsatisfactory, making the model useless. Parameter γ has a much greater influence on classification outcomes than C, because its value affects the partitioning outcome in the feature space. An excessively large value for parameter γ results in over-fitting, while a disproportionately small value leads to under-fitting. Grid search [13] is the most common method to determine appropriate values for C and γ. Values for parameters C and γ that lead to the highest classification accuracy rate in this interval can be found by setting appropriate values for the upper and lower bounds (the search interval) and the jumping interval in the search. Nevertheless, this approach is a local search method, and vulnerable to local optima. Additionally, setting the search interval is a problem. Too large a search interval wastes computational resource, while too small a search interval might render a satisfactory outcome impossible.

In addition to the commonly used grid search approach, other techniques are employed in SVM to improve the possibility of a correct choice of parameter values. Pai and Hong [14] proposed an SA-based approach to obtain parameter values for SVM, and applied it in real data; however, this approach does not address feature selection, and therefore may exclude the optimal result. As well as the two parameters C and γ, other factors, such as the quality of the feature's dataset, may influence the classification accuracy rate. For instance, the correlations between features influence the classification result. Accidental removal of important features might lower the classification accuracy rate. Additionally, some dataset features may have no influence at all, or may contain a high level of noise. Removing such features can improve the searching speed and accuracy rate.

It is worth underlining that the kernel-based implementation of SVM involves the problem of the selection of multiple parameters, including the kernel parameters (e.g., the γ and p parameters for the Gaussian and polynomial kernels, respectively) and the regularization parameters C.

Studies have also illustrated that a radial basis kernel yields the best results in remote sensing applications [15, 16]. We chose to use the radial basis kernel for SVM in this study. The verification of the applicability of other specialized kernel functions for the classification of remote sensing data may be used in future studies. The equation for the radial basis kernel is

$$K(x_i, x) = \exp\left(-g \parallel x_i - x \parallel^2\right) \quad (1)$$

where γ represents a parameter inversely proportional to the width of the Gaussian kernel.

3. The Proposed GA-FSSVM and PSO-FSSVM Models

3.1. Genetic Algorithm

The genetic algorithms are inspired by theory of evolution; it is type of an evolutionary computing. The problems are solved by an evolutionary process resulting in a fittest solution in genetic algorithm. A genetic algorithm (GA) is used to solve global optimization problems. The procedure starts from a set of randomly created or selected possible solutions, referred to as the population. Every individual in the population means a possible solution, referred to as a chromosome. Within every generation, a fitness function should be used to evaluate the quality of every chromosome to determine the probability of it surviving to the next generation; usually, the chromosomes with larger fitness have a higher survival probability. Thus, GA should select the chromosomes with larger fitness for reproduction by using operations like selection, crossover and mutation in order to form a new group of chromosomes which are more likely to reach the goal. This reproduction goes through one generation to another, until it converges on the individual generation with the most fitness for goal functions or the required number of generations was reached. The optimal solution is then determined.

GA coding strategies mainly include two sectors: one sector recommends the least digits for coding usage, such as binary codes, another one recommends using the real-valued coding based on calculation convenience and accuracy. Binary codes are adopted for the decision variables in solving the discrete problems, a suitable encoding scheme is needed to encode the chromosome of each individual, in our study, an encoding scheme is usually a binary string. We may define the length of bit string according the precision.

3.2. GA-FSSVM Model

As mentioned before, a kernel function is required in SVM for transforming the training data. This study adopts RBF as the kernel function to establish support vector classifiers, since the classification performance is significant when the knowledge concerning the data set is lacking. Therefore, there are two parameters, C and γ, required within the SVM algorithm for accurate settings, since they are closely related to the learning and predicting performance. However, determining the values exactly is difficult for SVM. Generally, to find the best C and γ, a given parameter is first fixed, and then within the value ranges another parameter is changed and cross comparison is made using the grid search algorithm. This method is conducted with a series of selections and comparisons, and it will face the problems of lower efficiency and inferior accuracy when conducting a wider search. However, GA for reproduction could provide the solution for this study. The scheme of an integration of GA and SVM is shown in **Figure 1**, to establish a training and SVM classification model that can be used to determine optimized SVM parameters and subset features mask. Following the above scheme of the proposed GA-FSSVM model, **Figure 1** describes the operating procedure in this study.

A fitness function assesses the quality of a solution in the evaluation step. The crossover and mutation functions are the main operators that randomly impact the fitness value. Chromosomes are selected for reproduction by evaluating the fitness value. The fitter chromosomes have higher probability to be selected into the recombination pool using the roulette wheel or the tournament selection methods. New population replaces the old population using the elitism or diversity replacement strategy and forms a new population in the next generation. The evolutionary process operates many generations until termination condition is satisfied.

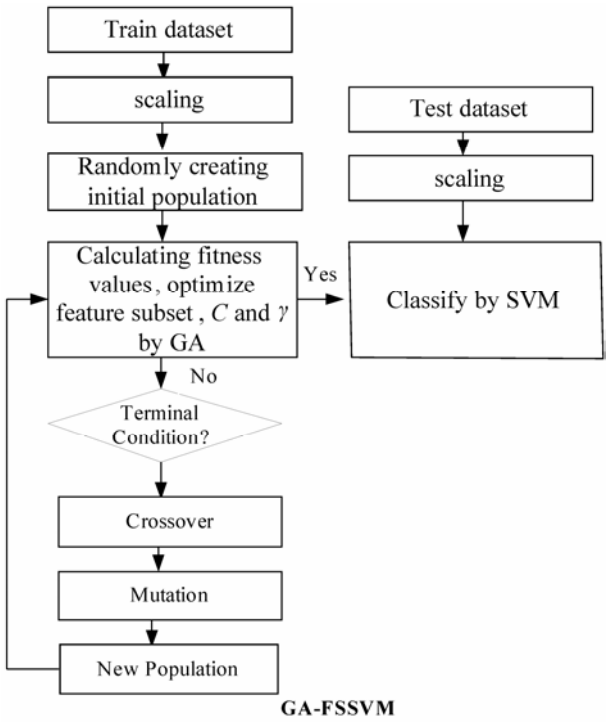


Figure 1. System architecture of the integrated GA-FSSVM scheme.

To implement our proposed approach, this research uses the RBF kernel function for the SVM classifier because the RBF kernel function can analysis higher-dimensional data and requires that only two parameters, *C* and γ be defined When the RBF kernel is selected, the parameters (*C* and γ) and features used as input attributes must be optimized using our proposed GA-based system. Therefore, the chromosome comprises three parts: *C*, γ and the features mask. However, these chromosomes have different parameters when other types of kernel functions are selected. The binary coding system is used to represent the chromosome.

Figure 2 shows the binary chromosome representation of our design. In **Figure 2**, $g_c^1 \sim g_c^{n_c}$ represents the value of parameter *C*, $g_g^1 \sim g_g^{n_g}$ represents the parameter value γ, and $g_f^1 \sim g_f^{n_f}$ represents the feature mask. n_c is the number of bits representing parameter *C*, n_g is the number of bits representing parameter g, and n_f is the number of bits representing the features. Note that we can choose n_c and n_g according to the calculation precision required, and that n_f equals the number of features varying from the different datasets. In **Figure 2**, the bit strings representing the genotype of parameter *C* and γ should be transformed into phenotype. Note that the precision of representing parameter depends on the

$$g_c^1 \mathbf{K}\, g_c^i \mathbf{K}\, g_c^{n_c} \quad g_g^1 \mathbf{K}\, g_g^i \mathbf{K}\, g_g^{n_g} \quad g_f^1 \mathbf{K}\, g_f^i \mathbf{K}\, g_f^{n_f}$$

Figure 2. The chromosome comprise three parts: *C*, γ and the features mask.

length of the bit string, and the minimum and maximum value of the parameter is determined by the user. For chromosome representing the feature mask, the bit with value '1' represents the feature is selected, and '0' indicates feature is not selected. In our study, classification accuracy, the numbers of selected features are the criteria used to design a fitness function. Thus, for the individual with high classification, a small number of features produce a high fitness value.

3.3. Particle Swarm Optimization

Particle swarm optimization (PSO) [17] is an emerging population-based meta-heuristic that simulates social behavior such as birds flocking to a promising position to achieve precise objectives in a multi-dimensional space. Like evolutionary algorithms, PSO performs searches using a population (called swarm) of individuals (called particles) that are updated from iteration to iteration. To discover the optimal solution, each particle changes its searching direction according to two factors, its own best previous experience (*pbest*) and the best experience of all other members (*gbest*). They are called *pbest* the cognition part, and *gbest* the social part. Each particle represents a candidate position (*i.e.*, solution). A particle is considered as a point in a *D*-dimension space, and its status is characterized according to its position and velocity. The *D*-dimensional position for the particle *i* at iteration *t* can be represented as $x_i^t = \{x_{i1}^t, x_{i2}^t, \ldots x_{iD}^t\}$. Likewise, the velocity (*i.e.*, distance change), which is also an D-dimension vector, for particle *i* at iteration *t* can be described as $v_i^t = \{v_{i1}^t, v_{i2}^t, \ldots v_{iD}^t\}$.

Let $p_i^t = \{p_{i1}^t, p_{i2}^t, \ldots p_{iD}^t\}$ represent the best solution that particle *i* has obtained until iteration *t*, and $p_g^t = \{p_{g1}^t, p_{g2}^t, \ldots p_{gD}^t\}$ denote the best solution obtained from p_i^t in the population at iteration *t*. To search for the optimal solution, each particle changes its velocity according to the cognition and social parts as follows:

$$V_{id}^t = V_{id}^{t-1} + c_1 r_1 \left(P_{id}^t - x_{id}^t\right) + c_2 r_2 \left(P_{gd}^t - x_{id}^t\right) \tag{2}$$

$$S\left(v_{id}^{t+1}\right) = \frac{1}{1+e^{-v_{id}^{t+1}}}$$
$$if \quad \left(rand < S\left(v_{id}^{t+1}\right)\right) \quad then \quad X_{id}^{t+1} = 1 \quad else \quad X_{id}^{t+1} = 0 \tag{3}$$

d=1, 2,…, D where c_1 indicates the cognition learning factor; c_2 indicates the social learning factor, and r_1 and

r_2 are random numbers uniformly distributed in $U(0,1)$. Each particle then moves to a new potential solution based on the following equation: $X_{id}^{t+1} = X_{id}^{t} + V_{id}^{t}$, $d = 1, 2,\ldots, D$. The basic process of the PSO algorithm is given as follows.

Step 1: (Initialization) Randomly generate initial particles.

Step 2: (Fitness) Measure the fitness of each particle in the population.

Step 3: (Update) Compute the velocity of each particle with Equation (2).

Step 4: (Construction) For each particle, move to the next position according to Equation (3).

Step 5: (Termination) Stop the algorithm if termination criterion is satisfied; return to Step 2 otherwise the iteration is terminated if the number of iteration reaches the pre-determined maximum number of iteration.

Figure 3 shows the flowchart for PSO-SVM classifier.

Based on the rules of particle swarm optimization, we set the required particle number first, and then the initial coding alphabetic string for each particle is randomly produced. In our study, we coded each particle to imitate a chromosome in a genetic algorithm.

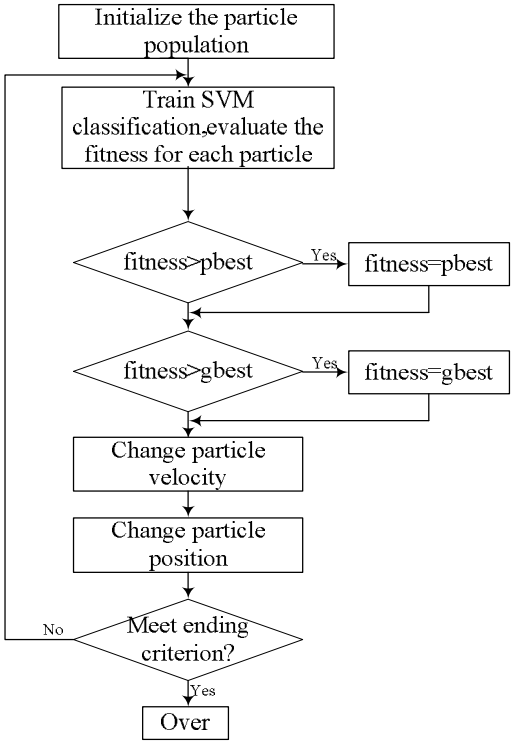


Figure 3. The process of PSO algorithm for solving optimization problems.

3.4. PSO-FSSVM Model

In this following of the section, we describe the proposed SVM-PSO classification system for the classification of high-dimensional data. As mentioned in the Introduction, the aim of this system is to optimize the SVM classifier accuracy by automatically: 1) detecting the subset of the best discriminative features (without requiring a user-defined number of desired features) and 2) solving the SVM-RBF model selection issue (*i.e.*, estimating the best values of the regularization and kernel parameters). In order to attain this, the system is derived from an optimization framework based on PSO.

This study developed a PSO approach, termed PSO-FSSVM, for parameter determination and feature selection in the SVM. Without feature selection, two decision variables, designated C and γ are required. For the feature selection, if n features are required to decide which features are chosen, then $2 + n$ decision variables must be adopted. The value of n variables ranges between 0 and 1. If the value of a variable is less than or equal to 0.5, then its corresponding feature is not chosen. Conversely, if the value of a variable is greater than 0.5, then its corresponding feature is chosen. **Figure 4** illustrates the solution representation. $P_c = C$, $P_\gamma = \gamma$, a_n = Feature n is selected or not.

The following shows the while process for PSO-FSSVM. First, the population of particles is initialized, each particle having a random position within the D-dimensional space and a random velocity for each dimension. Second, each particle's fitness for the SVM is

1	2	3	.....	$n + 2$
P_c	P_γ	a_1	.....	a_n

Figure 4. Solution representation.

evaluated. The each particle's fitness in this study is the classification accuracy. If the fitness is better than the particle's best fitness, then the position vector is saved for the particle. If the particle's fitness is better than the global best fitness, then the position vector is saved for the global best. Finally the particle's velocity and position are updated until the termination condition is satisfied. **Figure 5** shows the architecture of the developed PSO-based parameter determination and feature selection approach for SVM.

4. Experimental Results

To evaluate the classification accuracy of the proposed system in different classification tasks, we tried two real-world datasets: the UCI database [18] and hyperspectral benchmark data set [19]. These two data sets have been frequently used as benchmarks to compare the performance of different classification methods in the literature.

Our implementation was carried out on the Matlab 7.0 development environment by extending the LIBSVM

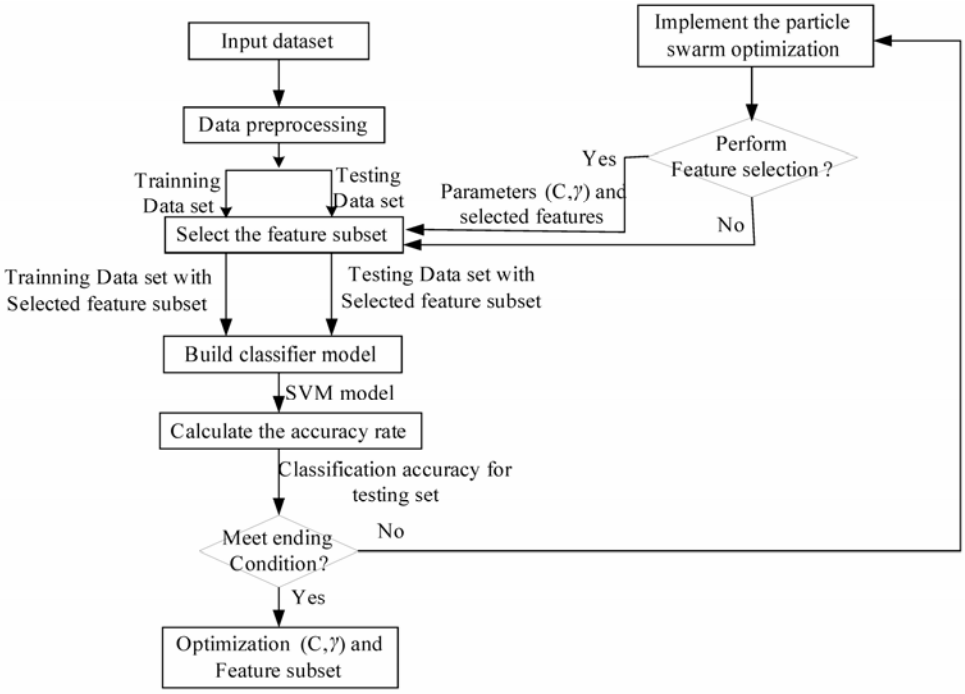


Figure 5. The architecture of the proposed PSD-SVM based parameters determination and feature selection approach for SVM.

which is originally designed by Chang and Lin [20]. The empirical evaluation was performed on Intel Pentium Dual-Core CPU running at 1.6 GHz and 2G RAM.

4.1. UCI Dataset

These UCI datasets consist of numeric and nominal attributes. To guarantee valid results for making predictions regarding new data, the dataset is further randomly partitioned into training sets and independent test sets via a *k*-fold cross validation. Each of the *k* subsets acted as an independent holdout test set for the model trained with the remaining $k-1$ subsets. The advantages of cross validation are that all of the test sets were independent and the reliability of the results could be improved. The data set is divided into *k* subsets for cross validation. A typical experiment uses $k = 10$. Other values may be used according to the data set size. For a small data set, it may be better to set larger *k*, because this leaves more examples in the training set. This study used $k = 10$, meaning that all of the data will be divided into 10 parts, each of which will take turns at being the testing data set. The other nine data parts serve as the training data set for adjusting the model prediction parameters.

The Grid search algorithm is a common method for searching for the best C and g. **Figure 6** shows the process of Grid algorithm combined with SVM classifier. In the Grid algorithm, pairs of (C, g) are tried and the one with the best cross-validation accuracy is chosen. After identifying a 'better' region on the grid, a finer grid search on that region can be conducted. This research conducted the experiments using the proposed GA-SVM and PSO-SVM approaches and the Grid algorithm. The results from the proposed method were compared with that from the Grid algorithm. In all of the experiments 10-fold cross validation is used to estimate the accuracy of each learned classifier.

