THE BEST-SO-FAR ARTIFICIAL BEE COLONY ALGORITHM AND ITS APPLICATIONS

MR. ANAN BANHARNSAKUN

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
(ELECTRICAL AND COMPUTER ENGINEERING)
FACULTY OF ENGINEERING
KING MONGKUT'S UNIVERSITY OF TECHNOLOGY THONBURI
2011

The Best-so-far Artificial Bee Colony Algorithm and Its Applications

Mr. Anan Banharnsakun M.Eng. (Computer Engineering)

A Dissertation Submitted in Partial Fulfillment
of the Requirements for
the Degree of Doctor of Philosophy (Electrical and Computer Engineering)
Faculty of Engineering
King Mongkut's University of Technology Thonburi
2011

Dissertation Committee

…… (Assoc. Prof. Naruemon Wattanapongsakorn, Ph.D.)	Chairman of Dissertation Committee
…… (Assoc. Prof. Tiranee Achalakul, Ph.D.)	Member and Dissertation Advisor
…… (Assoc. Prof. Booncharoen Sirinaovakul, D.Eng)	Member and Dissertation Co-Advisor
…… (Asst. Prof. Songrit Maneewongvatana, Ph.D.)	Member
…… (Lect. Boonserm Kaewkamnerdpong, Ph.D.)	Member
…… (Lect. Tulaya Limpiti, Ph.D.)	Member

King Mongkut's University of Technology Thonburi
Agreement on Intellectual Property Rights Transfer for Postgraduate Students

Date…..31 October 2011……

Name……Mr. Anan……Middle Name…….…….
Surname/Family Name……Banharnsakun…...……..
Student Number……52530004……. who is a student of King's Mongkut's University of Technology Thonburi (KMUTT) in ❍ Graduate Diploma ❍ Master Degree
⊗ Doctoral Degree
Program ..Doctor of Philosophy.. Field of Study ..Electrical and Computer Engineering.
Faculty/School…… Faculty of Engineering …….
Home Address ……53/128 Moo 10, Suksawat Road, Bangkru, Phrapradaeng,……
…….Samutprakan …….
Postal Code…….10130……Country…… Thailand …….

I, as 'Transferer', hereby transfer the ownership of my thesis copyright to King's Mongkut's University of Technology Thonburi who has appointed Assoc. Prof. Dr. Piyabutr Wanichpongpan Associate Dean for Academic Affairs (Acting for Dean) to be 'Transferee' of copyright ownership under the 'Agreement' as follows.

1. I am the author of the thesis entitled… The Best-so-far Artificial Bee Colony Algorithm and Its Applications ……..
under the supervision of …… Assoc. Prof. Dr. Tiranee Achalakul …..……
who is my supervisor, and/or……. Assoc. Prof. Dr. Booncharoen Sirinaovakul.....
who is/are my co-supervisor(s), in accordance with the Thai Copyright Act B.E. 2537. The thesis is a part of the curriculum of KMUTT.

2. I hereby transfer the copyright ownership of all my works in the thesis to KMUTT throughout the copyright protection period in accordance with the Thai Copyright Act B.E. 2537, effective on the approval date of thesis proposal consented by KMUTT.

3. To have the thesis distributed in any form of media, I shall in each and every case stipulate the thesis as the work of KMUTT.

4. For my own distribution of thesis or the reproduction, adjustment, or distribution of thesis by the third party in accordance with the Thai Copyright Act B.E. 2537 with remuneration in return, I am subject to obtain a prior written permission from KMUTT.

5. To use any information from my thesis to make an invention or create any intellectual property works within ten (10) years from the date of signing this Agreement, I am subject to obtain prior written permission from KMUTT, and KMUTT is entitled to have intellectual property rights on such inventions or intellectual property works, including entitling to take royalty from licensing together with the distribution of any benefit deriving partly or wholly from the works in the future, conforming with the Regulation of King Mongkut's Institute of Technology Thonburi *Re* the Administration of Benefits deriving from Intellectual Property B.E. 2538.

6. If the benefits arise from my thesis or my intellectual property works owned by KMUTT, I shall be entitled to gain the benefits according to the allocation rate stated in the Regulation of King Mongkut's Institute of Technology Thonburi *Re* the Administration of Benefits deriving from Intellectual Property B.E. 2538.

Signature.....Transferor
(Mr. Anan Banharnsakun)
Student

Signature.....Transferee
(Assoc. Prof. Dr. Piyabutr Wanichpongpan)
Associate Dean for Academic Affairs (Acting for Dean)

Signature.....Witness
(Assoc. Prof. Dr. Tiranee Achalakul)

Signature.....Witness
(Assoc. Prof. Dr. Booncharoen Sirinaovakul)

Dissertation Title	The Best-so-far Artificial Bee Colony Algorithm and Its Applications
Dissertation Credits	36
Candidate	Mr. Anan Banharnsakun
Dissertation Advisor	Assoc. Prof. Dr. Tiranee Achalakul
Dissertation Co-Advisor	Assoc. Prof. Dr. Booncharoen Sirinaovakul
Program	Doctor of Philosophy
Field of Study	Electrical and Computer Engineering
Department	Computer Engineering
Faculty	Engineering
B.E.	2554

Abstract

Metaheuristic is one of the approximate methods that are widely used to solve practical optimization problems. Although metaheuristic approaches do not guarantee that an optimal solution can be found, they are effective and often used to search very large spaces of candidate solutions. Artificial Bee Colony (ABC) is a metaheuristic technique in which a colony of artificial bees cooperates in finding good solutions in optimal search space. The algorithm is one of the Swarm Intelligence algorithms explored in recent literature. However, ABC can sometimes be a slow technique to converge. In order to improve its performance and make it easier to apply to the practical problems, the optimization framework and the modified version of ABC called Best-so-far ABC were proposed in this dissertation. The Best-so-far ABC algorithm modifies the mechanism used to calculate new candidate food sources in order to improve the search capability of the onlooker bees. In each iteration, the scout bees in the Best-so-far ABC adjust the radius of the search for new candidates, using a larger radius in earlier iterations and progressively reducing the radius as the process comes closer to converging. A more robust calculation to determine and compare the quality of alternative solutions is also employed. The objective value of function is directly used instead of fitness value for comparison and selection of the better solution. These modifications allow the algorithm to converge more quickly with a bias towards the Best-so-far solution. The performance of these proposed methods were empirically assessed on several standard data sets and application domains including Image Registration, Job Shop Scheduling, and Clustering. The results demonstrated that the

proposed method can produce higher quality solutions with faster convergence than either the original ABC or the current state-of-the-art ABC-based algorithm. Moreover, in order to improve the performance for solving complex large-scale optimization problems, the distributed version of Best-so-far ABC was also presented. The results indicate that distributed Best-so-far ABC is effective in both the perspectives of solution quality and processing time required.

Keywords: Best-so-far Artificial Bee Colony (Best-so-far ABC)/ Swarm Intelligence/ Metaheuristic/ Optimization/ Parallel Computing/ Image Registration/ Job Shop Scheduling/ Clustering

หัวข้อวิทยานิพนธ์	อัลกอริทึมฝูงผึ้งประดิษฐ์แบบ เบสท์-โซ-ฟาร์ และการประยุกต์
หน่วยกิต	36
ผู้เขียน	นายอนันต์ บรรหารสกุล
อาจารย์ที่ปรึกษา	รศ. ดร. ธีรณี อจลากุล
อาจารย์ที่ปรึกษาร่วม	รศ. ดร. บุญเจริญ ศิริเนาวกุล
หลักสูตร	ปรัชญาดุษฎีบัณฑิต
สาขาวิชา	วิศวกรรมไฟฟ้าและคอมพิวเตอร์
ภาควิชา	วิศวกรรมคอมพิวเตอร์
คณะ	วิศวกรรมศาสตร์
พ.ศ.	2554

บทคัดย่อ

เมตาฮิวริสติก (Metaheuristic) เป็นหนึ่งในหลายวิธีที่ได้รับความนิยมและใช้กันอย่างแพร่หลายในการแก้ปัญหาการหาค่าที่เหมาะสมที่สุด (Optimization Problems) แม้ว่าวิธีนี้จะไม่ยืนยันว่าคำตอบที่ได้จะเป็นคำตอบที่เหมาะสมที่สุด แต่วิธีนี้ก็มีประสิทธิภาพในการแก้ปัญหาการหาค่าที่เหมาะสมสำหรับปัญหาที่มีขอบเขตการค้นหาขนาดใหญ่ได้ อัลกอริทึมฝูงผึ้งประดิษฐ์ (Artificial Bee Colony : ABC) เป็นวิธีแบบ เมตาฮิวริสติก ที่เลียนแบบมาจากพฤติกรรมการหาอาหารของผึ้งในธรรมชาติมาประยุกต์เป็นอัลกอริทึมที่ใช้ในการแก้ปัญหาการหาค่าที่เหมาะสมที่สุด อัลกอริทึมนี้เป็นแขนงหนึ่งในสาขาของ ปัญญาแบบกลุ่ม (Swarm Intelligence) ที่ได้รับความนิยมแพร่หลายในช่วงไม่กี่ปีที่ผ่านมา อย่างไรก็ตาม ในบางกรณีปัญหาการหาค่าที่เหมาะสมของอัลกอริทึมฝูงผึ้งประดิษฐ์ ให้ผลการลู่เข้าสู่คำตอบที่เหมาะสมค่อนข้างช้า ดังนั้น เพื่อเพิ่มประสิทธิภาพของอัลกอริทึมนี้ และเพื่อให้ง่ายต่อการประยุกต์ใช้อัลกอริทึมนี้กับปัญหาในรูปแบบต่างๆ งานวิจัยนี้จึงได้นำเสนอ กรอบความคิด และการปรับปรุงให้ผลการลู่เข้าสู่คำตอบที่เหมาะสมที่สุดของอัลกอริทึมฝูงผึ้งประดิษฐ์เร็วขึ้น เรียกว่า ฝูงผึ้งประดิษฐ์แบบ เบสท์-โซ-ฟาร์ (Best-so-far ABC) ซึ่งเกิดจากการปรับปรุงคุณลักษณะที่สำคัญของวิธีการหาคำตอบของอัลกอริทึมได้แก่ การปรับปรุงสมการที่ใช้ในการคำนวณค่าคำตอบโดยฝูงผึ้งเฝ้าดู (Onlooker bee) ให้มุ่งไปในทิศทางคำตอบที่ดีที่สุดที่พบในแต่ละรอบของการค้นหา การปรับขอบเขตการค้นหาคำตอบของผึ้งสำรวจ (Scout bee) โดยอนุญาตให้ผึ้งสำรวจสามารถค้นหาคำตอบได้ในขอบเขตที่กว้างในระยะแรก และควบคุมขอบเขตการค้นหาให้แคบลงในระยะหลังของการค้นหาคำตอบ และการเปรียบเทียบคำตอบโดยอ้างอิงการใช้ค่าเป้าหมาย (Objective value) โดยตรงแทนที่การใช้ค่าฟิตเนส (Fitness value) การปรับปรุงคุณลักษณะเหล่านี้ทำให้อัลกอริทึม ฝูงผึ้งประดิษฐ์แบบ เบสท์-โซ-ฟาร์ สามารถหาคำตอบที่ดีที่สุดได้ในระยะเวลาอันรวดเร็ว ซึ่งอัลกอริทึมที่ปรับปรุงนี้ได้ถูกนำไปใช้ในการแก้ปัญหาในเชิงปฏิบัติ เช่น การลงทะเบียนภาพ (Image Registration)