The detail parameter setting for genetic algorithm is as the following: population size 20, crossover rate 0.6, mutation rate 0.05, single-point crossover, roulette wheel selection, and elitism replacement, we set $n_c = 12$, $n_\gamma = 15$, the value of n_f depends on the experimental datasets stated in **Table 2**. According to the fitness function and the number of selected features, we can compare different methods. The GA-FSSVM and PSO-FSSVM related parameters is described in **Table 3**.

The termination criterion is that the generation number reaches generation 100. The best chromosome is obtained when the termination criteria satisfy. Taking the German dataset, for example, over accuracy, number of selected features for each fold using GA-FSSVM approach, PSO-FSSVM approach and Grid algorithm are shown in **Table 1**. For GA-SVM approach, its average accuracy is 87.08%, and average number of features is 11.46, and for PSO-SVM approach, the average accuracy is 85.47% and average number of features is 10.92, but for Grid algorithm, its average accuracy is 81.46%, and all 24 features are used.

Table 1. Experimental result for German dataset using GA-based, PSO-based and grid algorithm approach.

Fold #	GA-FSSVM		PSO-FSSVM		Grid algorithm
	overall accuracy	selected features	overall accuracy [%]	selected features	overall accuracy [%]
1	85	12	91	9	79
2	86	8	90	12	80
3	87	14	87	10	78
4	88	11	86	11	81
5	87	10	83	14	84
6	84	9	84	13	80
7	86	14	87	14	82
8	88	14	86	10	82
9	87	13	83	10	86
10	89	10	88	7	82
11	91	13	81	11	81
12	85	13	82	11	84
13	89	8	83	10	80
Average	87.076923	11.46153	85.461538	10.923076	81.461538

Table 2. Experimental results for test dataset.

Names	GA-FSSVM		PSO-FSSVM		Grid algorithm
	overall accuracy	selected eatures	overall accuracy [%]	selected features	overall accuracy [%]
Australian	88.2	5.87	91.34	6.23	87.14
Heart disease	92.05	7.53	95.12	5.76	85.47
Vehicle	88.43	9.34	93.02	11.5	83.33
Sonar	96.26	18.23	98.24	16.25	95.19
breast cancer	95.87	1.39	98.9	1.2	94.67

Table 3. Parameters setting.

PSO-FSSVM		GA-FSSVM		Grid Algorithm	
Parameter	value	Parameter	Value	Parameter	Value
Population size	20	Population size	20	C	$0\ldots2^{15}$
Number of generations	100	Number of generations	100	γ	$2^{-15}\ldots1$
Vmax	4	Probability of crossover	0.6		
C_1,C_2	2	Probability of mutation	0.05		

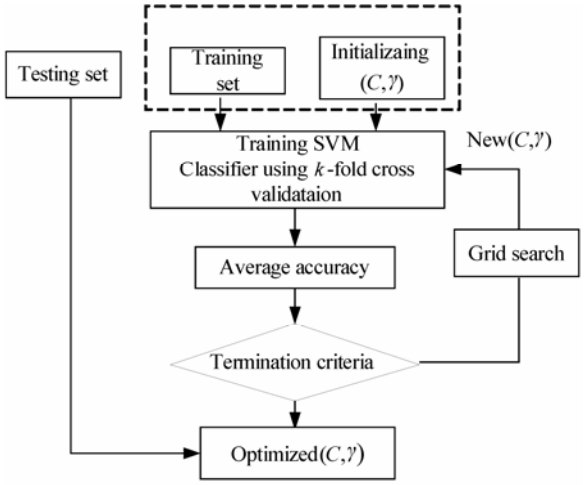


Figure 6. Parameters setting using grid algorithm.

To compare the two proposed evolutionary computing approaches with the Grid algorithm, as shown in **Table 1**. Generally, compared with the Grid algorithm, the two proposed approaches have good accuracy performance with fewer features.

4.2. Hyperspectral Dataset

4.2.1. Dataset Description

To validate the proposed intelligent optimization methods, we also compare the two evolutionary computing methods with traditional classification method for hyperspectral data classification. The classifier is used by SVM.

The hyperspectral remote sensing image used in our experiments is a section of a scene taken over northwest Indiana's Indian Pines by the AVIRIS sensor in 1992 [19]. From the 220 spectral channels acquired by the AVIRIS sensor, 20 channels are discarded because affected by atmospheric problems. From the 16 different land-cover classes available in the original ground truth, seven classes are removed, since only few training samples were available for them (this makes the experimental analysis more significant from the statistical viewpoint). The remaining nine land-cover classes were used to generate a set of 4757 training samples (used for learning the classifiers) and a set of 4588 test samples (exploited for assessing their accuracies) (see **Table 4**). The origin image and reference image are shown in **Figure 7.**

4.2.2. Experiment Settings

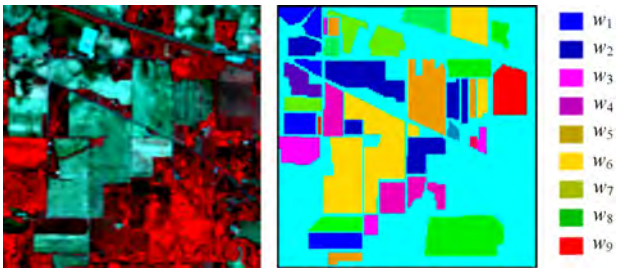
In the experiments, when using the proposed intelligent optimization methods, we considered the nonlinear SVM based on the popular Gaussian kernel (referred to as SVM-RBF). The related parameters C and γ for this kernel were varied in the arbitrarily fixed ranges $[10^{-3}, 300]$ and $[10^{-3}, 3]$, so as to cover high and small regularization of the classification model, and fat as well as thin kernels, respectively. The experiments are implemented by LIBSVM [20].

LIBSVM is widely used in SVM classifier, but the value of RBF kernel parameters is always difficult to define. The default values are as follows: C is 1, and γ is the reciprocal of the dimension. In our experiment, the dimension is the band number, so the parameter value of γ is 0.005.In the same way, the default value of C of SVM parameter in ENVI is 100, and γ is the reciprocal of the dimension. In our experiment, the dimension is the band number, so the parameter value of γ is 0.005. In addition, we also select SVM parameters by grid algorithm. In grid algorithm, according to reference [4], the range of C and γ is $[2^{-5}, 2^{15}]$ and $[2^{-15}, 2^{3}]$, the step length is 2^{2}.

Concerning the PSO algorithm, we considered the following standard parameters: swarm size S = 20, inertia weight w = 1, acceleration constants c_1 and c_2 equal to 2, and maximum number of iterations fixed at 300. The parameters setting is summarized in **Table 5**.

Table 4. Number of training and test samples.

CLASS	TRAINNING	TEST	TOTAL
ω_1-Corn-no till	742	692	1434
ω_2-Corn-min till	442	392	834
ω_3-Grass/Pasture	260	237	497
ω_4-Grass/Trees	389	358	747
ω_5-Hay-windrowed	236	253	489
ω_6-Soybean-no till	487	481	968
ω_7-Soybean-min till	1245	1223	2468
ω_8-Soybean-clean till	305	309	614
ω_9-Woods	651	643	1294
Total	4757	4588	9345



(a) (b)

Figure 7. Hyperspectral image data. (a) Origin image; (b) Ground truth image.

In addition, for comparison purpose, we implemented the three traditional methods and our two intelligent optimization methods for classification. The experimental comparison results are shown in **Figure 8** and **Table 6**.

We apply the PSO-FSSVM classifier to the available training data. Note that each particle of the swarm was defined by position and velocity vectors of a dimension of 202. At convergence of the optimization process, we assessed the PSO-FSSVM-RBF classifier accuracy on the test samples. The achieved overall accuracy is 95.25% corresponding to substantial accuracy gains as compared to what is yielded either by the SVM classifier (with the Gaussian kernel) based grid algorithm applied to all available features (+4.27%) or by the GA-FSSVM classifier (+2.36) with 101 features. Moreover, by means of this approach, the average subset feature number is 120, which is fewer than original feature number 220. The whole process is implemented automatically and without user's interface.

Table 6 illustrates the classification of different classifiers. As can be seen our proposed classifier have better accuracy than traditional classifiers. The LIBSVM default setting lead to the lowest accuracy, which is 52.79%.The best percentage of classification, is 95.25% by PSO-FSSVM method. The results still confirm the strong superiority of our proposed PSO-FSSVM over the other classifiers, with a gain in overall accuracy +12.80%, +4.27% and +2.36% with respect to the default SVM classifier in ENVI, the grid algorithm, and the GA-SVM classifiers (see **Table 6**).

From the obtained experimental results, we conclude the proposed PSO-FSSVM classifier has the best classification accuracy account of its superior generalization capability as compared to traditional classification techniques.

5. Discussion and Conclusion

Feature selection is an important issue in the construction of classification system. The number of input features in a classifier should be limited to ensure good predictive power without an excessively computationally intensive model. This work investigated two novel intelligent optimization models that hybridized the two evolutionary computing optimizations and support vector machines to maintain the classification accuracy with small and suitable feature subsets. The work is novel, since few researches have conducted on the GA-FSSVM and PSO-FSSVM classification system to find simultaneously an optimal feature subset and SVM model parameters in high-dimensional data classification.

In this paper, we addressed the problem of the classification of high dimensional data using intelligent optimization methods. This study presents two evolutionary computing optimization approaches: GA-FSSVM and PSO-FSSVM, capable of searching for the optimal parameter values for SVM and a subset of beneficial features. This optimal subset of features and SVM parameters are then adopted in both training and testing to obtain

Table 5. Parameters setting of different methods.

PSO-SVM		GA-SVM		Grid Algorithm	
Parameter	Value	Parameter	Value	Parameter	Value
Swarm size	20	Population size	20	C	$[2^{-5}, 2^{-3},\cdots,2^{15}]$
Number of generations	300	Number of generations	300	γ	$[2^{-15}, 2^{-13},\cdots,2^{3}]$
V_{max}	4	Probability of crossover	0.6		
C_1, C_2	2	Probability of mutation	0.05		
C:0-300; γ:0-3		C:0-300; γ:0-3			

Table 6. Classification result by different parameter selection methods.

Methods of selecting parameters	C	γ	Band num	Classification accuracy (%)
LIBSVM default	1	0.005	200	52.79
ENVI default	100	0.005	200	82.45
10–cross validation (grid search algorithm)	8	2	200	90.98
GA-FSSVM	157.89	0.243	101	92.89
PSO-FSSVM	223.32	0.9696	120	95.25

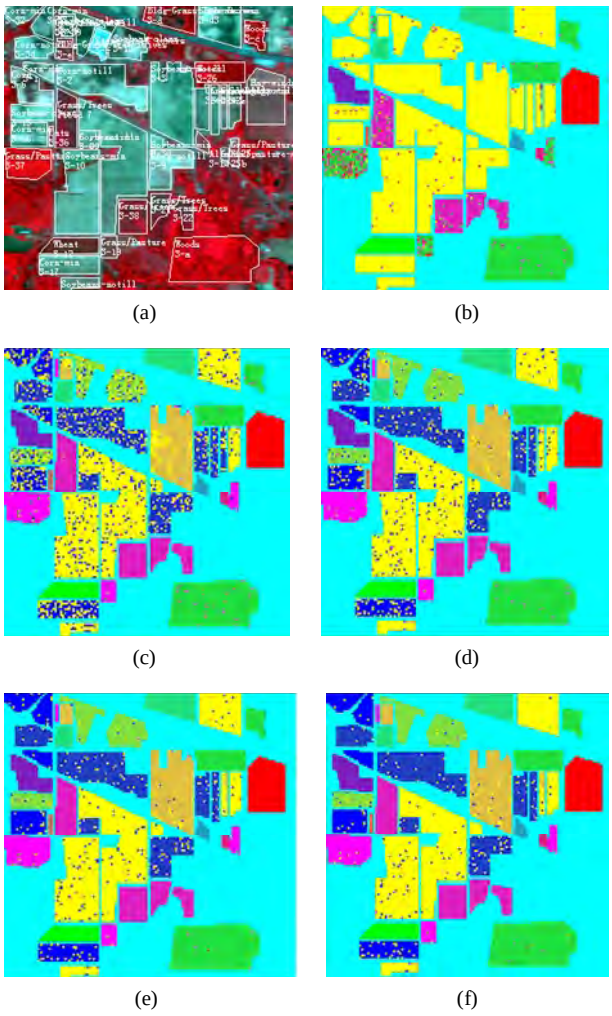


Figure 8. Classification accuracy of different classifiers. (a) Train data and test data; (b) LIBSVM 52.79%; (c) ENVI 82.45%; (d) Grid algorithm 90.98%; (e) GA-FSSVM 92.89%; (f) PSO-FSSVM 95.25%.

the optimal outcomes in classification. Comparison of the obtained results with traditional Grid-based approach demonstrates that the developed PSO-FSSVM and GA-FSSVM approach have better classification accuracy with fewer features. After using feature selection in the experiment, the proposed approaches are applied to eliminate unnecessary or insignificant features, and effectively determine the parameter values, in turn improving the overall classification results, and the PSO-FSSVM approach is better than GA-FSSVM in most datasets in our experiments.

Experimental results concerning a simulated dataset revealed that the proposed approach not only optimized the classifier's model parameters and correctly obtained the discriminating feature subset, but also achieved accurate classification accuracy.

Results of this study are obtained with an RBF kernel function. However, other kernel parameters can also be optimized using the same approach. Experimental results obtained from UCI datasets, other public datasets and real-world problems can be tested in the future to verify and extend this approach.

In the future, we need to further improve the classification accuracy by using other evolutionary optimization algorithm, such as Simulated Annealing Algorithm (SAA), Artificial Immune Algorithm (AIA). In addition, we are to optimize the SVM classifiers by several combinative evolutionary optimization methods in the future.

6. Acknowledgements

The authors would like to thank the anonymous referees for their valuable comments, which helped in improving the quality of the paper. This research was supported by National Natural Science Foundation of China under Grant NO. 60705012.

7. References

[1] V. N. Vapnik, "The Nature of Statistical Learning Theory," Springer Verlag, New York, 2000.

[2] H. Fröhlich and O. Chapelle, "Feature Selection for Support Vector Machines by Means of Genetic Algorithms," *Proceedings of the* 15*th IEEE International Conference on Tools with Artificial Intelligence*, Sacramento, 3-5 November 2003, pp. 142-148.

[3] C. W. Hsu and C. J. Lin, "A Simple Decomposition Method for Support Vector Machine," *Machine Learning*, Vol. 46, No. 3, 2002, pp. 219-314.

[4] H. Liu and H. Motoda, "Feature Selection for Knowledge Discovery and Data Mining," Kluwer Academic, Boston, 1998.

[5] R. C. Chen and C. H. Hsieh, "Web Page Classification Based on a Support Vector Machine Using a Weighed Vote Schema," *Expert Systems with Applications*, Vol. 31, No. 2, 2006, pp. 427-435.

[6] C. Gold, A. Holub and P. Sollich, "Bayesian Approach to Feature Selection and Parameter Tuning for Support Vector Machine Classifiers," *Neural Networks*, Vol. 18, No. 5-6, 2005, pp. 693-701.

[7] R. Kohavi and G. H. John, "Wrappers for Feature Subset Selection," *Artificial Intelligence*, Vol. 97, No. 1-2, 1997, pp. 273-324.

[8] T. Shon, Y. Kim and J. Moon, "A Machine Learning Framework for Network Anomaly Detection Using SVM and GA," *Proceedings of* 3*rd IEEE International Workshop on Information Assurance and Security*, 23-24 March 2005, pp. 176-183.

[9] L. Zhang, L. Jack and A. K. Nandi, "Fault Detection Using Genetic Programming," *Mechanical Systems and Signal Processing*, Vol. 19, No. 2, 2005, pp. 271-289.

[10] B. Samanta, K. R. Al-Balushi and S. A. Al-Araimi, "Artificial Neural Networks and Support Vector Machines

with Genetic Algorithm for Bearing Fault Detection," *Engineering Applications of Artificial Intelligence*, Vol. 16, No. 7-8, 2003, pp. 657-665.

[11] C. L. Huang, M. C. Chen and C. J. Wang, "Credit Scoring with a Data Mining Approach Based on Support Vector Machines," *Expert Systems with Applications*, Vol. 33, No. 4, 2007, pp 847-856.

[12] C. L. Huang and C. L. Wang, "A GA-Based Feature Selection and Parameters Optimization for Support Vector Machines," *Expert Systems with Applications*, Vol. 31, No. 2, 2006, pp. 231-240.

[13] C. W. Hsu, C. C. Chang and C. J. Lin, "A Practical Guide to Support Vector Classification," Technical Report, Department of Computer Science and Information Engineering, University of National Taiwan, Taipei, 2003, pp. 1-12.

[14] P. F. Pai and W. C. Hong, "Support Vector Machines with Simulated Annealing Algorithms in Electricity Load Forecasting," *Energy Conversion and Management*, Vol. 46, No. 17, 2005, pp. 2669-2688.

[15] F. Melgani and L. Bruzzone, "Classification of Hyperspectral Remote Sensing Images with Support Vector Machines," *IEEE Transactions on Geoscience and Remote Sensing*, Vol. 42, No. 8, 2004, pp. 1778-1790.

[16] G. M. Foody and A. A. Mathur, "Relative Evaluation of Multiclass Image Classification by Support Vector Machines," *IEEE Transactions on Geoscience and Remote Sensing*, Vol. 42, No. 6, 2004, pp. 1335-1343.

[17] J. Kennedy and R. C. Eberhart, "Particle Swarm Optimization," *IEEE International Conference on Neural Networks*, IEEE Neural Networks Society, Perth, 27 November-1 December 1995, pp. 1942-1948.