การจัดลำดับการทำงาน (Job Shop Scheduling) และการจัดกลุ่มข้อมูล (Clustering) ผลของงานวิจัยแสดงให้เห็นว่า การปรับปรุงด้วยวิธี เบสท์-โซ-ฟาร์ ทำให้อัลกอริทึมฝูงผึ้งประดิษฐ์ ให้ค่าของคำตอบ และ ความเร็วในค้นหาค่าที่เหมาะสมดีขึ้น นอกจากนี้เพื่อให้อัลกอริทึมนี้สามารถใช้แก้ปัญหาที่มีขนาดใหญ่มากในทางปฏิบัติได้ด้วย งานวิจัยนี้ยังได้ประยุกต์อัลกอริทึมนี้ให้สามารถประมวลผลแบบขนาน ผลลัพธ์ที่ได้แสดงให้เห็นว่า การประมวลผลแบบขนานของอัลกอริทึมฝูงผึ้งประดิษฐ์แบบเบสท์-โซ-ฟาร์ ให้ผลที่เป็นที่น่าพอใจ ทั้งในแง่ คุณภาพของคำตอบ และ เวลาที่ใช้ในการคำนวณ

คำสำคัญ : อัลกอริทึมฝูงผึ้งประดิษฐ์แบบ เบสท์-โซ-ฟาร์/ ปัญญาแบบกลุ่ม/ เมตาฮิวริสติก / การหาค่าที่เหมาะสมที่สุด/ การประมวลผลแบบขนาน/ การลงทะเบียนภาพ/ การจัดลำดับการทำงาน/ การจัดกลุ่ม

ACKNOWLEDGEMENTS

First of all, I would like to gratefully acknowledge Assoc. Prof. Dr. Tiranee Achalakul and Assoc. Prof. Dr. Booncharoen Sirinaovakul, my advisor and co-advisor, who always give me ideas, many valuable comments and contributions during the entire period of work. I also give my gratitude to all the lecturers, faculty members, and CASTLAB members at the Computer Engineering Department for the knowledge and the experience provided through in this Doctor of Philosophy Program. I also appreciate the Thailand Research Fund through the Royal Golden Jubilee Ph.D. Program for financial support during my doctoral study.

I offer the special thanks to Supannee Tanathong, who has always been a source of inspiration for me in pursuit of my Ph.D. Finally, I would like to express my best gratitude to my family, especially to my parents who always encourage and take care of me in all possible means.

CONTENTS

		PAGE
ENGLISH ABSTRACT		iii
THAI ABSTRACT		v
ACKNOWLEDGEMENTS		vii
CONTENTS		viii
LIST OF TABLES		x
LIST OF FIGURES		xi
LIST OF SYMBOLS		xiii
LIST OF TECHNICAL VOCABULARY AND ABBREVIATIONS		xiv
CHAPTER		
1. INTRODUCTION		1
1.1	Objectives	3
1.2	Contributions	4
1.3	Chapter Overviews	5
2. BACKGROUNDS AND REVIEWS		6
2.1	Backgrounds	6
2.1.1	Bees' Foraging Behavior	6
2.1.2	The Original ABC Metaheuristic Algorithm	6
2.2	Review of the Algorithms Based on Bee Foraging Behavior	9
2.2.1	Artificial Bee Colony	10
2.2.2	Bee Algorithm	11
2.2.3	Virtual Bee Algorithm	12
2.2.4	Elite Bee	12
2.2.5	Bee Swarm Optimization	12
2.2.6	Summary of the Algorithms Based on Bee Foraging Behavior	13
2.3	Current Applications of ABC Metaheuristic	13
2.3.1	Generalized Assignment Problem	14
2.3.2	Flow Shop Scheduling Problem	15
2.3.3	The Leaf-constrained Minimum Spanning Tree Problem	15
2.3.4	Training Neural Networks Problem	16
2.3.5	Data Clustering Problem	17
2.3.6	Pattern Recognition Problem	18
3. ARTIFICIAL BEE COLONY OPTIMIZATION FRAMEWORK		20
3.1	Optimization Model	20
3.2	ABC Based Optimization Framework	21
3.2.1	Objective Determination	21
3.2.2	Parameters Mapping	21
3.2.3	Balancing of Exploitation and Exploration	23
3.3	The Example of the ABC Metaheuristic Representation	25
3.3.1	The Traveling Salesman Problem (TSP)	25
3.3.2	Dimension Reduction in Bioinformatics Data	30

4. THE BEST-SO-FAR ABC ALGORITHM 34
4.1 The Ideas of the Best-so-far ABC Algorithm 34
4.1.1 The Best-so-far Method 37
4.1.2 The Adjustable Search Radius 38
4.1.3 Objective-value-based Comparison Method 40
4.2 Sensitivity of the Parameters Setting 41
4.3 Computational Complexity 46

5. ALGORITHM COMPARISON AND EVALUATION 48
5.1 Numerical Experiment Methodology 48
5.2 Numerical Results and Discussion 49

6. BEST-SO-FAR ABC ON DISTRIBUTED ENVIRONMENTS 56
6.1 Best-so-far ABC with Multiple Patrilines 56
6.2 Performance Scalability 62
6.2.1 The Evaluation of Solution Accuracy 63
6.2.2 The Evaluation of Algorithm's Efficiency 63

7. APPLYING BEST-SO-FAR ABC TO PRACTICAL APPLICATIONS 66
7.1 Image Registration 66
7.1.1 Background 66
7.1.2 Optimization Solution with Best-so-far ABC 68
7.1.3 Image Registration Application Experiment Methodology 70
7.1.4 Image Registration Results and Discussion 71
7.2 Job Shop Scheduling 75
7.2.1 Background 75
7.2.2 Mapping Best-so-far ABC Algorithm to the JSSP 77
7.2.3 Job Shop Scheduling Experiment Methodology 85
7.2.4 Job Shop Scheduling Results and Discussion 86
7.3 Clustering Application 91
7.3.1 Background 91
7.3.2 Best-so-far ABC Based Multiple Patrilines for Clustering 94
7.3.3 Clustering Experiments and Discussion 96
7.3.3.1 Solution Quality 98
7.3.3.2 Performance and Scalability 102

8. CONCLUSION AND FUTURE WORK 108

REFERENCES 112

APPENDIX
A Experimental Results from ABC with the Traveling Salesman Problem (TSP) 122
B Experimental Results from ABC with Dimension Reduction in Bioinformatics Data 126
C Experimental Results from the Effect of Proposed Methods and Sensitivity of the Parameters Setting in Best-so-far ABC 135

CURRICULUM VITAE 143

LIST OF TABLES

TABLE		PAGE
2.1	Current application of ABC algorithm	14
3.1	Example of objective determination	21
3.2	Examples of parameter mapping	23
4.1	Numerical benchmark functions	36
4.2	Mean of best function values obtained for 1000 iterations from Best-so-far ABC under different colony sizes	42
4.3	Sensitivity of the parameters setting	45
5.1	Result obtained in the ABC1 experiment	50
5.2	Result obtained in the ABC2 experiment	51
5.3	Rates of convergence in the ABC2 experiment	54
6.1	Solution quality obtained from Best-so-far ABC with multiple patrilines	63
6.2	Speedup and efficiency for given number of processors	64
7.1	The search space of each variable for rigid transformation process	70
7.2	The results obtained from the original ABC and best-so-far ABC method based on fixed initial solutions at approximately the same runtime value	72
7.3	The results obtained from the original ABC and best-so-far ABC method based on fixed initial solutions at approximately the same mean MI value	72
7.4	The results obtained from the original ABC and best-so-far ABC method based on random initial solutions by using approximately the same runtime value	73
7.5	The results obtained from the original ABC and best-so-far ABC method based on random initial solutions by using approximately the same mean MI value	74
7.6	An example of job processing time on each machine for 3-job x 3-machine	79
7.7	An example of job machine sequence for 3-job x 3-machine	79
7.8	Comparison results of the Best-so-far ABC and IABC	86
7.9	Comparison results of the Best-so-far ABC, MPSO, and HIA	88
7.10	Comparison results of the Best-so-far ABC, HEA, and HGA-Param	89
7.11	Comparison results of the Best-so-far ABC and PSO-VNS	90
7.12	Characteristic of data sets considered	98
7.13	Results obtained by the algorithms runs on Iris	100
7.14	Results obtained by the algorithms runs on CMC	100
7.15	Results obtained by the algorithms runs on Wine	101
7.16	Results obtained by the algorithms runs on Vowel	101
7.17	Speedup and efficiency results of variation in number of attributes	103
7.18	Speedup and efficiency results of variation in number of data records	105
A.1	Number of iterations used in ABC-GSX, ACO-PSO and BCO algorithms	123
A.2	Results of the first TSP experiment	124
A.3	Results of the second TSP experiment	124
B.1	The characteristics of data sets considered.	128
B.2	Comparison results of the ABC-kNN and LS-SVM, PCA-FDA, MSDR-LGC, LLDE-kNN	129
B.3	Comparison results of the ABC-kNN and IBPSO-kNN	131
C.1	Mean of best function values from ABC with different proposed method	136

LIST OF FIGURES

FIGURE **PAGE**

1.1	General optimization paradigm	1
2.1	ABC metaheuristic procedure	9
2.2	The behavior of honey bee foraging for nectar	10
2.3	The structure of the bee swarm and the flying patterns of the bees	12
3.1	Optimization model	20
3.2	Exploitation and exploration in ABC	24
3.3	The ABC algorithm flowchart for TSP	26
3.4	The mapping of the food sources and the sequence of the tour	26
3.5	Greedy subtour crossover method	27
3.6	Example of greedy subtour crossover method	28
3.7	The 2-opt method	29
3.8	The mapping of the food sources and a feasible set of features	30
3.9	The flowchart of ABC-kNN method	31
4.1	The pseudo-code of the best-so-far ABC algorithm	35
4.2	Iterations to convergence for ABC and ABC with best-so-far method (ABC with BSF)	38
4.3	Iterations to convergence for ABC and ABC with adjustable search radius (ABC with ASR)	39
4.4	The comparison between the old and the new method for comparing the solutions	40
4.5	Iterations to convergence for ABC and ABC with objective-value-based comparison method (ABC with OBC)	41
4.6	Iterations to convergence on Sphere and Griewank	42
4.7	Iterations to convergence on Rastrigin and Rosenbrock	43
4.8	Iterations to convergence on Ackley and Schaffer	43
4.9	Iterations to convergence under different “limit” values on Griewank	44
5.1	Iterations to convergence for original and best-so-far algorithms when dimension = 30 (dimension of Schaffer Function = 2)	53
6.1	The patriline cube	57
6.2	The information exchange method	58
6.3	The Best-so-far ABC with multiple patrilines flowchart	59
6.4	Scalability Plot	64
7.1	Automated image registration based on the best-so-far ABC method	69
7.2	Image Pair I: Brain image for registration	71
7.3	Image Pair II: Dentistry image for registration	71
7.4	Image Pair III: Satellite image for registration	71
7.5	Image Pair IV: Camera Man image for registration	72
7.6	The mean of the mutual information across iterations in Image Pair I-IV based on fixed initial solutions	73
7.7	The mean of the mutual information across iterations in Image Pair I-IV based on random initial solutions	74
7.8	The Best-so-far ABC algorithm flowchart for JSSP	78
7.9	The example of operation scheduling list representation for 3-job x 3-machine	79