[18] S. Hettich, C. L. Blake and C. J. Merz, "UCI Repository of Machine Learning Databases," Department of Information and Computer Science, University of California, Irvine, 1998. http//www.ics.uci.edu/~mlearn/MLRepository.html

[19] "Aviris Indiana's IndianPinesl DataSet." ftp://ftp.ecn.Purdue.edu/biehl/MultiSpec/92AV3C.lan; ftp://ftp.ecn.purdue.edu/biehl/PCMultiSpeeThyFiles.zip

[20] C. C. Chang and C. J. Lin, "LIBSVM: A Library for Support Vector Machines," 2005. http://www.csie.ntu.edu.tw/~cjlin/libsvm

Intelligent Information Management, **2010, 2, 365-374**
doi:10.4236/iim.2010.26044 Published Online June 2010 (http://www.SciRP.org/journal/iim)

Steady-State Queue Length Analysis of a Batch Arrival Queue under N-Policy with Single Vacation and Setup Times

Zhong Yu[1], Mingwu Liu[2], Yongkai Ma[1]
[1]*School of Management and Economics, University of Electronic Science and Technology of China, Chengdu, China*
[2]*School of Management, Chongqing Jiaotong University, Chongqing, China*
E-mail: *liumingwu*2007@*yahoo.cn*
Received March 10, 2010; *revised April* 20, 2010; *accepted May* 23, 2010

Abstract

This paper investigates the steady state property of queue length for a batch arrival queue under N-policy with single vacation and setup times. When the system becomes empty, the server is turned off at once and takes a single vacation of random length V. When he returns, if the queue length reaches or exceeds threshold $N(N \geq 1)$, the server is immediately turned on but is temporarily unavailable due to a random setup time U before offering service. If not, the server stays in the system until the queue length at least being N. We derive the system size distribution and confirm the stochastic decomposition property. We also derive the recursion expressions of queue length distribution and other performance measures. Finally, we present some numerical examples to show the analytical results obtained. Sensitivity analysis is also performed.

Keywords: Queue Length, Recursion Expressions, N Policy, Setup

1. Introduction

This paper pays attention to a batch arrival queueing system under N-policy with a single vacation and setup times, which can model queue-like manufacturing/production/inventory system. Consider a system of processing in which the operation does not start until some predetermined number (*N*) of semi-finished products waiting for processing. To be more realistic, the machine need a setup time for some preparatory work before starting processing. When all the semi-finished products in the system are processed, the machine is shut down and leaves for a vacation. The operator performs machine repair, preventive maintenance and some other jobs during the vacation. After these extra operations, the operator returns and checks the number of semi-finished products in the queue determining whether or not start the machine.

Queueing system with vacations has been attracted considerable attention to many authors. It has effectively been applied in computers and communication systems, production/inventory system. Doshi [1] and Takagi [2] presented an excellent survey of queueing system with server vacations. One of the important achievements for vacation queueing system is the famous stochastic decomposition results, which was first established by Fuhrmann and Cooper [3].

The N-policy was first introduced by Yadin and Naor [4], which is a control policy turning the server on whenever N (a predetermined value) or more customers in the system, turning off the server when system is empty. Lee *et al*. [5] successfully combined the batch arrival queue with N-policy and obtained the analytical solutions. Later, Lee *et al*. [6,7] analyzed in detail a batch arrival $M^x/G/1$ queue under N-policy with a single vacation and repeated vacation respectively. They derived the system size distribution which confirmed the famous stochastic decomposition property, and the optimal stationary operating policy was also investigated. At the present day, batch arrival queueing system under N-policy with different vacation policies have been received considerable attention because of its practical implication in production/inventory system. A number of searchers, such as Choudhury *et al*. [8-10], Ke [11,12], and Reddy *et al*. [13], and many authors not be listed

The authors also thank the National Nature Science Foundation of China for its support (contract: 70672104).

above have considered batch arrival queueing system under N-policy with various vacation policies.

It is to be noted that few authors involved above considered the probability distribution of the number of customers in the system for batch arrival queue under N-policy with different vacations policies. Recently, Wang *et al*. [14] analyzed the behaviors of queue length distributions of a batch arrival queue with server vacations and breakdowns based on a maximum entropy approach [15,16]. Ke *et al*. [17] also used the maximum entropy solutions for batch arrival queue with an un-reliable server and delaying vacations. They all derived the approximate formula for the probability distribution of the number of customers in the system. Tang and his co-author [18,19] paid attention to the queue length distribution for queueing system, because it is an important performance measure for the system design and optimization. For example, the queue-length distribution has been applied for the communication system buffer design [20].

To the best of our knowledge, fewer researchers have derived the queue length distribution under N-policy batch arrival queueing system. Although, the batch arrival under N-policy have been studied extensively, but the exact analytical solutions for queue length distribution do not be obtained. This motives us to develop an approach for the queue length distribution of the N-policy batch arrival queueing system. In this paper, we study the steady state queue length for an $M^x/G/1$ under N policy with single vacation and setup times. First, we derive the system size distribution by the supplementary variables method. Second, using the Leibniz formula of derivation combing some knowing results, we obtain the additional queue length distribution and the recursion expressions of the queue length distribution. Finally, we present several examples for application of these recursion expressions. Also, the effect of different system parameters on the queue length distribution is investigated.

2. The Mathematical Model and Notations

This paper considers an $M^x/G/1$ queueing system where the arrival occurs according to a compound Poisson process with random batch size X. Arriving customers in the queue form a single waiting line and the service discipline is assumed to be FCFS. The service time is an independent and identically distributed random variable with a general distribution function $S(t), t \geq 0$, and with finite mean service time. The server can only process one customer at a time. The server is turned off each time if there is no customer in the queue and leaves for a vacation of random length V. When he returns from the vacation and finds the queue length is no less than N, the server starts to setup with random length U. Otherwise, he stays in the system and does not start setup until the customers reaches and exceeds N. When the setup is completed, the server begins to serve the customers until there is no customer in the system.

Throughout the analysis, the following notations and the variables will be adopted.

N = threshold ($N \geq 1$)

λ = mean arrival rate

μ = mean service rate

$E[V]$ = mean vacation time

$E[U]$ = mean setup time

S = service time random variable

V = vacation times random variable

U = setup times random variable

$s(x)$ = the probability density function S

$v(x)$ = the probability density function V

$u(x)$ = the probability density function U

$S^*(\theta)$ = the laplace-Stieltjies transform (LST) of S

$V^*(\theta)$ = the LST of V

$\hat{U}^*(\theta)$ = the LST of U

$S^0(t)$ = remaining service time of the customer in service at time t

$V^0(t)$ = remaining vacation time of the customer in vacation at time t

$U^0(t)$ = remaining setup time of the customer in setup at time t

$N(t) = j$, the number of customers in the queue at time t

α_k = probability that k customers arrival during a vacation

$g_k = P(X = k)$, where $k = 1, 2, \cdots$

$X(z) = \sum_{k=1}^{\infty} g_k z^k$, the p.g.f. of each batch size

$Y(t) = 0$ if the server is on vacation

$= 1$ if the server is in dormancy

$= 2$ if the server is in setup

$= 3$ if the server is busy

ρ = traffic intensity, $\rho = \lambda E[X]/\mu$. In the steady state ρ should be assumed to be less than unity

$m[k] = m_1 + \cdots + m_k$, the number of customers presents the sum of k batches of customers

$\sum_{m[k]=j-1} = \sum_{m_1=1}^{i_1} \cdots \sum_{m_k=1}^{i_k}$ *and* $i_1 + \cdots + i_k = j-1$, the sum of k batches of customer equals to $j-1$.

3. Preliminary Formula for $M^x/G/1$ Queueing System

Before discussing our model, let us recall some results in the ordinary $M^x/G/1$ queueing system. Let $\tilde{p}_j^0 = \lim_{t\to\infty} P\cdot\{N(t)=j\}, j\geq 0$ be the steady state queue length distribution for the $M^x/G/1$ queueing system. From Tang *et al*. [19], the recursion expressions of queue-length distribution are as follows:

$$\tilde{p}_0^0 = 1-\rho \tag{1}$$

$$\tilde{p}_j^0 = \lambda(1-\rho)\left\{\theta_j + \sum_{s=1}^{j-1}\theta_{j-s}\left[1-\sum_{i=1}^{s} g_i\right]\right\}, j\geq 1 \tag{2}$$

where

$$\theta_1 = \frac{1-s(\lambda)}{\lambda s(\lambda)};\ \ s(\lambda) = \int_0^\infty e^{-\lambda t} dS(t);$$

$$\theta_j = \frac{1}{s(\lambda)}\left\{\sum_{i=1}^{j-1}\sum_{m[i]=j-1} g_{m_1}\cdots g_{m_i}\int_0^\infty e^{-\lambda t}\left[1-S(t)\right]\frac{(\lambda t)^i}{i!}dt + \sum_{i=1}^{j-1}\theta_{j-1}\left[1-s(\lambda)-\sum_{i=1}^{i}\sum_{m[k]=k} g_{m_1}\cdots g_{m_k}\int_0^\infty e^{-\lambda t}\frac{(\lambda t)^k}{k!}dS(t)\right]\right\}$$

$j\geq 1$.

Let $G(z)$ be the p.g.f. of the number of customers in the queue at stationary. We have

$$G(z) = \frac{(1-\rho)(1-z)S^*(\lambda-\lambda X(z))}{S^*(\lambda-\lambda X(z))-z} \tag{3}$$

4. The System Size Distribution

This section, we set up the system equation for the system size distribution at stationary and derive the p.g.f. of the system size distribution. We introduce the supplementary variables $S^0(t)$, $V^0(t)$ and $U^0(t)$ for obtaining a Markov process $\{N(t),\delta(t)\}$, where $\delta(t)=V^0(t)$ if $Y(t)=0$, $\delta(t)=0$, if $Y(t)=1$, $\delta(t)=U^0(t)$, if $Y(t)=2$ and $\delta(t)=S^0(t)$, if $Y(t)=3$. Denote

$$Q_n(x,t)dt \equiv P\left[N(t)=n, x\leq V^0(t)\leq x+dt, Y(t)=0\right] \quad n=0,1,\cdots$$

$$R_n(t) \equiv P\left[N(t)=n, Y(t)=1\right], n=0,\cdots,N-1$$

$$U_n(x,t)dt \equiv P\left[N(t)=n, x\leq U^0(t)\leq x+dt, Y(t)=2\right] \quad n=0,1,\cdots$$

$$P_n(x,t) \equiv P\left[N(t)=n, x\leq S^0(t)\leq x+dt, Y(t)=3\right] \quad n=0,1,\cdots$$

Following the argument of Lee *et al*. [5-7], we can easily set up the following steady state system equations the supplementary variables technique.

$$-\frac{d}{dx}P_1(x) = -\lambda P_1(x) + P_2(0)s(x) \tag{4}$$

$$-\frac{d}{dx}P_n(x) = -\lambda P_n(x) + P_{n+1}(0)s(x) + \lambda\sum_{k=1}^{n-1}P_{n-k}(x)g_k, \quad n=2,\cdots,N-1 \tag{5}$$

$$-\frac{d}{dx}P_n(x) = -\lambda P_n(x) + P_{n+1}(0)s(x) + \lambda\sum_{k=1}^{n-1}P_{n-k}(x)g_k + U_n(0)s(x), n\geq N \tag{6}$$

$$-\frac{d}{dx}Q_0(x) = -\lambda Q_0(x) + P_1(0)v(x) \tag{7}$$

$$-\frac{d}{dx}Q_n(x) = -\lambda Q_n(x) + \lambda\sum_{k=1}^{n}Q_{n-k}(x)g_k, \quad n=1,2,\cdots \tag{8}$$

$$-\frac{d}{dx}U_N(x) = -\lambda U_N(x) + Q_N(0)u(x) + \lambda u(x)\sum_{k=0}^{N-1}R_k(x)g_{N-k} \tag{9}$$

$$-\frac{d}{dx}U_n(x) = -\lambda U_n(x) + \lambda\sum_{k=1}^{n-N}U_{n-k}(x)g_k + Q_n(0)u(x) + \lambda u(x)\sum_{k=0}^{N-1}R_k g_{n-k}, n=N+1,\cdots \tag{10}$$

$$0 = -\lambda R_0 + Q_0(0) \tag{11}$$

$$0 = -\lambda R_n + Q_n(0) + \lambda\sum_{k=1}^{n}R_{n-k}g_k, n=1,\cdots,N-1 \tag{12}$$

Taking the LST of both sides of Equations (4)-(10), we get

$$\theta P_1^*(\theta) - P_1(0) = \lambda P_1^*(\theta) - P_2(0)S^*(\theta) \tag{13}$$

$$\theta P_n^*(\theta)-P_n(0)=\lambda P_n^*(\theta)-P_{n+1}(0)S^*(\theta) -\lambda\sum_{k=1}^{n-1}P_{n-k}^*(\theta)g_k, \quad n=2,\cdots,N-1 \tag{14}$$

$$\theta P_n^*(\theta)-P_n(0)=\lambda P_n^*(\theta)-P_{n+1}(0)S^*(\theta) -\lambda\sum_{k=1}^{n-1}P_{n-k}^*(\theta)g_k-U_n(0)S^*(\theta), \quad n\geq N \tag{15}$$

$$\theta Q_0^*(\theta)-Q_0(0)=\lambda Q_0^*(\theta)-P_1(0)V^*(\theta) \tag{16}$$

$$\theta Q_n^*(\theta)-Q_n(0)=\lambda Q_n^*(\theta)-\lambda\sum_{k=1}^{n}Q_{n-k}^*(\theta)g_k, \quad n=1,2,\cdots \tag{17}$$

$$\theta U_N^*(\theta)-U_N(0)=\lambda U_N^*(\theta)-Q_N(0)\hat{U}^*(\theta) -\lambda\hat{U}^*(\theta)\sum_{k=0}^{N-1}R_k g_{N-k} \tag{18}$$

$$\theta U_n^*(\theta)-U_n(0)=\lambda U_n^*(\theta)-\lambda\sum_{k=1}^{n-N}U_{n-k}^*(\theta)g_k -Q_n(0)\hat{U}^*(\theta)-\lambda\hat{U}^*(\theta)\sum_{k=0}^{N-1}R_k g_{n-k}, \quad n=N+1,\cdots \tag{19}$$

Noted that the LST of $P_n(x)$, $Q_n(x)$ and $U_n(x)$ is defined as follows:

$$P_n^*(\theta)=\int_0^\infty e^{-\theta x}P_n(x)dx,\quad Q_n^*(\theta)=\int_0^\infty e^{-\theta x}Q_n(x)dx,\quad U_n^*(\theta)=\int_0^\infty e^{-\theta x}U_n(x)dx.$$

Now, we define the following p.d.f.

$$P^*(z,\theta)=\sum_{n=1}^{\infty}P_n^*(\theta)z^n,\quad P(z,0)=\sum_{n=1}^{\infty}P_n(0)z^n$$

$$Q^*(z,\theta)=\sum_{n=0}^{\infty}Q_n^*(\theta)z^n,\quad Q(z,0)=\sum_{n=0}^{\infty}Q_n(0)z^n$$

$$U^*(z,\theta)=\sum_{n=N}^{\infty}U_n^*(\theta)z^n,\quad Q(z,0)=\sum_{n=0}^{\infty}Q_n(0)z^n$$

$$R(z)=\sum_{n=0}^{N-1}R_n z^n.$$

From Equations (13)-(15), we have

$$\left[\theta-\lambda+\lambda X(z)\right]P^*(z,\theta)=P(z,0) -S^*(\theta)\left[\frac{P(z,0)}{z}-P_1(0)\right]-S^*(\theta)U(z,0) \tag{20}$$

Letting $\theta=\lambda-\lambda X(z)$, we get

$$P(z,0)=\frac{zS^*\left[\lambda-\lambda X(z)\right]\left[U(z,0)-P_1(0)\right]}{z-S^*\left[\lambda-\lambda X(z)\right]} \tag{21}$$

Substituting (21) into Equation (20), we have

$$P^*(z,\theta)=\frac{z\left\{S^*\left[\lambda-\lambda X(z)\right]-S^*(\theta)\right\}\left[U(z,0)-P_1(0)\right]}{\left\{z-S^*\left[\lambda-\lambda X(z)\right]\right\}\left[\theta-\lambda+\lambda X(z)\right]} \tag{22}$$

Similarly, from (16), (17) and (18), (19) respectively and combining (11) and (12), we have

$$Q(z,0)=P_1(0)V^*\left[\lambda-\lambda X(z)\right] \tag{23}$$

$$Q^*(z,\theta)=\frac{P_1(0)\left\{V^*\left[\lambda-\lambda X(z)\right]-V^*(\theta)\right\}}{\theta-\lambda+\lambda X(z)} \tag{24}$$

$$U(z,0)=\hat{U}^*\left[\lambda-\lambda X(z)\right]\cdot\left[Q(z,0)+\lambda\left[X(z)-1\right]R(z)\right] \tag{25}$$

$$U^*(z,\theta)=\left\{\hat{U}^*\left[\lambda-\lambda X(z)\right]-U^*(\theta)\right\}\times\frac{\left[Q(z,0)+\lambda\left[X(z)-1\right]R(z)\right]}{\theta-\lambda+\lambda X(z)} \tag{26}$$

Let $P(z)$ be the p.g.f. of the queue size at an arbitrary time epoch. Then,

$$P(z)=P^*(z,0)+Q^*(z,0)+U^*(z,0)+R(z) \tag{27}$$

Using Equations (21)-(26) in $P(z)$, we have

$$P(z)=\frac{(z-1)S^*\left[\lambda-\lambda X(z)\right]}{z-S^*\left[\lambda-\lambda X(z)\right]}\times\left\{\frac{1-V^*\left[\lambda-\lambda X(z)\right]\hat{\hat{U}}^*\left[\lambda-\lambda X(z)\right]}{\lambda-\lambda X(z)}P_1(0) +\hat{U}^*\left[\lambda-\lambda X(z)\right]R(z)\right\} \tag{28}$$

Following Lee *et al.* [7], we have

$$R(z)=\frac{1}{\lambda}P_1(0)\sum_{n=0}^{N-1}\Psi_n z^n \tag{29}$$

where $\Psi_0=\alpha_0=V^*(\lambda)$, $\Psi_n=\sum_{i=0}^{n}\alpha_i\pi_{n-i}, n\geq 1$, $\pi_0=1$, and $\pi_n=\sum_{i=1}^{n}g_i\pi_{n-i}, n\geq 1$. π_n is the probability that system state visit n during an idle period in the M^X/G/1/N-policy queue [5].