7.10	The feasible schedule constructed from the operation scheduling list in Figure 7.9	80
7.11	The mapping between the food sources and operation scheduling list	80
7.12	Exchanging information with a neighboring food source based on PBX method	81
7.13	The updating food source based on radius search by scout bee to find v_{ij}	82
7.14	Variable Neighboring Search (VNS) Procedure	83
7.15	The exchanging process in VNS method (for new x_b)	84
7.16	The inserting process in VNS method (for new x_b)	84
7.17	Varying the attributes size with 800 data records	103
7.18	Comparing the ideal time to the actual time on various numbers of attributes	104
7.19	Varying the data record size with 30 attributes	106
7.20	Comparing the ideal time to the actual time on v various data record size	106
B.1	The number of iterations vs. genes selected and classification accuracy on Acute_Leukemia dataset	130
B.2	The number of iterations vs. genes selected and classification accuracy on Colon_Cancer and Hepatocellular_Carcinoma datasets	130
B.3	The number of iterations vs. genes selected and classification accuracy on High_Grade_Glioma and Prostate_Cancer datasets	130
B.4	Number of genes selected for each of the 5 datasets obtained from ABC-kNN	131
B.5	The number of iterations vs. genes selected and classification accuracy on Leukemia1and Leukemia2 datasets	132
B.6	The number of iterations vs. genes selected and classification accuracy on SRBCT and DLBCL datasets	132
B.7	The number of iterations vs. genes selected and classification accuracy on Brain_Tumor1 and Brain_Tumor2 datasets	133
B.8	The number of iterations vs. genes selected and classification accuracy on Lung_Cancer and Prostate_Tumor datasets	133
B.9	The number of iterations vs. genes selected and classification accuracy on 9_Tumors, 11_Tumors, and 14_Tumors datasets	133
B.10	Number of genes (features) selected for each of the 11 datasets obtained from ABC-kNN versus IBPSO-kNN	134
C.1	The effect of proposed methods on Sphere and Griewank	136
C.2	The effect of proposed methods on Rastrigin and Rosenbrock	137
C.3	The effect of proposed methods on Ackley and Schaffer	137
C.4	Iterations to convergence under different “limit” values on Sphere	138
C.5	Iterations to convergence under different “limit” values on Rastrigin	139
C.6	Iterations to convergence under different “limit” values on Rosenbrock	140
C.7	Iterations to convergence under different “limit” values on Ackley	141
C.8	Iterations to convergence under different “limit” values on Schaffer	142

LIST OF SYMBOLS

C	=	Makespan Value
D	=	Dimensions of solution, Sum of squared Euclidean distance
E	=	A finite set of connections that fully connects V
$F(x_i)$	=	The objective function
f_b	=	The fitness value of the best-so-far food source
$f()$	=	The fitness function
fit_i	=	The fitness value of the food source i
G	=	The completely connected graph that represents the problem's search space
$H()$	=	Shannon's entropy function
I	=	Image
J	=	Job
k	=	The number of clusters
m	=	The number of machines, The number of processor cores
n	=	The number of jobs
M	=	Machine
MCN	=	The maximum number of cycles
N_v	=	The number of sub-solutions
$p()$	=	The marginal probability mass functions
P_i	=	The probability that a food source i will be selected
R^D	=	A Search Space in real number value
s	=	The candidate food source
s^*	=	A globally optimal feasible food source
S	=	The set of candidate food sources
$\tilde{S}$	=	The set of feasible food sources
SN	=	The number of food sources
S_p	=	Speedup
V	=	A finite set of sub-solutions
x	=	A candidate set of sub-solutions
x_{bj}	=	The best-so-far food source in selected dimension j
x_i	=	A position of food source i
x_{ij}	=	A previous solution i in dimension j
x_j^{min}	=	The lower bound of the food source position in dimension j
x_j^{max}	=	The upper bound of the food source position in dimension j
x_{kj}	=	A neighboring solution k in dimension j
x_{new}	=	A new solution
x_{old}	=	An old solution
$\mathcal{X}$	=	The set of all possible solutions in the graph G
$\tilde{\mathcal{X}}$	=	A set of feasible solutions
v_{ij}	=	A new feasible solution i in dimension j
$\emptyset_{ij}$	=	A random number between [-1,1]
Φ	=	A random number between [-1,1]
Ω	=	A set of constraints
ω_{max}	=	The maximum percentage of the position adjustment for the scout bee
ω_{min}	=	The minimum percentage of the position adjustment for the scout bee

LIST OF TECHNICAL VOCABULARY AND ABBREVIATIONS

ABC	=	Artificial Bee Colony
ACO	=	Ant Colony Optimization
AIS	=	Artificial Immune System
BA	=	Bee Algorithm
BCO	=	Bee Colony Optimization
BKS	=	Best Known Solution
BSO	=	Bee Swarm Optimization
DE	=	Differential Evolution
E	=	Efficiency
ES	=	Evolution Strategies
GA	=	Genetic Algorithm
GSX	=	Greedy Subtour Crossover
HEA	=	Hybrid Evolutionary Algorithm
HGA	=	Hybrid Genetic Algorithm
HIA	=	Hybrid Intelligent Algorithm
IABC	=	Improved Artificial Bee Colony
JSSP	=	Job Shop Scheduling Problem
KNN	=	K-Nearest Neighbors
MCN	=	Maximum Cycle Number
MI	=	Mutual Information
MPI	=	Message Passing Interface
MPSO	=	Multi-type Individual Enhancement PSO
NM	=	Nelder-Mead
OR	=	Operation Research
PBX	=	Position Base Crossover
PS-EA	=	Particle Swarm Inspired Evolutionary Algorithm
PSO	=	Particle Swarm Optimization
RPE	=	Relative Percent Error
SA	=	Simulated Annealing
S.D.	=	Standard Deviation
TSP	=	Traveling salesman problem
VBA	=	Virtual Bee Algorithm
VNS	=	Variable Neighboring Search

CHAPTER 1 INTRODUCTION

An optimization problem is a problem of finding the best solution from all feasible solutions. It can be categorized into two major groups: combinatorial and numerical optimization problems. Most of such problems are considered *NP*-hard; it is strongly believed that they cannot be solved to optimality within polynomial computation time. Therefore, In order to solve these problems, previous research tends to employ an approximation that finds a near-optimal solution in a reasonable amount of time rather than a method that is guaranteed to find the optimal solution in an exponential time.

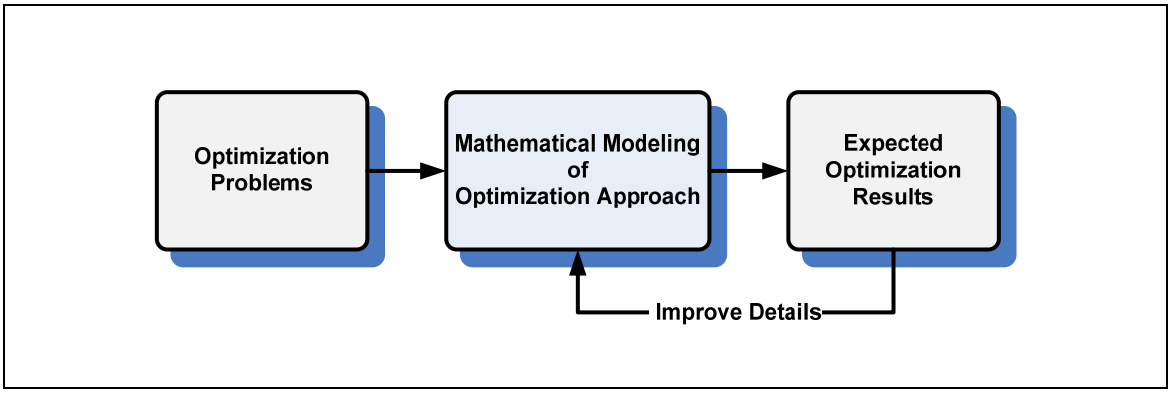


Figure 1.1 General Optimization Paradigm

Figure 1.1 illustrates a general optimization paradigm that is used to develop an optimization model for solving the optimization problems. Generally, to achieve the expected optimization results, a design of the optimization model should be considered on two search strategies including exploration and exploitation.

Exploration and exploitation are the important mechanisms in a robust search process. While exploration process is related on the independent search for an optimal solution, exploitation uses existing knowledge to bias the search.

Metaheuristics (Yang, 2008) are one of many approximation methods widely used to solve practical optimization problems. It is a set of concepts that can be used to define heuristic methods applicable to a wide range of problems. It is a general algorithmic framework that can be applied to solve problems in various domains with relatively few modifications to make them adaptable to a specific problem.

Metaheuristic approaches are defined as high-level strategies that guide the search process. They consist of different intelligent concepts for exploring and exploiting search spaces. In these concepts, probabilistic decision that relies on randomness is used during the search process. However, the randomness is not used blindly but in intelligently biased forms such as descent bias (based on objective functions), memory bias (based on previously made decisions) and experience bias (based on prior performance). By effects of bias, the high quality solutions can be found quickly.

Several methods are considered metaheuristic. They are such as Simulated Annealing (SA) (Kirkpatrick, et al., 1983), Tabu Search (Glover and Laguna, 1997), Iterated Local Search (Lourenc, et al., 2002), Evolutionary Computation (Fogel, 2006). Swarm Intelligence, emerging in the last decades, is also a metaheuristic method in the field of artificial intelligence, used to solve optimization problems. It is based on the collective behavior of social insects, flocks of birds, or schools of fish. These animals can solve complex tasks without having a centralized control unit.

Researchers have analyzed such behaviors and designed algorithms that can be used to solve combinatorial and numerical optimization problems in many science and engineering domains. Previous research (Aghdam, et al., 2009; Yagmahan and Yenisey, 2008; Perez and Behdinan, 2007; Yin, 2006) has shown that algorithms based on Swarm Intelligence have great potential. The algorithms that have emerged in recent years include Ant Colony Optimization (ACO) (Dorigo and Stützle, 2004), based on the foraging behavior of ants, and Particle Swarm Optimization (PSO) (Kennedy and Eberhart, 1995), based on the behaviors of bird flocks and fish schools.

In the recent years, there are new series of algorithms designed based on bee foraging behavior. The algorithms are developed to improve both exploration and exploitation for solving the numerical optimization problems. The examples are Artificial Bee Colony (ABC) (Karaboga, 2005), the Bee Algorithm (BA) (Pham, et al., 2006), the Virtual Bee Algorithm (VBA) (Yang, 2005), the Elite Bee Method (Sundareswaran and Sreedevi, 2008) and the Bee Swarm Optimization (BSO) (Akbari, et al., 2010).

The experimental results from these algorithms show that the algorithms based on bee foraging behavior can be adopted successfully to solve numerical optimization problems. However, in some cases the convergence speed can be too slow.

This dissertation proposes an optimization framework based on the ABC metaheuristic model and a modified version of the algorithm based on bee foraging behavior, called Best-so-far ABC, in order to improve the algorithm's performance.

To validate and evaluate the model, we performed the analysis of the model as well as empirically assessed the performance of proposed work on both numerical and application experiments to demonstrate the universal and practical applications of the best-so-far ABC model.

Moreover, in many real-world problems, there exist a large computational and resource demands and a single processor may not be computationally sufficient. In this research, we also designed a distributed version of this algorithm to obtain the enhanced computational throughput and the global search capability for solving complex large-scale optimization problems.

The proposed algorithm was applied to applications in various domains including the job shop scheduling, the image registration, and clustering. The results were evaluated in both the perspective of solution quality and execution performance.

1.1 Objectives

1. Develop a mathematical model of best-so-far approach based on the Artificial Bee Colony (ABC) algorithm for solving the optimization problems.
2. Study the algorithm performance by the analysis of the computational complexity and the rate of convergence.
3. Develop a computational framework capable for solving the practical problems based on a sequential and parallel system.
4. Validate and evaluate the model numerically by measuring the accurate result and convergent speed between the original model and the best-so-far model.
5. Apply the algorithm to an application in job shop scheduling, the image registration, and clustering.