Taking (29) into (28), we have

$$P(z)=\frac{(z-1)S^*[\lambda-\lambda X(z)]}{z-S^*[\lambda-\lambda X(z)]}P_1(0)\times\left\{\frac{1-V^*[\lambda-\lambda X(z)]\hat{\hat{U}}^*[\lambda-\lambda X(z)]}{\lambda-\lambda X(z)}+\frac{\hat{U}^*[\lambda-\lambda X(z)]\sum_{n=0}^{N-1}\Psi_n z^n}{\lambda}\right\} \quad (30)$$

From Equation (30) and $P(1)=1$, we get

$$P_1(0)=\frac{\lambda(1-\rho)}{\lambda(E[U]+E[V])+\sum_{n=0}^{N-1}\Psi_n} \quad (31)$$

Thus, the p.d.f. of the system size distribution in the steady state becomes

$$P(z)=G(z)\cdot\phi(z) \quad (32)$$

where

$$\phi(z)=\frac{1}{\lambda(E[U]+E[V])+\sum_{n=0}^{N-1}\Psi_n}\times\left\{\frac{1-V^*[\lambda-\lambda X(z)]\hat{\hat{U}}^*[\lambda-\lambda X(z)]}{1-X(z)}+\hat{U}^*[\lambda-\lambda X(z)]\sum_{n=0}^{N-1}\Psi_n z^n\right\}$$

Remark 4.1

Note that for $P(U=0)=1$, $E[U]=0$ and $\hat{\hat{U}}^*[\lambda-\lambda X(z)]=1$, then the Equation (32) agrees with Equation (25) of Lee *et al*. [7].

Remark 4.2

Let $N=1$, our model can be reduced to the batch arrival queue under a single vacation policy with startup. In this case the result (32) coincides with Equation (30) of Ke [21], in which the closedown time is assumed to be zero.

5. The Queue Length Distribution

5.1. The Additional Queue Length Distribution

This section, we will derive the additional queue length distribution. From (32), we see that the stationary queue length distribution of the M^x/G/1queue under N policy with a single vacation and setup times decomposes into two independent random variables: one is the stationary queue length distribution of the ordinary M^x/G/1 queue; another is the additional queue length (L^a) distribution due to N policy with a single vacation and setup times.

From the definition of p.d.f., the additional queue length distribution p_j^a is given by

$$p_j^a=P(L^a=j)=\frac{1}{j!}\frac{d^j}{dz^j}\phi(z)|_{z=0} \quad (33)$$

Obviously, the probability of additional queue length being zero is as follows

$$p_0^a=\phi(z)|_{z=0}=\frac{1}{\lambda(E[U]+E[V])+\sum_{n=0}^{N-1}\Psi_n} \quad (34)$$

When $j=1,\cdots,N-1$, we can get the probability distribution of the additional queue length $L^a=j$ by direct derivative of $\phi(z)$.

$$p_j^a=\frac{1}{\lambda(E[U]+E[V])+\sum_{n=0}^{N-1}\Psi_n}\frac{1}{j!}\frac{d^j}{dz^j}\times\left\{\frac{1-V^*[\lambda-\lambda X(z)]\hat{\hat{U}}^*[\lambda-\lambda X(z)]}{1-X(z)}+\hat{U}^*[\lambda-\lambda X(z)]\sum_{n=0}^{N-1}\Psi_n z^n\right\}|_{z=0} \quad (35)$$

Following Leibniz formula, we have

$$p_j^a=\frac{1}{\lambda(E[U]+E[V])+\sum_{n=0}^{N-1}\Psi_n}\frac{1}{j!}\sum_{k=0}^{j}\binom{j}{k}\left\{\frac{d^k}{dz^k}\left(\frac{1}{1-X(z)}\right)\times\frac{d^{j-k}}{dz^{j-k}}\left(1-V^*[\lambda-\lambda X(z)]\hat{\hat{U}}^*[\lambda-\lambda X(z)]\right)+\frac{d^k}{dz^k}\hat{U}^*[\lambda-\lambda X(z)]\frac{d^{j-k}}{dz^{j-k}}\left(\sum_{n=0}^{N-1}\Psi_n z^n\right)\right\}|_{z=0},\quad j=1,\cdots,N-1 \quad (36)$$

First, let us define the following functions $f_x(z)=\frac{1}{1-X(z)}$, $f_V(z)=V^*[\lambda-\lambda X(z)]$ and $f_{\hat{U}}(z)=\hat{U}^*\cdot[\lambda-\lambda X(z)]$. For sake of convenience, we note $\frac{d^k}{dz^k}f_X(z)|_{z=0}=f_x^{(k)}(0)$. It is to be noted that the following recursion expressions hold.

$$\frac{f_x^{(n)}(0)}{n!}=\sum_{k=0}^{n-1}\frac{f_x^{(k)}(0)}{k!}\cdot g_{n-k},\quad n\ge 1 \tag{37}$$

$$\frac{f_{\hat{U}}^{(n)}(0)}{n!}=\int_0^{\infty}e^{-\lambda t}\frac{y^{(n)}(0,t)}{n!}dU(t),\quad n\ge 1 \tag{38}$$

$$\frac{f_V^{(n)}(0)}{n!}=\int_0^{\infty}e^{-\lambda t}\frac{y^{(n)}(0,t)}{n!}dV(t),\quad n\ge 1 \tag{39}$$

where

$$\frac{y^{(n)}(0,t)}{n!}=\lambda t\sum_{k=0}^{n-1}\frac{k+1}{n}\cdot g_{k+1}\cdot\frac{y^{(n-k-1)}(0,t)}{(n-k-1)!},$$

$$y(z,t)=e^{\lambda tX(z)}.$$

Substitute (37)-(39) into (36), after some manipulations, we have

$$p_j^a=\frac{\sum_{k=0}^{j-1}-\frac{f_X^{(k)}(0)}{k!}\sum_{s=0}^{j-k}\frac{f_V^{(s)}(0)}{s!}\frac{f_{\hat{U}}^{(j-k-s)}(0)}{(j-k-s)!}}{\lambda\left(E[U]+E[V]\right)+\sum_{n=0}^{N-1}\Psi_n}$$
$$+\frac{\frac{f_X^{(j)}(0)}{j!}\left(1-V^*(\lambda)\hat{U}^*(\lambda)\right)+\sum_{k=0}^{j}\frac{f_{\hat{U}}^{(k)}(0)}{k!}\Psi_{j-k}}{\lambda\left(E[U]+E[V]\right)+\sum_{n=0}^{N-1}\Psi_n} \tag{40}$$

$j=1,\cdots,N-1$

When $j=N,N+1,\cdots$, in the similar manner proceeding with (36), we get the following additional queue length distribution.

$$p_j^a=\frac{\sum_{k=0}^{j-1}-\frac{f_X^{(k)}(0)}{k!}\sum_{s=0}^{j-k}\frac{f_V^{(s)}(0)}{s!}\frac{f_{\hat{U}}^{(j-k-s)}(0)}{(j-k-s)!}}{\lambda\left(E[U]+E[V]\right)+\sum_{n=0}^{N-1}\Psi_n}$$
$$+\frac{\frac{f_X^{(j)}(0)}{j!}\left(1-V^*(\lambda)\hat{U}^*(\lambda)\right)+\sum_{k=0}^{N-1}\Psi_k\frac{f_{\hat{U}}^{(j-k)}(0)}{(j-k)!}}{\lambda\left(E[U]+E[V]\right)+\sum_{n=0}^{N-1}\Psi_n} \tag{41}$$

$j=N,N+1,\cdots$

With a direct method, we have derived the additional queue length distribution presented by (34), (40) and (41). Also, following the stochastic decomposition (32), the analytical queue length distribution can be derived.

5.2. The Queue Length Distribution in Equilibrium

Let $p_j=P\{L=j\}$ be the steady state probability distribution of queue length (L) for the M^x/G/1 queueing system under N-policy with a single vacation and setup times. From (32), (1) and (34), we have

$$p_0=P(z)|_{z=0}=(1-\rho)\frac{1}{\lambda\left(E[U]+E[V]\right)+\sum_{n=0}^{N-1}\Psi_n} \tag{42}$$

When $j=1,\cdots,N-1$, the probability of queue length $L=j$ is given by

$$p_j=P(L=j)=\frac{1}{j!}\sum_{k=0}^{j}\binom{j}{k}\frac{d^kG(z)}{dz^k}\cdot\frac{d^{j-k}\phi(z)}{dz^{j-k}}|_{z=0}$$
$$=\sum_{k=0}^{j}p_k^0\cdot p_{j-k}^a \tag{43}$$

where p_{j-k}^a is given by (40).

When $j=N,N+1,\cdots$, we first analyze the following two cases:

1) If $k\le j-N$, that is $j-k\ge N$, p_{j-k}^a is given by (41).

2) If $k\ge j-N+1$, that is $j-k\le N-1$, p_{j-k}^a is given by (40).

So, the probability of queue length $L=j(j\ge N)$ is given by

$$p_j=\sum_{k=0}^{j}p_k^0\cdot p_{j-k}^a=\sum_{k=0}^{j-N}p_k^0\cdot p_{j-k}^a+\sum_{k=j-N+1}^{j}p_k^0\cdot p_{j-k}^a \tag{44}$$

The recursion expressions (42), (43) and (44) present the steady state queue length distribution for the M^x/G/1 queueing system under N-policy with a single vacation and setup times. We can see that linking θ_j with (37)-(39) the queue length distribution could be calculated by these recursion expressions.

6. Numerical Experiments

This section we try to illustrate the application of these recursion expressions by taking several numerical examples. Here we assume that the batch size distribution is displaced geometric distribution *i.e.* $g_k=(1-p)p^{k-1}$, $k\ge 1$. From (37), we have $f_x^{(n)}(0)/n!=1-p,n\ge 1$, which can be derived by the mathematical induction. The service time distributions are assumed to follow the exponential distribution with mean $E[S]=1/\mu$. Furthermore, we take the vacation time distributions and setup time distributions are 3-stage Erlang distribution with finite mean $E[V]=3/v$ and 4-stage Erlang distribution $E[U]=4/u$ respectively. For the sake of convenience, we take $p=0.5,\lambda=2$ and $\mu=10$. We will investigate the effects of N,v and u on the queue length distribution with constant other system parameters.

We present the results of the queue length distribution in **Tables 1-3** with different N, u and v. Observing the **Table 1**, it is clear that: 1) the probability of the system being empty (P_0) decreases as N increases; 2) when N is constant, the probability increases from P_1 to P_3 and then decreases and converges to zero.

Table 1. Queue length distribution vs the threshold *N*.

$u = 10, v = 12$						
N	4	6	8	10	12	14
P_0	0.1944	0.1504	0.1227	0.1036	0.0897	0.0791
P_1	0.1469	0.1137	0.0928	0.0783	0.0678	0.0598
P_2	0.1515	0.1172	0.0956	0.0808	0.0699	0.0616
P_3	0.1526	0.1181	0.0964	0.0814	0.0704	0.0621
P_4	0.0960	0.1180	0.0963	0.0813	0.0704	0.0620
P_5	0.0720	0.1175	0.0958	0.0809	0.0701	0.0618
P_6	0.0531	0.0732	0.0953	0.0805	0.0696	0.0614
P_7	0.0387	0.0544	0.0946	0.0799	0.0692	0.0610
P_8	0.0279	0.0398	0.0586	0.0795	0.0688	0.0606
P_9	0.0200	0.0288	0.0433	0.0791	0.0684	0.0603
P_{10}	0.0142	0.0206	0.0316	0.0488	0.0681	0.0601
P_{11}	0.0102	0.0148	0.0229	0.0361	0.0680	0.0600
P_{12}	0.0073	0.0106	0.0165	0.0264	0.0419	0.0598
P_{13}	0.0053	0.0075	0.0118	0.0191	0.0310	0.0598
P_{14}	0.0039	0.0054	0.0084	0.0137	0.0227	0.0368
P_{15}	0.0031	0.0039	0.0060	0.0098	0.0164	0.0273
P_{16}	0.0026	0.0029	0.0043	0.0070	0.0118	0.0199
P_{17}	0.0022	0.0023	0.0032	0.0050	0.0085	0.0144
P_{18}	0.0020	0.0019	0.0023	0.0036	0.0061	0.0104
P_{19}	0.0018	0.0017	0.0019	0.0026	0.0044	0.0075
P_{20}	0.0017	0.0015	0.0016	0.0020	0.0031	0.0053
EL	3.3376	4.0702	4.8892	5.7465	6.6072	7.7492
sum	0.2660	0.2693	0.4023	0.4126	0.4192	0.4216

Table 2. Queue length distribution vs the parameter *u*.

$N = 6, v = 12$						
u	10	11	12	13	14	15
P_0	0.1504	0.1523	0.1539	0.1552	0.1564	0.1575
P_1	0.1137	0.1157	0.1175	0.1191	0.1204	0.1217
P_2	0.1172	0.1190	0.1205	0.1219	0.1230	0.1240
P_3	0.1181	0.1197	0.1210	0.1221	0.1231	0.1240
P_4	0.1180	0.1194	0.1207	0.1217	0.1226	0.1234
P_5	0.1175	0.1188	0.1200	0.1210	0.1218	0.1226
P_6	0.0732	0.0719	0.0707	0.0696	0.0686	0.0676
P_7	0.0544	0.0529	0.0515	0.0503	0.0492	0.0483
P_8	0.0398	0.0383	0.0371	0.0359	0.035	0.0341
P_9	0.0288	0.0275	0.0264	0.0254	0.0246	0.0239
P_{10}	0.0206	0.0195	0.0186	0.0179	0.0172	0.0166
P_{11}	0.0148	0.0140	0.0133	0.0127	0.0122	0.0117
P_{12}	0.0106	0.0099	0.0094	0.0089	0.0086	0.0082
P_{13}	0.0075	0.0070	0.0067	0.0063	0.0061	0.0058
P_{14}	0.0054	0.0051	0.0048	0.0045	0.0043	0.0042
P_{15}	0.0039	0.0037	0.0035	0.0033	0.0032	0.0030
P_{16}	0.0029	0.0027	0.0026	0.0024	0.0023	0.0023
P_{17}	0.0023	0.0022	0.002	0.0019	0.0019	0.0018
P_{18}	0.0019	0.0018	0.0017	0.0016	0.0016	0.0015
P_{19}	0.0017	0.0016	0.0015	0.0014	0.0013	0.0013
P_{20}	0.0015	0.0014	0.0013	0.0013	0.0012	0.0012
EL	4.0702	3.9953	3.9327	3.8797	3.8343	3.7948
sum	0.2693	0.3783	0.3711	0.3644	0.3591	0.3541

Table 3. Queue length distribution vs the parameter *v*.

$N = 6, u = 10$						
v	10	11	12	13	14	15
P_0	0.1482	0.1494	0.1504	0.1513	0.152	0.1527
P_1	0.1126	0.1132	0.1137	0.1140	0.1142	0.1144
P_2	0.1172	0.1173	0.1172	0.1171	0.117	0.1168
P_3	0.1184	0.1183	0.1181	0.1179	0.1178	0.1176
P_4	0.1182	0.1181	0.1180	0.1179	0.1178	0.1178
P_5	0.1175	0.1175	0.1175	0.1175	0.1175	0.1175
P_6	0.0736	0.0734	0.0732	0.0731	0.0730	0.0729
P_7	0.0547	0.0545	0.0544	0.0542	0.0541	0.0540
P_8	0.0401	0.0399	0.0398	0.0396	0.0396	0.0395
P_9	0.0291	0.0289	0.0288	0.0287	0.0286	0.0285
P_{10}	0.0209	0.0207	0.0206	0.0205	0.0204	0.0204
P_{11}	0.0151	0.0149	0.0148	0.0147	0.0146	0.0146
P_{12}	0.0109	0.0107	0.0106	0.0105	0.0104	0.0103
P_{13}	0.0079	0.0077	0.0075	0.0074	0.0074	0.0073
P_{14}	0.0057	0.0056	0.0054	0.0053	0.0053	0.0052
P_{15}	0.0042	0.0041	0.0039	0.0038	0.0038	0.0037
P_{16}	0.0032	0.0030	0.0029	0.0028	0.0027	0.0027
P_{17}	0.0026	0.0025	0.0023	0.0022	0.0021	0.0021
P_{18}	0.0022	0.0021	0.0019	0.0018	0.0018	0.0017
P_{19}	0.0020	0.0018	0.0017	0.0016	0.0015	0.0014
P_{20}	0.0018	0.0016	0.0015	0.0014	0.0013	0.0013
EL	4.1328	4.0969	4.0702	4.0500	4.0343	4.0220
sum	0.2740	0.2714	0.2693	0.2676	0.2666	0.2656

From **Tables 2-3**, one sees that 1) the probability of the system being empty increases as u or v increases; 2) the probability $P_j\,(1 \le j \le N-1)$ increases as u or v increases, while the probability $P_j\,(j \ge N)$ decreases as u or v increases; 3) the probability P_j increases from P_1 to P_3 and then decreases and converges to zero under constant u or v.

The mean queue length (EL) is also presented in **Tables 1-3** respectively. It is shown that as N increases mean queue length increases, while mean queue length decreases as u or v increases. It is worth mentioning that the mean queue length could not being the only one performance measure for the system design and optimization. For example, when $N=8,\ u=10,\ v=12$, the mean queue length is 4.8892, but the total of the probability (sum) when the queue length exceeding 5 is 0.4023, which could not be neglected.