1.2 Contributions

1. This dissertation reports on a modified version of Artificial Bee Colony (ABC) algorithm called Best-so-far ABC. The novel algorithm improves solution quality and convergence speed of the original and many variations of ABC for optimization problems. Three major changes are introduced, namely; the best-so-far method, the adjustable search radius, and objective-value-based comparison method. The proposed ABC modifications improve both the exploration and the exploitation search of the bee agents in the algorithm. In addition, the solution updating model for combinatorial optimization problems is addressed. We also studied the sensitivity of the input parameters, such as the number of bee agent in order to analyze the Best-so-far framework in depth.
2. The ABC optimization framework described based on the use of the set theory is introduced in this thesis. The framework allows the researchers, who are interested in ABC adoption, to better understand the process of problem mapping. In other words, the researchers will be able to easily understand how to derive objective function(s) and identify appropriate fitness value(s). Several examples of problem mapping for standard algorithms and specialized application domains are given.
3. The performance of the Best-so-far ABC technique is compared with the results from the original ABC and the state-of-the-art algorithm variations using numerical benchmarking functions and complex optimization problems including image registration and job shop scheduling. All the experimental results are favorable.
4. Distributed computing framework for Best-so-far ABC based on multiple patrilines concept is presented and evaluated. Hypercube model is used for information exchanging among patrilines. A large scale optimization problem in data clustering is used to access the performance. The multiple patrilines version is proved to be fast and effective. The algorithm is able to solve the clustering problems within a reasonable amount of time, while still keeping good solution quality.

1.3 Chapter Overview

This dissertation consists of eight Chapters. This Chapter discusses the need of a modified version of the optimization algorithm based on bee foraging behavior. It also states the objectives and contributions of this study. Chapter 2 presents some backgrounds which are crucial to the current work and provides a review of relevant literatures in the optimization algorithms based on bee foraging behavior. Chapter 3 introduces the Artificial Bee Colony algorithm based optimization framework. Chapter 4 proposes a modified version of optimization method based on the best-so-far selection in Artificial Bee Colony algorithm. Chapter 5 presents the initial experiments and discusses the performance results of the proposed method. Chapter 6 introduces a distributed version of the Best-so-far ABC algorithm. Chapter 7 describes the adoption of the Best-so-far ABC on practical applications. Finally, Chapter 8, a summary of the work is given.

CHAPTER 2 BACKGROUND AND REVIEW

This Chapter lays out the basic background of this dissertation. The related works in the improvement of optimization algorithms based on bee foraging behavior and their current applications are thoroughly reviewed. However, the contents in this chapter are merely the summaries of knowledge. For further details in each topic, please refer to more comprehensive references given at the end of this book.

2.1 Backgrounds

2.1.1 Bees' Foraging Behavior

Foraging (Biesmeijer and Seeley, 2005) is described as how honeybees find food sources. In this process, quality food sources are selected based on group decision making by the swarm. The important factors can be identified as independence and interdependence in collective decision making in this mechanism.

The bees independently evaluate the quality of different new candidate food sources on their own. However, the interdependence among them makes them more attentive to candidate food sources discovered and advertised by others. Waggle dances which are done by employed bees in the food source selection process are used to exchange information on new candidate food sources and to recruit unemployed bees to follow employed bees to their sources. Through this kind of information exchange and learning, the honeybee swarm manages to discover the highest quality food sources.

2.1.2 The Original ABC Metaheuristic Algorithm

Artificial Bee Colony (ABC) is a metaheuristic in which artificial bees of a colony cooperate in finding good solutions to problems of optimization. A characteristic of ABC is that it is inspired by nature, or more precisely by the behavior of honey bees seeking a quality food source. ABC algorithm takes the concepts from this foraging process to discover good solutions in an optimization problem. Essential components in ABC modeled after the foraging process are defined as follows:

- ***Food Source***: This component represents a feasible solution in an optimization problem.
- ***Fitness Value***: The value represents the *profitability* of a food source. For simplicity, it is represented as a single quantity associated with the objective function of a feasible solution.
- ***Bee Agents***: This component is a set of computational agents. Honey bees in ABC are categorized into three groups: employed bees, onlooker bees and scout bees. The colony is equally separated into employed bees and onlooker bees. Each solution in the search space consists of a set of optimization parameters which represent a food source's location. The number of employed bees and the number of food sources are equal. In other words, there would be one employed bee for each food source.

The employed bees will be responsible for investigating their food sources and sharing information about these food sources to recruit the onlooker bees. The onlooker bees will make a decision to choose a food source based on this information. The food source with higher quality will have a larger chance to be selected by onlooker bees than lower quality ones. An employed bee whose food source is rejected as low quality by employed and onlooker bees will change to a scout bee to search randomly for new food sources. By this mechanism, the exploitation will be handled by employed and onlooker bees while the exploration will be maintained by scout bees. The details of the algorithm are as follows.

First, randomly distributed initial food source positions are generated. The process can be represented by equation 2.1. After initialization, the population is subjected to repeated cycles of three major steps: updating feasible solutions, selecting feasible solutions, and avoiding suboptimal solutions.

$$F(x_i)\,, x_i \in R^D\,, i \in \{1,2,3..,SN\}, \tag{2.1}$$

x_i is a position of food source as a D-dimensional vector, $F(x_i)$ is the objective function which determines how good a solution is, and *SN* is the number of food sources.

After initialization, the population is subjected to repeated cycles of four major steps: updating feasible solutions by employed bees, selection of feasible solutions by onlooker bees, updating feasible solutions by onlooker bees, and avoidance of suboptimal solutions by scout bees.

Updating feasible solutions by employed bees

The position of the new feasible food source discovered by an employed bee is calculated from equation (2.2) below.

$$v_{ij} = x_{ij} + \emptyset_{ij}(x_{ij} - x_{kj}) \tag{2.2}$$

In the equation (2.2), v_{ij} is a new feasible solution that is modified from its previous solution value (x_{ij}) based on a comparison with the randomly selected position from its neighboring solution (x_{kj}). $\emptyset_{ij}$ is a random number between [-1,1] which is used to adjust the old solution to become a new solution in the next iteration. $k \in \{1,2,3..,SN\} \wedge k \neq i$ and $j \in \{1,2,3..,D\}$ are randomly chosen indexes. The difference between x_{ij} and x_{kj} is a difference of position in a particular dimension.

If a new food source (v_{ij}) is better than an old food source (x_{ij}), the old food source is replaced by the new food source.