7. Conclusions

In this paper we have derived the additional queue length distribution and the recursion expressions of the queue length distribution for the M^x/G/1 queueing system under N-policy with a single vacation and setup times. Furthermore, we present the numerical results of the queue length distribution and the distribution properties are also investigated. The results in this paper would be significant and useful to system designers and others. The approach developed in this paper is powerful and can be used to analyze more complex queueing system.

8. References

[1] B. T. Doshi, "Queueing Systems with Vacations–a Survey," *Queueing System*, Vol. 1, No. 1, 1986, pp. 29-66.

[2] H. Takagi, "Queueing Analysis: A Foundation of Performance Evaluation, Vacation and Priority System," Elsevier, Amsterdam, Vol. 1, 1991.

[3] S. W. Fuhrmann and R. B. Cooper, "Stochastic Decompositions in the M/G/1 Queue with Generalized Vacations," *Operations Research*, Vol. 33, No. 5, 1985, pp. 1117-1129.

[4] M. Yadin and P. Naor, "Queueing Systems with a Removable Service Station," *Operational Research Quarterly*, Vol. 14, No. 3, 1963, pp. 393-405.

[5] H. W. Lee, S. S. Lee and K. C. Chae, "Operating Characteristics of M^x/G/1 Queue with N-Policy," *Queueing Systems*, Vol. 15, No. 1-4, 1994, pp. 387-399.

[6] H. W. Lee, S. S. Lee, J. O. Park and K. C. Chae, "Analysis of M^x/G/1 Queue with N Policy and Multiple Vacations," *Journal of Applied Probability*, Vol. 31, No. 2, 1994, pp. 467-496.

[7] S. S. Lee, H. W. Lee, S. H. Yoon and K. C. Chae, "Batch Arrival Queue with N Policy and Single Vacation," *Computers and Operations Research*, Vol. 22, No. 2, 1995, pp. 173-189.

[8] G. Choudhury and M. Paul, "A Batch Arrival Queue with an Additional Service Channel under N-Policy," *Applied Mathematics and Computation*, Vol. 156, No. 1, 2004, pp. 115-130.

[9] G. Choudhury and K. C. Madan, "A Two-Stage Batch Arrival Queueing System with a Modified Bernoulli Schedule Vacation under N-Policy," *Mathematical and Computer Modelling*, Vol. 42, No. 1-2, 2005, pp. 71-85.

[10] G. Choudhury and M. Paul, "A Batch Arrival Queue with a Second Optional Service Channel under N-Policy," *Stochastic Analysis and Applications*, Vol. 24, No. 1, 2006, pp. 1-21.

[11] J.-C. Ke, "The Control Policy of an M^x/G/1 Queueing System with Server Startup and Two Vacation Types," *Mathematical Methods of Operations Research*, Vol. 54, No. 3, 2001, pp. 471-490.

[12] J.-C. Ke, "Optimal Strategy Policy in Batch Arrival Queue with Server Breakdowns and Multiple Vacations," *Mathematical Methods of Operations Research*, Vol. 58, No. 1, 2003, pp. 41-56.

[13] R. G. V. Krishna, R. Nadarajan and R. Arumuganathan, "Analysis of a Bulk Queue with N-Policy Multiple Vacations and Setup Times," *Computers and Operations Research*, Vol. 25, No. 11, 1998, pp. 957-967.

[14] K.-H. Wang, M.-C. Chan and J.-C. Ke, "Maximum Entropy Analysis of the M^x/M/1 Queueing System with Multiple Vacations and Server Breakdowns," *Computers & Industrial Engineering*, Vol. 52, No. 2, 2007, pp. 192-202.

[15] J. E. Shore, "Information Theoretic Approximations for M/G/1 and G/G/1 Queueing Systems," *Acta Informatica*, Vol. 17, No.1, 1982, pp. 43-61.

[16] M. A. El-Affendi and D. D. Kouvatsos, "A Maximum Entropy Analysis of the M/G/1 and G/M/1 Queueing Systems at Equilibrium," *Acta Informatica*, Vol. 19, No. 4, 1983, pp. 339-355.

[17] J.-C. Ke and C.-H. Lin, "Maximum Entropy Solutions for Batch Arrival Queue with an Unreliable Server and Delaying Vacations," *Applied Mathematics and Computation*, Vol. 183, No. 2, 2006, pp. 1328-1340.

[18] Y. H. Tang, "The Transient Solution for M/G/1 Queue with Server Vacations," *Acta Mathematica Scientia*, Vol. 17, No. 3, 1997, pp. 276-282.

[19] Y. H. Tang and X. W. Tang, "The Queue-Length Distribution for M^x/G/1 Queue with Single Server Vacation," *Acta Mathematica Scientia*, Vol. 20, No. 3, 2000, pp. 397-408.

[20] Y. H. Tang, X. Yun and S. J. Huang, "Discrete-Time Geo^x/G/1 Queue with Unreliable Server and Multiple Adaptive Delayed Vacations," *Journal of Computational and Applied Mathematics*, Vol. 220, No. 1-2, 2008, pp. 439-455.

[21] J.-C. Ke, "Batch Arrival Queues under Vacation Policies with Server Breakdowns and Startup/Closedown Times," *Applied Mathematical Modelling*, Vol. 31, No. 7, 2007, pp. 1282-1292.

Intelligent Information Management, 2010, 2, 375-379
doi:10.4236/iim.2010.26045 Published Online June 2010 (http://www.SciRP.org/journal/iim)

Study on Delaunay Triangulation with the Islets Constraints

Dong Wei[1,2], Xinghua Liu[2]
[1]*Computer College, Hefei University of Technology, Hefei, China*
[2]*Information Technology and Engineering College, Shenyang University of Technology, Shenyang, China*
E-mail: *dongweisut@126.com, xinghualiu21mg@163.com*
Received March 16, 2010; *revised April* 18, 2010; *accepted May* 21, 2010

Abstract

Aiming at Delaunay triangulation with islets constrains in terrain simulation. A general Delaunay triangulation algorithm for constrained data set with islets is proposed. The algorithm firstly constructs Constrained Delaunay Triangulation with constraint polygons which are inner boundary of islets, then according to topological relations within edge, surface, arc segment, applies bidirectional search to find the triangle in islet, lastly it carries on certain corresponding processing to complete the Delaunay triangulation algorithm with islets. The analyses show the algorithm simple, fast speed. The algorithm can be used in 3-D terrain vision.

Keywords: Islets Constraints, Bidirectional Search, Delaunay Triangulation

1. Introduction

The triangulation has a wide range of applications in the GIS, geology, computer graphics, virtual reality and so on. Delaunay triangulation has the most performance in the terrain simulation. The build Delaunay triangulation technology taking into account the mandatory constraint data in the establishment of high-quality DEM played a decisive role [1]. At present, based on the points, line segments constrained, the Delaunay triangulation (CDT) has been a lot of algorithms. But the triangulation with the islets constrained data field, due to the complex of data constraint, it is necessary to satisfy the constraints and the characteristics of Delaunay triangulation, so the algorithm with great difficulty [2,3]. The existing triangulation algorithm needs to determine the convex-concave of nature polygon, or using connect bridge technology, the algorithms or more complex, or the result is not Delaunay triangulation [4-8]. This paper proposes an algorithm solution to the islets constrained, the algorithms is simple, without determine the convex-concave of polygon and geometric intersection Computing.

2. With Islets Constrained Triangulation Algorithm

2.1. The Basic Concept of the Islets

Anyone of islet can be abstracted into polygon (convex-concave polygon). The line of Constituted polygonal called arc, polygon composed by arc is Islet, polygon does not contain islet called a simple polygon, is single-connected region, the polygon with islet called compound polygon, is complex connected region. In the complex connected region, including the outer boundary and inner boundaries, the islet polygon is inner boundary. The Polygons with islets was defined as follows: Set P_1, P_2, ..., P_n ($n > 1$) are simple polygon, if the degrees of all vertex of P_i (i = 1,2, ..., n) is 2, the edges without common vertex do not intersect, and one of polygons (set P_j, $1 \le j \le n$) included all the other polygons, then the polygon composed by P_1, P_2, ..., P_n is called islet polygons (shown in **Figure 1**), P_1 is called the outer polygon, the other is called the inner polygon (islet or hole) [7].

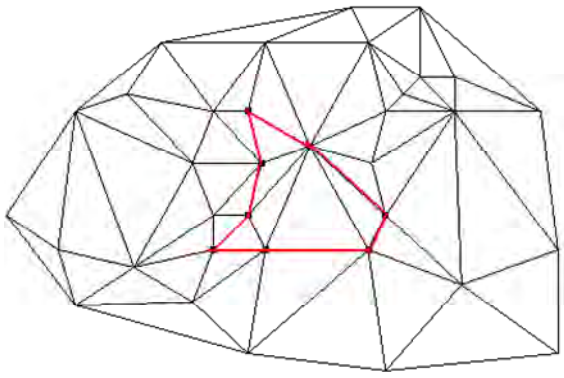


Figure 1. Constrained triangulation segment.

Triangulation with islets constraints is build delaunary triangulation between the islets and outer boundary that is with islets.

2.2. Classic Triangulation Algorithm with Islets Constraints

For the Delaunay triangulation algorithm with islets constrained, according to occasion of islets constrained data embedded, can be divided into the following categories:

1) Taking into account the impact of islets In the Delaunay triangulation, directly build Delaunay Triangulation with islets constraint. Such as the Triangulation algorithm with islets constrained data fields proposed by Liu Shao-Hua [4].

2) First, without considering islets constraints, building Delaunay triangulation with all the vertex of islet polygons and edges constraints, then delete the triangles within constraints fields. Such as Reference [7,8]. These algorithms are simple, compatible with non-constrained Delaunay triangulation.

3. Improvement of Delaunary Triangulation with Islets Constraints

3.1. Algorithm Theory

Two-dimensional Delaunay triangulation with islets constrained is described as: For the plane field D (P, Q), it containing a set of points P and Q, Q = {Qi ∈ D|I = 1, n2} is a points set that distributed in the boundary of islets, if a triangulation of the point set P, Q is a Delaunay triangulation (DT), delete the triangle within the islets, then it is a Delaunay triangulation with islets constrained on the plane field D (P, Q) [8].

Based on the above principle, we can build Delaunay triangulation with islets constrained step by step. First, without considering the nature of boundary points of islets, building Delaunay triangulation with the P and Q, then embedding edge constraint, the edges constrained are boundary line segment of islets, shown in **Figure 1**. Finally we found all the triangles inside the islets, according to need to be deleted, or retained. Based on the relationship between edges and triangles, this paper proposed a bi-directional search method and found all the triangles inside island, in order to complete with island constraints DT division.

3.2. Data Structure

According to the algorithm needs to establish topological relations among point, line, surface, and arc. This data structure can preferably establish topology data model of triangular mesh, and can be quickly index at the point, edge, triangle, and arcs. The algorithm is program with the Java, The data structure is as follows:

1) The point class:

```
public class Point{//class Point
public double x, y, z;
}
```

2) The edge class:

In order to finding two triangles that hane one and the same edge quickly, so the class has the two neighbor triangles.

```
Public class Edge {//Edge
private int [] pointNum = new int[2]
//the point number
private Triangle [] tri = new Triangle[2];
//Edges corresponding to two adjacent triangles
private boolean constrainEdge = false;
//is it binding edge
}
```

3) The triangle class:

In order to finding the edge of the triangle quickly, so the class has the three edges.

```
Public class Triangle {//class Triangle
private int [] pointNum = new int [3];
//Triangle of three dots
private Edge [] edge = new Edge [3];
//Side of the object created by the three Edge
}
```

4) The islet class:

Storages a list of all inner border line segment of islet, according to counter-clockwise storages.

```
Public class IsletArea {//Islet Class
private List <Edge> Pointlist = new LinkedList <Edge>
();
//the Island of arc
}
```

5) The Islets influence field class:

The internal class, record the edge within islet, and a known triangle in a side of the edge a.

```
private class IsletDomain {
private Edge e; //Islands within the affected side of
private Triangle tri; //Islandsthathey affect a certain
edge e adjacent to a known triangle
}
```

Considering the deleted or retained the triangle within Islands, in the algorithm, there is a list that stores the triangular within islet.

3.3. Algorithm Implementation

Assume all the arcs that surround the inner border of islets are counter-clockwise order, so the left side of closed arcs is within the islets, the right is exterior the islet, the triangle on the left of the arcs fall within the islet. The key of algorithm is to find the triangle within the islet, based on the relationship between the edges and

the triangle, from a known triangle and a triangle edge start to expend the other two edges of the known triangle and the adjacent triangle both sides, until the find all the triangles within the islets, the steps are as follows:

1) Doing Delaunay triangulation for the islets boundary points and the other discrete points within the plane.

2) All the borders of the islets are as line segment constraints, embedded in the Delaunay triangulation, carried out Delaunay triangulation with constraints, stored line segment constraints into constraint edge list.

3) Order all line segment constraints in the constraint edge list by counter-clockwise.

4) Pick-up any inner boundary arc (edge) *e* from anyone islets, and two triangles both sides *e*, to find the left side triangle *t* from these two triangles, building a new IsletDomain object with *e* and *t*, push it into the stack. Modify the topology of *e*, set the reference (pointer) of *e* and *t* to null.

5) Pick-up the top element of the stack, that is a IsletDomain object. Store the member triangle of IslandDomain, *tri*, into triangle within islets list, and pick-up the other two side edges of the *tri*, from the two direction to search, there are three situation:

- Two edges are not in the constraint edge list;
- one edge is the constraint edge list, another edge is not in the constraint edge list;
- Two edges are in the constraint edge list.

For three cases, if the edge is in the constraint edge list, looking for adjacent triangle of the edge, building the IsletDomain object, push stack; if the edge is not in the constraint edge list, modify the topology of the edge, set the reference (pointer) of the triangle that pointed to by *tri* to null.

6) Redo step 5, until the stack is empty.

7) According to the needs, delete or retain elete the triangles in the triangle within islets list.

Based on the above steps, the flow chart of Delaunay triangulation with island constraint is **Figure 2**.

Based on the above steps to complete Delaunay triangulation with islets constrained, the result shown in **Figure 3**.

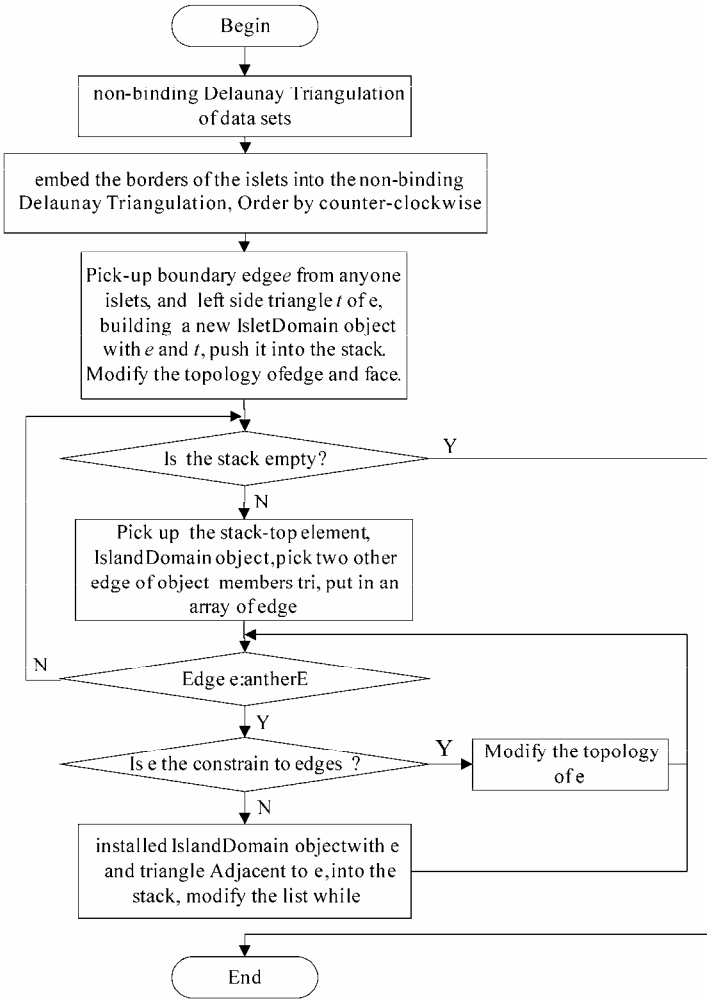


Figure 2. Flow diagram of delaunay triangulation with islets constrained.

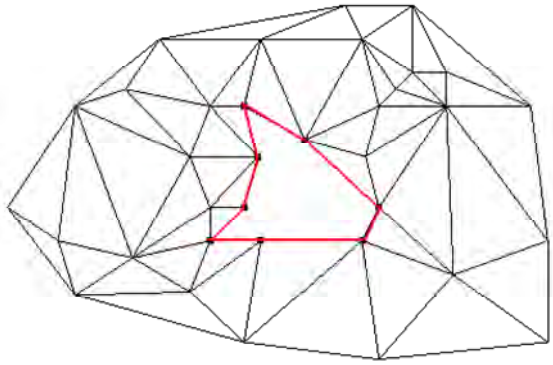
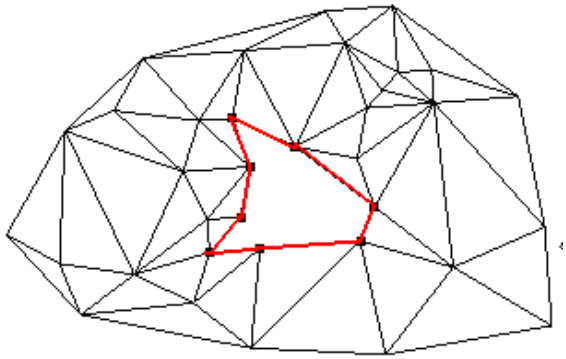


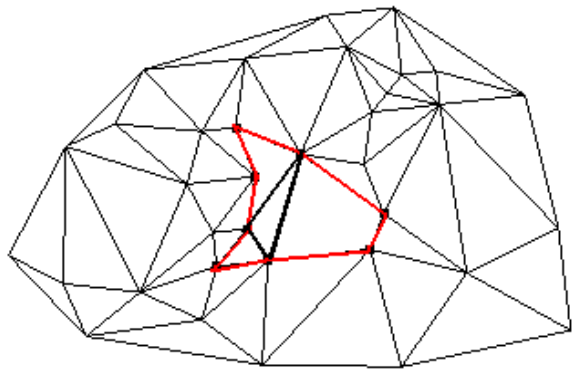
Figure 3. Delaunay triangulation with an islet constrained.

4. Algorithms Analysis

Tang Wei proposed algorithm and Ma Hong-bin proposed algorithm, simple, do not to determine the convex-concave of polygon and geometric intersection operations. Tang Wei's algorithm to find a triangle within islets by the line segment constraints that enclosed the islets, therefore, for the complex islets, there may be do not find all triangles within islet. (a) and (b) of **Figure 4** is

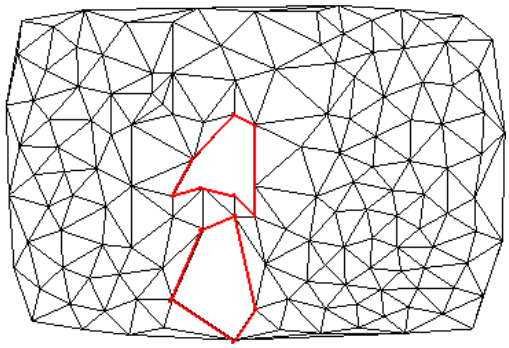


(a)

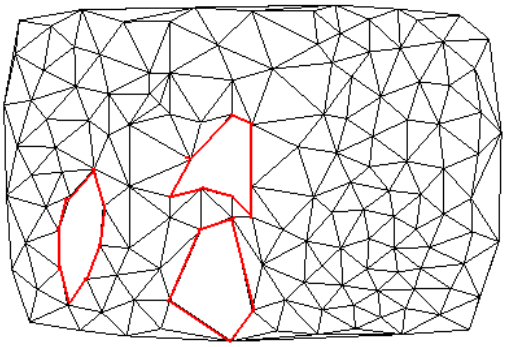


(b)

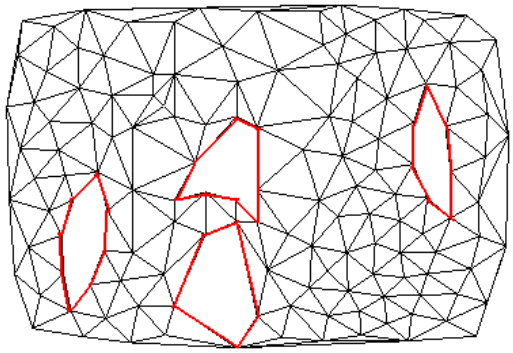
Figure 4. Tang Wei's algorithm effect diagram. (a) Triangular mesh based on lines list; (b) Triangular form based on triangular mesh.



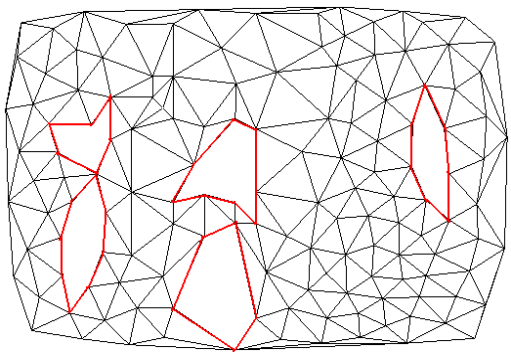
(a)



(b)



(c)



(d)

Figure 5. Delaunay triangulation with islets constrained. (a) Two islands; (b) Three islands; (c) Four islands; (d) Five islands.

Table 1. Algorithms comparison.

Islets number	This paper algorithm (ms)	Triangle center of gravity algorithm (ms)
2	2.5	4.5
3	3.8	5.7
4	4.5	6.6
5	5.3	8

based on the line list and triangles list the algorithm generated to form a Triangular mesh. Through the figure can know that the algorithm can remove the extra line in the line list, but can not remove the redundant triangles in the triangle list.

For the algorithm of Ma Hong-bin, asked the three vertices of all triangles in the triangulation are the vertex of the polygon, that is restrictive conditions, is not suitable for large scattered set of points and more Islets.

The algorithm of Center of gravity location triangle counts needs to determine all the triangles whether in the polygon, and the need for geometric intersection operations. When the triangular mash is more large, and a lot of Islets, it will waste too much resources when determination if the triangle is not within islets. Compared with algorithm proposed in this paper, the data shown in **Figure 5**, the results of comparison is **Table 1**.

5. Conclusions

Through the studied and summarized for existing algorithms, according to the topological relations between edges and triangles, proposed a method of looking for triangles within the islets, the algorithm simple, without geometric intersection operations. The algorithms been used in 3-D terrain model.

6. References

[1] Z. M. Ma and B. Luo, "Entire Optimized Triangulation Algorithm of Delaunay Triangle Network for DEM Construction," *Journal of Chang'an University* (*Natural Science Edition*), Vol. 28, No. 3, 2008, pp. 44-48.

[2] L. P. Chew, "Constrained Delaunay Triangulations," *ACM Symposium on Computational Geometry*, Springer-Verlag, Berlin, 1987, pp. 215-222.

[3] L. X. Wu, Y. B. Wang and W. Z. Shi, "Integral Ear Elimination and Virtual Point Based Updating Algorithms for Constraint Delaunay TIN," *Science in China*: *E*, Vol. 51, No. S1, 2008, pp. 135-144.

[4] S. H. Liu, P. G. Cheng and H. H. Chen, "Study of Algorithm for Triangulation of Restrained Data Set with Islets," *Computer Application*, Vol. 23, No. 4, 2003, pp. 96-98.

[5] S. G. Deng, M. Chen, *et al*., "Study on Algorithm for Delaunay Triangular Irregular Network of Islets Constrained Data Field," *Science of Surveying and Mapping*, Vol. 32, No. 5, 2007, pp. 63-64.

[6] M. Lamot and B. Zalik, "A Fast Polygon Triangulation Algrithm Based on Uniform Plane Subdivision," *Computers and Graphics*, Vol. 27, No. 2, 2003, pp. 239-253.

[7] H. B. Ma, J. T. Guo, *et al*., "Study on Delaunay Triangulation Algorithm for Polygon with inside Islets," *Journal of Northeastern University* (*Natural Science*), Vol. 30, No. 5, 2009, pp. 733-736.

[8] W. Tang, S. Z. Chen, *et al*., "An Approach to the Modification of the Triangulation Algorithm with Islets Constrains," *Hydrogeology* & *Engineering Geology*, Vol. 33, No. 5, 2006, pp. 58-60.

Intelligent Information Management, **2010, 2, 380-385**
doi:10.4236/iim.2010.26046 Published Online June 2010 (http://www.SciRP.org/journal/iim)

The Line Clipping Algorithm Basing on Affine Transformation

Wenjun Huang
College of Math and Computer Science, Guangxi University for Nationalities, Nanning, China
E-mail: *hwjart@126.com*
Received March 20, 2010; *revised April* 25, 2010; *accepted May* 27, 2010

Abstract

A new algorithm for clipping line segments by a rectangular window on rectangular coordinate system is presented in this paper. The algorithm is very different to the other line clipping algorithms. For the line segments that cannot be identified as completely inside or outside the window by simple testings, this algorithm applies affine transformations (the shearing transformations) to the line segments and the window, and changes the slopes of the line segments and the shape of the window. Thus, it is clear for the line segment to be outside or inside of the window. If the line segments intersect the window, the algorithm immediately (no solving equations) gets the intersection points. Having applied the inverse transformations to the intersection points, the algorithm has the final results. The algorithm is successful to avoid the complex classifications and computations. Besides, the algorithm is effective to simplify the processes of finding the intersection points. Comparing to some classical algorithms, the algorithm of this paper is faster for clipping line segments and more efficient for calculations.

Keywords: Computer Graphics, Line Clipping, Algorithm, Affine Transformation

1. Introduction and Previous Work

In computer graphics, line clipping is a basic and important operation, and has many applications. For example, extracting part of a defined scene for viewing must take line clipping. The region that includes the part of the defined scene is called a clip window. Generally, the window is a rectangle or a general polygon.

The early and classical algorithms of line clipping are Cohen-Sutherland Line Clipping algorithm [1], Cyrus-Beck Line Clipping algorithm [2] and Nicholl-Lee-Nicholl Line Clipping algorithm [3].

Cohen-Sutherland Line Clipping algorithm is one of the oldest and most popular line-clipping procedures. The algorithm uses a rectangle window with a coding scheme to subdivide the two-dimensional space which includes the graph. Then, each endpoint of a line segment of the graph is assigned the code of the sub-region in which it lies. And according to the value of the "&" and "|" which are made by the two codes of the two endpoints of the line segment, the algorithm determines the line segment to be inside of the widow or not. For the simple situations (the line segments are completely inside or outside of the window), the algorithm can quickly get the results. But for the line segments that cannot be identified as completely inside or completely outside the window by the scheme of the algorithm, the algorithm has to make computations and turn the line segments into the "simple situations". Obviously, if the line segment is outside of the window, the computations are waste.

Later, Cyrus-Becky proposed Cyrus & Beck algorithm. The algorithm treats line in parametric form. The theoretical model of this algorithm is general. However, it is rather inefficient. To clip a line segment which is neither vertical nor horizontal and lies entirely within the window, it will perform 12 additions, 16 subtractions, 20 multiplications and 4 divisions [4]. Besides, for the general case (the line segments will cross all the boundaries of the window), the algorithm first makes computations and find the parameters of the intersection points. Then, according to the signs of the denominators of the parameters and the values of the parameters, the algorithm determines which parts of the line segments are inside the window. Clearly, if the line segment is outside of the window, the computations are useless.

In [3], Nicholl-Lee-Nicholl Line Clipping algorithm makes four rays which pass an endpoint of the line segment and four vertexes of the window, and creates three regions by the four rays. Then, the algorithm determines which region that the line segment lies in, and determines finding the intersections or rejecting the line segment. Before finding the intersection points of the line segment and the window, the algorithm first determines the position of the first endpoint of the line segment for the nine possible regions relative to the clipping window. If the point is not in the one of the three especial regions, the algorithm has got to transform the point to the one of the three especial regions. To find the region in which the other endpoint of the line segment lies, the algorithm has got to compare the slope of the line segment to the slopes of the four rays. So, for the algorithm, finding the intersection points are efficient, but finding the positions of the two endpoints of the line segment are more complicated than Cohen-Sutherland Line Clipping algorithm.

Independently, You-dong Liang, Brian A.Barsky, Mel Slater offered a more faster algorithm [5,6]. This algorithm is based on a parametric representation of the line segment. The algorithm is somewhat complicated and inefficient. To clip a line segment which is neither vertical nor horizontal, it will perform 16 comparisons, 7 subtractions, and 4 divisions.

In [7], Vaclv Skala proposed a line clipping algorithm for convex polygon window. The algorithm uses the binary search to find the intersections in the clipping window. The complexity is O (lg N). But for the rectangle window, the algorithm does not have obvious advantage in comparison with the Cyrus-Beck algorithm.

In [8], the authors proposed the Optimal Tree algorithm. Based on the regions (there are nine regions subdivided by the four boundaries of the window) that the endpoints of the line segment lies in, the authors proposed five types of "Partition-Pairs": the "window-side or side-window" (including 8 cases), the "window-corner or corner-window" (including 16 cases), the "side-side" (including 20 cases), the "side-corner or corner-side" (including 16 cases) and the "corner-corner" (including 4 cases). There are 64 cases in the five types of "Partition-Pairs" and the optimal tree includes these 64 cases. The algorithm performed uniformly faster than all above algorithms. But the algorithm is too complicated.

In [9], the author proposed an algorithm based on homogeneous coordinates. In the algorithm, the author assumes a rectangular window P and a line p given as $F(x) = ax + by + c = 0$. The line p subdivides the space into two half-spaces as $F(x) < 0$ and $F(x) >= 0$. According to the locations of all the vertexes of the window to the line, the author makes out 16 possible cases and makes a table storing the cases. To clipping a line, the algorithm makes the calculations and determines the locations of all the vertexes of the window to the line. Having compared the locations with the cases in the table, the algorithm determines which edges of the window intersect the line and finds the intersection points by the cross products of their homogeneous coordinates. The algorithm is inefficient. To clip a line segment which will cross the window, the algorithm first codes the two endpoints of the line segment, and makes 4 comparisons and 2 cross products (taking 12 multiplications). If turning the intersection points (xi, yi, w) into (xi/w, yi/w), the algorithm still makes 4 divisions. So, in Euclidean space the computational complexity of the algorithm is more than Cohen- Sutherland algorithm.

In this paper, a new line clipping algorithm for a rectangle clip window will be given. Comparing those algorithms above, this algorithm makes the speed of line clipping faster and makes the calculations more efficient.

2. Theorems

2.1. Theorem 1

In a plane, the necessary and sufficient conditions for two line segments without any points of intersection are that there are no any points of intersection of the two line segments after applying an affine transformation to the two line segments.

Proof. We suppose that there are two line segments L_1 (the endpoints are p_{11} and p_{12}) and L_2 (the endpoints are p_{21} and p_{22}) in a plane, and $L_1 \bigcap L_2 = \varnothing$. Also, we suppose that there is a affine transformation T, and we apply the affine transformation T to the two line segments L_1 and L_2:

$$T(L_1) = L_1' \text{ (the endpoints are } p_{11}' \text{ and } p_{12}'),$$

$$T(L_2) = L_2' \text{ (the endpoints are } p_{21}' \text{ and } p_{22}').$$

2.1.1. The Sufficient Condition (Proof by Contradiction)

If $L_1' \bigcap L_2' = A'(A' \neq \varnothing)$, we apply the T^{-1} (T^{-1} exist because T is an affine transformation) to L_1' and L_2', and have $T^{-1}(L_1') = L_1$, $T^{-1}(L_2') = L_2$. The T^{-1} is still an affine transformation. According to the properties of affine transformation, we get the conclusion: Straight line $L_1 \bigcap$ straight line $L_2 \neq \varnothing$. So, we set the straight line $L_1 \bigcap$ The straight line $L_2 = A$ ($A \neq \varnothing$). Because $L_1 \bigcap L_2 = \varnothing$, so the point $A \in$ the extension of the line segment L_1 or the line segment L_2. So

$$p_{11}A / p_{12}A > 0 \text{ or } p_{21}A / p_{22}A > 0.$$

But

$$P_{11}'A' / P_{12}'A' < 0 \text{ and } p_{21}'A' / p_{22}'A' < 0.$$

Those are contradictory in the properties of affine transformation.

2.1.2. The Necessary Condition

The proof is as same as the proof of the sufficient condition.

From the theorem, two important inferences can be derived:

1) On a plane, the necessary and sufficient conditions for two line segments with a point of intersection are that there is a point of intersection of the two line segments after applying an affine transformation to the two line segments.

2) On a plane, the necessary and sufficient conditions for that a line segment is inside of a window (or outside of the window, or across the window) are that the line segment is inside of the window (or outside of the window, or across the window) after applying an affine transformation to the line segment and the window.

2.2. Theorem 2

In a rectangular coordinate system, we suppose that the slope of a straight line a (The endpoints are $A_1(x_{a1}, y_{a1})$ and $A_2(x_{a2}, y_{a2})$) is $1/c(c \neq 0)$ and the straight line b (The endpoints are $B_1(x_{b1}, y_{b1})$ and $B_2(x_{b2}, y_{b2})$) is vertical to the axis x (*i.e.* $b \perp$ the axis x). If apply the affine transformation

$$T_x : x' = x\text{-}cy,\ y' = y;\ ((x, y) \text{ is a point.})$$

to the line segments a and b, *i.e.*

$$a' = T_x(a) \text{ and } b' = T_x(b).$$

There is a conclusion: The line segment $a' \perp$ the axis x and the slope of $b' = -1/c$.

Proof. $T_x(a)$ and $T_x(b)$ are

$$x_i' = x_i - cy_i,\ y_i' = y_i,\ (i = a1, a2, b1, b2).$$

1) $x_{a1}' - x_{a2}'$
$= (x_{a1} - cy_{a1}) - (x_{a2} - cy_{a2})$
$= (x_{a1} - x_{a2}) - c(y_{a1} - y_{a2})$
$= (x_{a1} - x_{a2}) - (x_{a2} - x_{a1})/(y_{a2} - y_{a1})(y_{a1} - y_{a2})$
$= 0$
$\Rightarrow a' \perp$ the axis x.

2) The slope of b'
$= (y_{b2}' - y_{b1}')/(x_{b2}' - x_{b1}')$
$= (y_{b2} - y_{b1})/((x_{b2} - cy_{b2}) - (x_{b1} - cy_{b1}))$
$= (y_{b2} - y_{b1})/((x_{b2} - x_{b1}) - (cy_{b2} - cy_{b1}))$
$= (y_{b2} - y_{b1})/(0 - (cy_{b2} - cy_{b1}))$
$= -1/c$

2.3. Theorem 3

In a rectangular coordinate system, we suppose that the slope of a straight line a (The endpoints are $A_1(x_{a1}, y_{a1})$ and $A_2(x_{a2}, y_{a2})$) is c $(c \neq 0)$ and the straight line b (The endpoints are $B_1(x_{b1}, y_{b1})$ and $B_2(x_{b2}, y_{b2})$) is vertical to the axis y (*i.e.* $b \perp$ the axis y). If apply the affine transformation

$$T_y : x' = x,\ y' = -cx + y;\ ((x, y) \text{ is a point.})$$

to the line segment a and b, *i.e.*

$$a' = T_y(a) \text{ and } b' = T_y(b)$$

There is a conclusion: The line segment $a' \perp$ the axis y and the slope of $b' = -c$.

Proof. $T_y(a)$ and $T_y(b)$ are

$$y_i' = y_i - cx_i,\ x_i' = x_i,\ (i = a1, a2, b1, b2).$$

1) $y_{a1}' - y_{a2}'$
$= (y_{a1} - cx_{a1}) - (y_{a2} - cx_{a2})$
$= (y_{a1} - y_{a2}) - c(x_{a1} - x_{a2})$
$= (y_{a1} - y_{a2}) - (y_{a2} - y_{a1})/(x_{a2} - x_{a1})(x_{a1} - x_{a2})$
$= 0$
$\Rightarrow a' \perp$ the axis y.

2) The slope of b'
$= (y_{b2}' - y_{b1}')/(x_{b2}' - x_{b1}')$
$= ((y_{b2} - cx_{b2}) - (y_{b1} - cx_{b1}))/(x_{b2} - x_{b1})$
$= ((y_{b2} - y_{b1}) - (cx_{b2} - cx_{b1}))/(x_{b2} - x_{b1})$
$= (0 - c(x_{b2} - x_{b1}))/(x_{b2} - x_{b1})$
$= -c$

3. The Basic Idea of the Algorithm

We suppose that there are a rectangular window and some line segments in a rectangular plane coordinate system (see **Figure 1**).

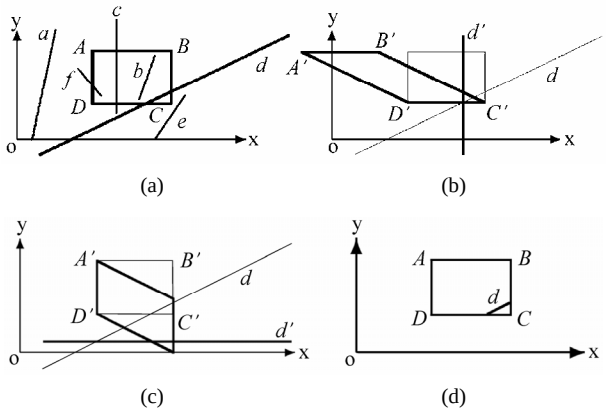
Four types of the line segments are gotten by classifying the line segments according to the positions against the window.

The first are outside of the rectangular window (see the line segment a in **Figure 1(a)**).

The second are inside of the rectangular window (see the line segment b in **Figure 1(a)**).

The third are parallel or vertical to the edges of the rectangular window and intersecting the rectangular window (see the line segment c in **Figure 1(a)**).

The fourth are the other line segments that do not belong to any types as above (see the line segments d, e, and f in **Figure 1(a)**).



In the Figure (a) and (d), the vertexes of the window are $A\,(x_{wl}, y_{wt})$, $B\,(x_{wr}, y_{wt})$, $C\,(x_{wr}, y_{wb})$ and $D\,(x_{wl}, y_{wb})$.

In the Figure (b), the vertexes of the window are $A'\,(x'_{wlt}, y'_{wt})$, $B'\,(x'_{wrt}, y'_{wt})$, $C'\,(x'_{wrb}, y'_{wb})$ and $D'\,(x'_{wlb}, y'_{wb})$.

In the Figure (c), the vertexes of the window are $A'\,(x'_{wt}, y'_{wtl})$, $B'\,(x'_{wr}, y'_{wtr})$, $C'\,(x'_{wr}, y'_{wbr})$ and $D'\,(x'_{wl}, y'_{wbl})$.

Figure 1. The process of the clipping. (a) Lines and window; (b) T_x(d) and T_x(w); (c) T_y(d) and T_y(w); (d) The result.

For the first, the second and the third, we process them with subtraction. For the fourth, we apply the affine transformations to the line segments of the fourth and apply the same affine transformations to the window with the theorem 2 and the theorem 3, turning the fourth into the line segments that are vertical or parallel to axis *x*, and turning the window into a parallelogram that have two edges which are vertical to the line segment (see **Figures 1 (b)** and **(c)**). Now, according to theorem 1 ~ theorem 3, we can easily determine the line segment is outside of the window or across the window.

After getting the intersections of the line segment and the window, we apply the inverse transformations of the affine transformations to the intersections, and the line segment clipped by the window is gotten.

4. The Steps of the Algorithm

/*In the step (2), (3), (4) and (5), we process the first, the second, the third and the fourth as above orderly.*/

1) Preparation:

Give a rectangular plane coordinate system *xoy*;

Give four edges of a rectangular window *W*:

float x_{wl}, x_{wr}, y_{wt}, y_{wb}, ($x_{wl} < x_{wr}$, $y_{wb} < y_{wt}$);

Give a line segment randomly:

float $p_1(x_1, y_1)$, $p_2(x_2, y_2)$; int flag: = 0;

float $p_{11}(x_{11}, y_{11})$: = $p_1(x_1, y_1)$,

$p_{22}(x_{22}, y_{22})$: = $p_2(x_2, y_2)$;

int f_1: = ($x_{wl} \le x_1 \le x_{wr}$) && ($y_{wb} \le y_1 \le y_{wt}$);

int f_2: = ($x_{wl} \le x_2 \le x_{wr}$) && ($y_{wb} \le y_2 \le y_{wt}$);

2) if ((x_1 and x_2) $\le x_{wl}$) || ((x_1 and x_2) $\ge x_{wr}$) || ((y_1 and y_2) $\le y_{wb}$) || ((y_1 and y_2) $\ge y_{wt}$), goto (7);

3) else if (f_1 && f_2), goto (6);

4) else if ($y_1 = y_2$) {

if ($x_1 > x_2$){ swap(p_1, p_2); swap(p_{11}, p_{22});}

if ($x_1 \le x_{wl}$) and ($x_2 \ge x_{wr}$)

{ x_{11}: = x_{wl}; x_{22}: = x_{wr}; goto (6);}

else if ($x_1 \le x_{wl}$) and ($x_2 \ge x_{wl}$)

{ x_{11}: = x_{wl}; goto (6);}

else if ($x_1 \le x_{wr}$) and ($x_2 \ge x_{wr}$)

{ x_{22}: = x_{wr}; goto (6);}

}

else if ($x_1 = x_2$) {

if ($y_1 > y_2$){ swap(p_1, p_2); swap(p_{11}, p_{22});}

if ($y_1 \le y_{wb}$) and ($y_2 \ge y_{wt}$)

{ y_{11}: = y_{wb}; y_{22}: = y_{wt}; goto (6);}

else if ($y_1 \le y_{wb}$) and ($y_2 \ge y_{wb}$)

{ y_{11}: = y_{wb}; goto (6);}

else if ($y_1 \le y_{wt}$) and ($y_2 \ge y_{wt}$)

{ y_{22}: = y_{wt}; goto (6);}

}

5) else{

5.1) *c*: = (x_2 - x_1)/(y_2 - y_1);

5.2) if ($y_1 > y_2$) {swap (p_1, p_2); swap(p_{11}, p_{22});}

p'_1: = $T_x(p_1)$; p'_2: = $T_x(p_2)$; *w'*:= $T_x(w)$;

/*After getting the affine transformation, the rectangular window become a parallelogram having two edges that parallel to axis *x*, and the line segment (p'_1, p'_2) is vertical to the axis *x*. See **Figure 1(b)**.*/

5.3) if (*c* >0) && (($x'_1 \ge x'_{wrb}$) || ($x'_1 \le x'_{wlt}$)) goto (7); /*see **Figure 1(b)***/

else if(*c* < 0) && (($x'_1 \ge x'_{wrt}$) || ($x'_1 \le x'_{wlb}$)) goto(7); /*refer to the **Figure 1(b)***/

else {

if ($x'_1 \ge x'_{wlb}$) and ($x'_1 \le x'_{wrb}$) {*flag*++;

if ($y'_1 \le y'_{wb}$) and ($y'_2 \ge y'_{wb}$) $y'_1 = y'_{wb}$;

$y_{11} = T_x^{-1}(y'_1)$; $x_{11} = T_x^{-1}(x'_1)$;

}

if ($x'_1 \ge x'_{wlt}$) and ($x'_1 \le x'_{wrt}$) {*flag*++;

if ($y'_1 \le y'_{wt}$) and ($y'_2 \ge y'_{wt}$) $y'_2 = y'_{wt}$;

$y_{22} = T_x^{-1}(y'_2)$; $x_{22} = T_x^{-1}(x'_2)$;

}

if (*flag* = 2) goto (6);

/*"flag = 2" means that the line clipping have been finished.*/

5.4) if ($x_1 > x_2$){swap(p_1, p_2); swap(p_{11}, p_{22});} c: = 1/c; $p_1^{'} := T_y(p_1)$; $p_2^{'} := T_y(p_2)$; $w' = T_y(w)$;

/*After getting the affine transformation, the rectangular window become a parallelogram having two edges that parallel to axis y, and the line segment ($p_1^{'}$, $p_2^{'}$) is vertical to the axis y. See **Figure 1(c).***/

5.5) if ($y_1^{'} \geq y_{wbl}^{'}$) and ($y_1^{'} \leq y_{wtl}^{'}$) {*flag*++;

if ($x_1^{'} \leq x_{wl}^{'}$) and ($x_2^{'} \geq x_{wl}^{'}$) $x_1^{'} = x_{wl}^{'}$;

$y_{11} = T_y^{-1}(y_1^{'})$; $x_{11} = T_y^{-1}(x_1^{'})$;

}

if ($y_1^{'} \geq y_{wbr}^{'}$) and ($y_1^{'} \leq y_{wtr}^{'}$) { *flag*++;

if ($x_1^{'} \leq x_{wr}^{'}$) and ($x_2^{'} \geq x_{wr}^{'}$) $x_2^{'} = x_{wr}^{'}$;

$y_{22} = T_y^{-1}(y_2^{'})$; $x_{22} = T_y^{-1}(x_2^{'})$;

}

/*see **Figure 1 (c)***/

}

6) Drawing the line (x_{11}, y_{11}, x_{22}, y_{22});

7) The end;

5. The Calculation Complexity

In the most complex case (the lines belong to the fourth type as above), after using 2 divisions to get the slope and the 1/*slope* of a line segment, the algorithm uses two steps with 4 multiplications to make the clipping.

First, the algorithm translates the window and places the "bottom edge" on the axis x, and makes the same translation for the line segment. Then, it uses one multiplication to apply an affine transformation to the "top edge" of the window and uses another multiplication to get the intersection of the window and the line segment (see **Figure 1(b)**).

Second, the algorithm translates the window and places the "left edge" on the axis y, and applies the translation to the line segment. Then, it uses one multiplication to apply an affine transformation to the "right edge" of the window and use another multiplication to get the intersection of the window and the line segment (see **Figure 1(c)**).

So, the algorithm at most uses 2 divisions and 4 multiplications to finish the clipping for a line segment (See **Table 1**).

Here, we list the calculation complexities of the algorithm and other algorithms in **Table 1** for comparing. Where "C-S algorithm", "C-B algorithm", "N-L-N algorithm", "L-B algorithm", "VS algorithm" and "L-B-2 algorithm" indicate that the Cohen-Sutherland Algorithm [1], the Cyrus-Beck Algorithm [2], the Nicholl-Lee-Nicholl Algorithm [3], the Liang-Barsky-Slater Algorithm [5,10], the O(lg N) Line Clipping Algorithm in E^2 [7] and the Optimal Tree Algorithm for Line Clipping [8] orderly.

6. Results and Discussion

The algorithm in this paper has been realized with a computer in C language with TC system. It is successful for the algorithm to clip the random line segments (see **Figure 1(a)**). We take the special situation like line segment d in Figure 1 for a sample to perform the process of the clipping and to make comparisons. The comparisons between the algorithm in this paper and Cohen-Sutherland algorithm have been list in **Table 2**. In **Table 2**, the first row give the numbers of the line segments, the second row give the times of performing the algorithm in this paper, and the third row give the times of performing Cohen-Sutherland algorithm. We use the function *cclok*() in TC to keep the times.

Some important facts are as follows:

1) The complexity of the algorithm in this paper is less than VS algorithm, see **Table 1**;

2) L-B-2 algorithm is faster than C-S algorithm, L-B algorithm, N-L-N algorithm and C-B algorithm [6,8];

Table 1. The calculation complexities.

Algorithms	Operations		
	×	÷	*making codes*
Our algorithm	4	2	0
C-S algorithm	4	2	6
L-B algorithm	4	4	0
N-L-N algorithm	1	6	0
C-B algorithm	12	2	0
L-B-2 algorithm	4	2	0
VS algorithm	7	2	0

Table 2. The times of the clipping.

	Lines				
	5000	10,000	20,000	30,000	60,000
T	0	0	0	0	1
Tc-s	0	1	1	1	3
1,200,000		1,500,000	1,800,000		2,100,000
18		23	28		31
56		70	85		98

3) L-B-2 algorithm is 2.5 (the average) or 3.03 (the maximum) times [8] as fast as Cohen-Sutherland algorithm for the speed of line clipping.

4) The algorithm in this paper is 3.1 (the average) or 3.5 (the maximum) times as fast as Cohen-Sutherland algorithm, see **Table 2**.

From the facts above, we derive the conclusion that the algorithm in this paper is faster than the other algorithms in **Table 1** for line clipping.

7. Conclusions

For the special situation that the line segment or its extension (like the line segment *d* in **Figure 1**) intersects all the edges or their extensions of the window, the clipping speed of our algorithm is obviously faster than other algorithms. But for the random situations, the average clipping speed of our algorithm is a little bit faster than other algorithms.

8. References

[1] D. Hearn and M. P. Baker, "Computer Graphics," C Version, 2nd Edition, Prentice Hall, Inc., Upper Saddle River, 1998, p. 226.

[2] M. Cyrus and J. Beck, "Generalized Two and Three Dimensional Clipping," *Computers and Graphics*, Vol. 3, No. 1, 1978, pp. 23-28.

[3] D. Hearn and M. P. Baker, "Computer Graphics," C Version, 2nd Edition, Prentice Hall, Inc., Upper Saddle River, 1998, p. 233.

[4] T. M. Nicholl, D. T. Lee and R. A. Nicholl, "An Efficient New Algorithm for 2-D Line Clipping: Its Development and Analysis," *Computers and Graphics*, Vol. 21, No. 4, 1987, pp. 253-262.

[5] D. Hearn and M. P. Baker, "Computer Graphics," C Version, 2nd Edition, Prentice Hall, Inc., Upper Saddle River, 1998, p. 230.

[6] C. B. Chen and F. Lu, "Computer Graphics Basis," Publishing House of Electronics Industry, Beijing, 2006, pp. 167-168.

[7] V. Skala, "O (lg N) Line clipping Algorithm in E^2," *Computers and Graphics*, Vol. 18, No. 4, 1994, pp. 517-527.

[8] Y. D. Liang and B. A. Barsky, "The Optimal Tree Algorithm for Line Clipping," *Technical Paper Distributed at Eurographics*'92 *Conference*, Cambridge, 1992, pp. 1-38.

[9] V. Skala, "A New Approach to Line and Line Segment Clipping in Homogeneous Coordinates," *Visual Computer*, Vol. 21, No. 11, 2005, pp. 905-914.

[10] Y. D. Liang, B. A. Barsky and M. Slater, "Some Improvements to a Parametric Line Clipping Algorithm," *Technical Report No. UCB/CSD* 92/688, Computer Science Division, University of California, Berkeley, 1992, pp. 1-22.

Intelligent Information Management, **2010, 2, 386-390**
doi:10.4236/iim.2010.26047 Published Online June 2010 (http://www.SciRP.org/journal/iim)

Experiments with Two New Boosting Algorithms

Xiaowei Sun[1], Hongbo Zhou[2]
[1]*Software College, Shenyang Normal University, Shenyang, China*
[2]*Liaoning SG Automotive Group Co., Ltd., Shenyang, China*
E-mail: *junyaomail@163.com, hongbo.zhou@hotmail.com*
Received April 2, 2010; *revised May* 5, 2010; *accepted June* 8, 2010

Abstract

Boosting is an effective classifier combination method, which can improve classification performance of an unstable learning algorithm. But it dose not make much more improvement of a stable learning algorithm. In this paper, multiple TAN classifiers are combined by a combination method called Boosting-MultiTAN that is compared with the Boosting-BAN classifier which is boosting based on BAN combination. We describe experiments that carried out to assess how well the two algorithms perform on real learning problems. Finally, experimental results show that the Boosting-BAN has higher classification accuracy on most data sets, but Boosting-MultiTAN has good effect on others. These results argue that boosting algorithm deserve more attention in machine learning and data mining communities.

Keywords: Boosting, Combination Method, TAN, BAN, Bayesian Network Classifier

1. Introduction

Classification is a fundamental task in fault diagnosis, pattern recognition and forecasting. In general, a classifier is a function that chooses a class label (from a group of predefined labels) for instance described by a set of features (attributes). Learning accurate classifiers from pre-classified data is a very active research topic in machine learning and data mining. In the past two decades, many classifiers have been developed, such as decision trees based classifiers and neural network based classifiers.

Boosting [1-4] is a general method for improving the performance of any "weak" learning algorithm. In theory, boosting can be used to significantly reduce the error of any "weak" learning algorithm that consistently generates classifiers which need only be a little bit better than random guessing. Despite the potential benefits of boosting promised by the theoretical results, the true practical value of boosting can only be assessed by testing the method on "real" learning problems. In this paper, we present such experimental assessment of two new boosting algorithms.

The first provably effective boosting algorithms were presented by Schapire [5] and Freund [3]. Boosting works by repeatedly running a given weak learning algorithm on various distributions over the training data, and then combining the classifiers produced by the weak learner into a single composite classifier. More recently, we described and analyzed AdaBoost, and we argued that this new boosting algorithm has certain properties which make it more practical and easier to implement than its predecessors.

TAN [6] and BAN [7] are augmented Bayesian network classifiers provided by Friedman and Cheng J. In these papers, we treat the classification node as the first node in the ordering. The order of other nodes is arbitrary; we simply use the order they appear in the dataset. Therefore, we only need to use the CLB1 algorithm, which has the time complexity of $O(N^2)$ on the mutual information test (N is the number of attributes in the dataset) and linear on the number of cases. The efficiency is achieved by directly extending the Chow-Liu tree construction algorithm [8] to a three-phase BN learning algorithm: *drafting*, which is essentially the Chow-Liu algorithm, *thickening*, which adds edges to the draft, and *thinning*, which verifies the necessity of each edge.

In this paper, multiple TAN classifiers are combined by a combination method called Boosting-MultiTAN that is compared with the Boosting-BAN classifier which is boosting based on BAN combination. Section 2 defines two classes of BNs, *i.e.*, Tree augmented Naive-Bayes (TANs) and BN augmented Naïve-Bayes (BANs), and describes methods for learning each. Section 3 proposes two new boosting algorithms, such as Boosting-

MultiTAN classifier and Boosting-BAN classifier. Section 4 presents and analyzes the experimental results. These results argue that boosting algorithm deserve more attention in machine learning and data mining communities.

2. Learning Bayesian Network Classifiers

Learning Bayesian network classifiers involves two steps: structure learning and parameter (conditional probability tables) learning. We will focus on structure learning methods for different Bayesian network classifiers in the subsections below.

2.1. Tree Augmented Naive-Bayes (TAN)

Letting $X = \{x_1, \ldots, x_n, c\}$ represent the node set (where c is the classification node) of the data. The algorithm for learning TAN classifier first learns a tree structure over V\{c}, using mutual information tests. It then adds a link from the classification node to each feature node in the manner as we construct a Naive-Bayes (*i.e.*, the classifycation node is a parent of all other nodes.) A simple TAN structure is shown in **Figure 1** (Note that features x_1, x_2, x_3, x_4 form a tree).

The learning procedure can be described as follows.

1) Take the training set and $X\backslash\{c\}$ as input.

2) Call the modified Chow-Liu algorithm. (The originnal algorithm is modified by replacing every mutual information test $I(x_i, x_j)$ with a conditional mutual information test $I(x_i, x_j|\{c\})$).

3) Add c as a parent of every x_i where $1 \le i \le n$.

4) Learn the parameters and output the TAN.

This algorithm, which is modified from the Chow-Liu algorithm, requires $O(N^2)$ numbers of conditional mutual information tests. This algorithm is essentially the first phase of the BAN-learning algorithm. TAN classifier is stable that can not be combined with a quite strong learning algorithm by boosting.

2.2. BN Augmented Naive-Bayes (BAN)

BAN classifier has been studied in several papers. The basic idea of this algorithm is just like the TAN learner of Subsection 2.1, but the unrestricted BN-learning algorithm instead of a tree-learning algorithm (see **Figure 2**).

Letting $X = \{x_1, \ldots, x_n, c\}$ represent the feature set (where c is the classification node) of the data, the learning procedure based on mutual information test can be described as follows.

1) Take the training set and $X\backslash\{c\}$ (along with the ordering) as input.

2) Call the modified CBL1 algorithm. (The original algorithm is modified in the following way: replace every

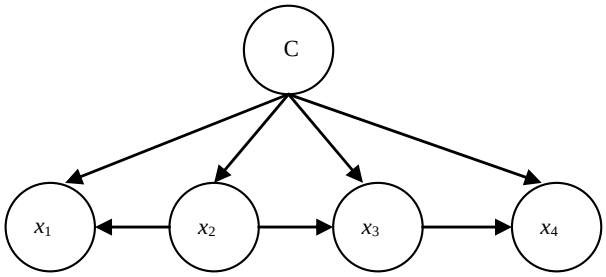


Figure 1. A simple TAN structure.

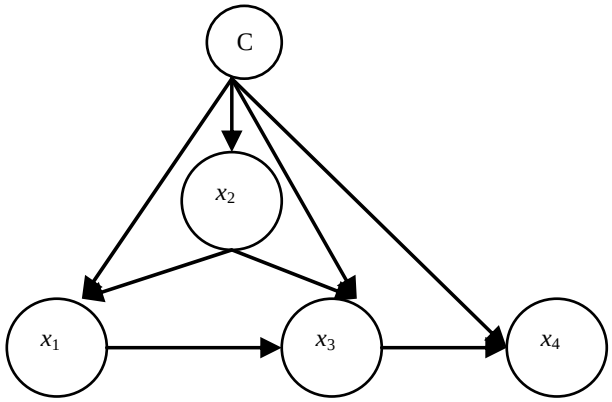


Figure 2. A simple BAN structure.

mutual information test $I(x_i, x_j)$ with a conditional mutual information test $I(x_i, x_j|\{c\})$; replace every conditional mutual information test $I(x_i, x_j|Z)$ with $I(x_i, x_j|Z + \{c\})$, where $Z \subset X\backslash\{c\}$.

3) Add c as a parent of every x_i where $1 \le i \le n$.

4) Learn the parameters and output the BAN.

Like the TAN-learning algorithm, this algorithm dose not require additional mutual information tests, and so it requires $O(n^2N)$ (where n is the number of node attributes; N is the number of training examples) mutual information tests. The longest time spent in the algorithm is to calculate mutual information. In BAN structure, the second step in the three-phase is used ε to sort mutual information. The ε is a given small positive threshold, it is not fixed, and can be changed in many times. By setting different thresholds ε can construct many BAN classifiers. BAN classifier is unstable that can be combined with a quite strong learning algorithm by boosting.

3. Two New Boosting Algorithms

3.1. Boosting-MultiTAN Algorithm

GTAN [9] is proposed by Hongbo Shi in 2004. GTAN used conditional mutual information as CI tests to measure the average information between two nodes when the statuses of some values are changed by the condition-set C. When $I(x_i, x_j|\{c\})$ is larger than a certain threshold

value ε, we choose the edge to the BN structure to form TAN. *Start-edge* and ε are two important parameters in GTAN. Different *Start-edges* can construct different TANs. GTAN classifier is unstable that can be combined with a quite strong learning algorithm by boosting.

The Boosting-MultiTAN algorithm may be characterized by the way in which the hypothesis weights w_i are selected, and by the example weight update step.

Boosting-MultiTAN (*Dataset*, *T*):

Input: sequence of *N* example *Dataset* = $\{(x_1, y_1),\ldots, (x_N, y_N)\}$ with labels $y_i \in Y = \{1,\ldots, k\}$, integer *T* specifying number of iterations.

Initialize $w_i^{(1)} = 1/N$ for all *i*, *TrainData*-1 = *Dataset*

Start-edge = 1; *t* = 1; *l* = 1

While ((*t* ≤ *T*) *and* (*l* ≤ 2*T*))

1) Use *TrainData-t* and *start-edge* call GTAN, providing it with the distribution.

2) Get back a hypothesis $TAN^{(t)} = X \to Y$.

3) Calculate the error of $TAN^{(t)}$:

$$e^{(t)} = \sum_{i=1}^{N} w_i^{(t)} I\left(y_i \neq TAN^{(t)}\left(x_i\right)\right).$$

If $e^{(t)} \geq 0.5$, then set *T* = *t* – 1 and abort loop.

4) Set $\mu^{(t)} = e^{(t)} \Big/ \left(1 - e^{(t)}\right)$.

5) Updating distribution $w_i^{(t+1)} = w_i^{(t)}\left(\mu^{(t)}\right)^s$, where $s = 1 - I\left(y_i \neq TAN^{(t)}\left(x_i\right)\right)$.

6) Normalize $w_i^{(t+1)}$, to sum to 1.

7) t = t + 1, l = l + 1, *start-edge* = *start-edge* + *n*/2*T*.

8) end While

Output the final hypothesis:

$$H(x) = \arg\max_{y \in Y}\left(\sum_{t=1}^{T}\left(\log\left(\frac{1}{\mu^{(t)}}\right)\right) * I\left(y = TAN^{(t)}(x)\right)\right)$$

3.2. Boosting-BAN Algorithm

Boosting-BAN works by fitting a base learner to the training data using a vector or matrix of weights. These are then updated by increasing the relative weight assigned to examples that are misclassified at the current round. This forces the learner to focus on the examples that it finds harder to classify. After *T* iterations the output hypotheses are combined using a series of probabilistic estimates based on their training accuracy.

The Boosting-BAN algorithm may be characterized by the way in which the hypothesis weights w_i are selected, and by the example weight update step.

Boosting-BAN (*Dataset*, *T*):

Input: sequence of *N* example *Dataset* = $\{(x_1, y_1),\ldots, (x_N, y_N)\}$ with labels $y_i \in Y = \{1, \ldots, k\}$, integer *T* specifying number of iterations.

Initialize $w_i^{(1)} = 1/N$ for all *i*, *TrainData*-1 = *Dataset*

Do for t = 1, 2,…, *T*

1) Use *TrainData-t* and threshold ε call BAN, providing it with the distribution.

2) Get back a hypothesis $BAN^{(t)} = X \to Y$.

3) Calculate the error of $BAN^{(t)}$:

$$e^{(t)} = \sum_{i=1}^{N} w_i^{(t)} I\left(y_i \neq BAN^{(t)}\left(x_i\right)\right).$$

If $e^{(t)} \geq 0.5$, then set *T* = *t* – 1 and abort loop.

4) Set $\mu^{(t)} = e^{(t)} \Big/ \left(1 - e^{(t)}\right)$

5) Updating distribution $w_i^{(t+1)} = w_i^{(t)}\left(\mu^{(t)}\right)^s$, where $s = 1 - I\left(y_i \neq BAN^{(t)}\left(x_i\right)\right)$.

6) Normalize $w_i^{(t+1)}$, to sum to 1.

Output the final hypothesis:

$$H(x) = \arg\max_{y \in Y}\left(\sum_{t=1}^{T}\left(\log\left(\frac{1}{\mu^{(t)}}\right)\right) * I\left(y = BAN^{(t)}(x)\right)\right)$$

4. The Experimental Results

We conducted our experiments on a collection of machine learning datasets available from the UCI [10]. A summary of some of the properties of these datasets is given in **Table 1**. Some datasets are provided with a test set. For these, we reran each algorithm 20 times (since some of the algorithms are randomized), and averaged the results. For datasets with no provided test set, we used 10-fold cross validation, and averaged the results over 10 runs (for a total of 100 runs of each algorithm on each dataset).

In our experiments, we set the number of rounds of boosting to be *T* = 100.

The results of our experiments are shown in **Table 2**. The figures indicate test correct rate averaged over multiple runs of each algorithm. The bold in the table show that the classification is superior than another one obviously. From **Table 2** in the 20 datasets, Boosting-BAN did significantly and uniformly better than Boosting-MultiTAN.

On the data sets "Car", "Iris" and "LED", the Boosting-MultiTAN was inferior to the Boosting-BAN. The Boosting-BAN correct rate was better than the Boosting-MultiTAN correct rate in another 17 datasets. The reason is, in these cases the rate of attributes and classes are less than other Datasets. This reveals that the features in the three datasets are most dependent to each other. These weak dependencies can improve the prediction accuracy significantly, as we see from **Table 2**. These experiments also indicate that when the dataset is small and data loss, the boosting error rate is worse.

Table 1. Dataset used in the experiments.

No.	Dataset	Instances	Classes	Attributes	Missing values
1	Anneal	898	6	38	√
2	Audiology	226	24	69	√
3	Breast Cancer	699	2	9	×
4	Bupa	345	2	6	×
5	Car	1728	4	6	×
6	Cleveland	303	2	13	×
7	Crx	653	2	15	√
8	German	1000	2	20	×
9	House-votes-84	435	2	16	√
10	Hypothyroid	3163	2	25	√
11	Iris	150	3	4	×
12	Kr-rs-kp	3169	2	36	×
13	LED	1000	10	7	×
14	Mushroom	8124	2	22	√
15	Promoters	106	2	57	×
16	Segment	2310	7	19	×
17	Soybean Large	683	19	35	√
18	Tic-Tac-Toe	958	2	9	×
19	Wine	178	3	13	×
20	Zoology	101	7	16	×

Table 2. Experimental results.

No.	Dataset	Boosting-MultiTAN	Boosting-BAN
1	Anneal	98.3	**99.2**
2	Audiology	76.2	**78.8**
3	Breast Cancer	95.5	**95.9**
4	Bupa	58.5	**59.8**
5	Car	**87.1**	85.5
6	Cleveland	82.7	**84.6**
7	Crx	85.2	**86.5**
8	German	70.8	**74.6**
9	House-votes-84	94.9	**95.8**
10	Hypothyroid	99.1	**99.8**
11	Iris	**93.8**	90.5
12	Kr-rs-kp	93.3	**99.3**
13	LED	**73.9**	72.3
14	Mushroom	99.9	**100**
15	Promoters	89.3	**91.7**
16	Segment	94.3	**96.4**
17	Soybean Large	92.6	**93.7**
18	Tic-Tac-Toe	74.7	**79.2**
19	Wine	97.3	**98.5**
20	Zoology	96.8	**97.7**

5. Conclusions

GTAN and BAN classifiers are unstable, by setting different parameters, we can form a number of different TAN and BAN classifiers. In this paper, multiple TAN classifiers are combined by a combination method called Boosting-MultiTAN that is compared with the Boosting-BAN classifier which is boosting based on BAN combination. Finally, experimental results show that the Boosting-BAN has higher classification accuracy on most data sets.

When implementing Boosting classifiers, we were able to calculate the value of c directly given our prior knowledge. Of course, in a real situation we would be very unlikely to know the level of class noise in advance. It remains to be seen how difficult it would prove to estimate c in practice.

6. References

[1] Y. Freund and R. E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," *Computational Learning Theory*: 2*nd Euro-*

pean Conference (*EuroCOLT*'95), Barcelona, 13-15 March 1995, pp. 23-37.

[2] R. E. Schapire, Y. Freund, Y. Bartlett, *et al*., "Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods," In: D. H. Fisher, Ed., *Proceedings of the* 14*th International Conference on Machine Learning*, Morgan Kaufmann Publishers, San Francisco, 1997, pp. 322-330.

[3] Y. Freund, "Boosting a Weak Learning Algorithm by Majority," *Information and Computation*, Vol. 121, No. 2, 1995, pp. 256-285.

[4] J. R. Quinlan, "Bagging, Boosting, and C4.5," In: R. Ben-Eliyahu, Ed., *Proceedings of the* 13*th National Conference on Artificial Intelligence*, Portland, 4-8 August 1996, pp. 725-730.

[5] R. E. Schapire, "The Strength of Weak Learnability," *Machine Learning*, Vol. 5, No. 2, 1990, pp. 197-227.

[6] N. Friedman, D. Geiger and M. Goldszmidt, "Bayesian Network Classifiers," *Machine Learning*, Vol. 29, No. 2-3, 1997, pp. 131-163.

[7] J. Cheng and R. Greiner, "Comparing Bayesian Network Classifiers," In: K. B. Laskey and H. Prade, Ed., *Proceedings of the* 15*th Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann Publishers, San Francisco, 15 October 1999, pp. 101-108.

[8] J. Cheng, D. A. Bell and W. Liu, "An Algorithm for Bayesian Belief Network Construction from Data," In: *Proceedings of Conference on Artificial Intelligence and Statistics*, Lauderdale, January 1997, pp. 83-90.

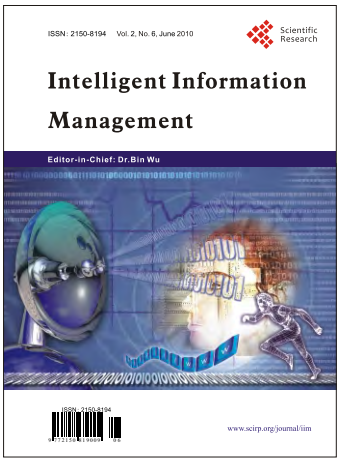
[9] H. B. Shi, H. K. Huang and Z. H. Wang, "Boosting-Based TAN Combination Classifier," *Journal of Computer Research and Development*, Vol. 41, No. 2, 2004, pp. 340-345.

[10] UCI Machine Learning Repository. http://www.ics.uci.edu/~mlearn/ML.Repository.html

[11] Y. Freund and R. E. Schapire, "Experiments with a New Boosting Algorithm," In: L. Saitta, Ed., P*roceedings of the* 13*th International Conference on Machine Learning*, Bari, 3-6 July 1996, pp. 148-156.

[12] X. W. Sun, "Augmented BAN Classifier," *Proceedings of International Conference on Computational Intelligence and Software Engineering*, Wuhan, 11-13 December 2009.

Call for Papers



Intelligent Information Management

ISSN 2150-8194 (print) ISSN 2150-8208 (online)
www.scirp.org/journal/iim

IIM is a peer reviewed international journal dedicated to the latest advancement of intelligent information management. The goal of this journal is to keep a record of the state-of-the-art research and promote the research work in these fast moving areas. The journal publishes the highest quality, original papers includ but are not limited to the fields:

- ★ Computational intelligence
- ★ Data mining
- ★ Database management
- ★ Distributed artificial intelligence
- ★ General systems theory and methodology
- ★ Information security
- ★ Intelligent information management systems
- ★ Knowledge discovery and management
- ★ Nonlinear system and control theory
- ★ Optimal algorithms
- ★ Other related areas and applications

We are also interested in short papers (letters) that clearly address a specific problem, and short survey or position papers that sketch the results or problems on a specific topic. Authors of selected short papers would be invited to write a regular paper on the same topic for future issues of the IIM.

Website and E-Mail

http://www.scirp.org/journal/iim **E-Mail:iim@scirp.org**

Intelligent Information Management, 2010, 2, 343-390.
Published Online June 2010 in SciRes (http://www.SciRP.org/journal/iim/).

TABLE OF CONTENTS

Volume 2 Number 6 **June 2010**