Selection of feasible solutions by onlooker bees

When the employed bees return to their hive, they share information with the onlooker bees about candidate solutions they found. The onlooker bees select these solutions based on probability. Solutions of higher fitness have a larger chance of being selected by onlooker bees than ones of lower fitness. The probability that a food source will be selected can be obtained from equation (2.3) below.

$$P_i = \frac{\text{fit}_i}{\sum_{n=1}^{SN} \text{fit}_n} \tag{2.3}$$

Where fit_i is the fitness value of the food source *i,* which is related to the objective function value ($F(x_i)$) of the food source *i*.

Updating feasible solutions by onlooker bees

Based on the information obtained from the employed bees, the onlooker bees select their feasible food sources. The selected food sources are then updated using equation (2.2), i.e. an old food source is replaced by a new food source if the new food source is of a better quality.

Avoidance of suboptimal solutions by scout bees

This step is done by reassigning employed bees whose contributions are rejected as low quality to become scout bees who will randomly search for new solutions. The new random position chosen by the scout bee will be calculated from equation (2.4) below.

$$x_{ij} = x_j^{min} + rand[0,1] * \left(x_j^{max} - x_j^{min}\right) \quad (2.4)$$

Where x_j^{min} is the lower bound of the food source position in dimension j and x_j^{max} is the upper bound of the food source position in dimension j.

These four major steps mentioned above are repeated until an optimal solution is found or the number of iteration (cycle) reaches the termination criteria, MCN (Karaboga and Basturk, 2007).

The processes of the ABC metaheuristic can be shown in pseudo-code as Figure 2.1.

```
Procedure ABC_Metaheuristic
        Initial_Solutions
        While (criterion)
                Update_Feasible_Solutions (Employed bees)
                Select_Feasible_Solutions (Onlooker bees)
                Update_Feasible_Solutions (Onlooker bees)
                Avoid_Sub-Optimal_Solutions (Scout bee)
        End while
End-Procedure
```

Figure 2.1 ABC Metaheuristic Procedure

2.2 Review of the Algorithms Based on Bee Foraging Behavior

In recent years, many algorithms mimicking the food foraging behavior of swarms of honey bees were developed to solve optimization problems. Examples in these algorithms are the Artificial Bee Colony (ABC) (Karaboga, 2005), the Bee Algorithm (BA) (Pham, et al., 2006), the Virtual Bee Algorithm (VBA) (Yang, 2005), the Elite Bee Method (Sundareswaran and Sreedevi, 2008) and the Bee Swarm Optimization (BSO) (Akbari, et al., 2010).

2.2.1 Artificial Bee Colony (Karaboga, 2005)

The Artificial Bee Colony (ABC) algorithm is one approach that has been used to find an optimal solution in numerical optimization problems. This algorithm is inspired by the behavior of honey bees when seeking a quality food source shown as Figure 2.2 below.

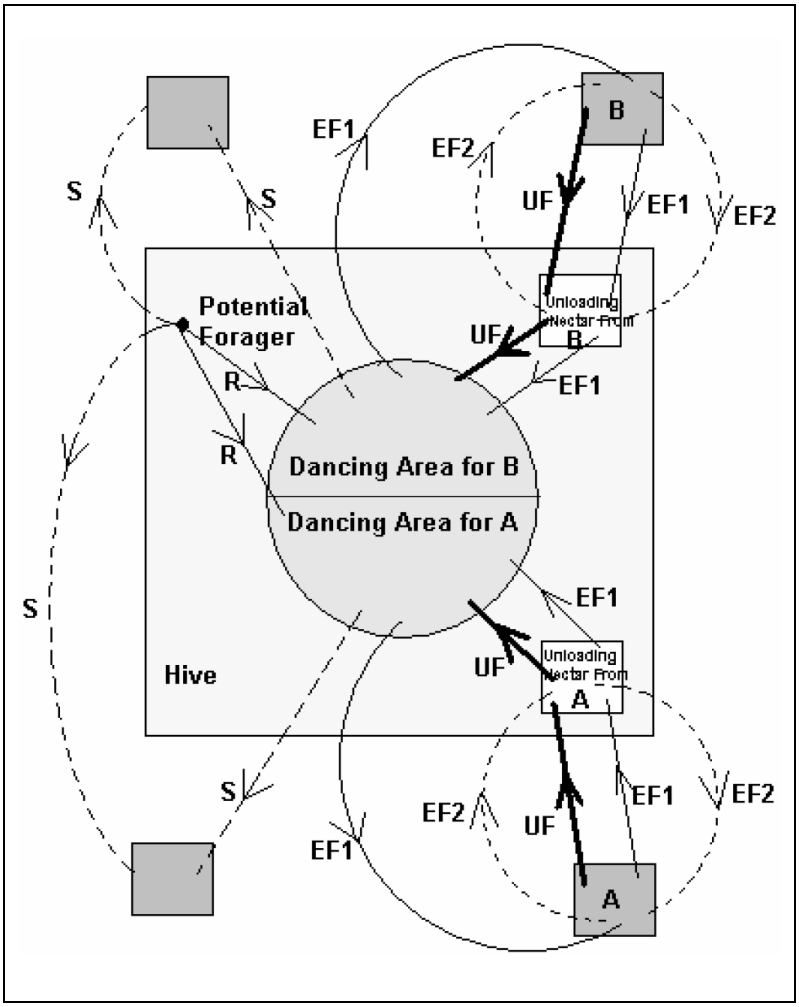


Figure 2.2 The behavior of honey bee foraging for nectar (Karaboga, 2005)

For Figure 2.2, assume that there are two discovered food sources: A and B. At the first step, a potential forager will start as unemployed forager. That bee will have no knowledge about the food sources around the hive. There are two possible options for such bee:

1. It can be a scout and starts searching around the nest (S on Figure 2.2).
2. It can be a recruit after watching the waggle dances and starts searching for a food source (R on Figure 2.2).

After the food sources are found, the bee utilizes its own capability to memorize the location and then immediately starts exploiting it. Hence, the bee will become an "employed forager". The foraging bee takes a load of nectar from the source and returns to the hive and unloads the nectar to a food store. After unloading the food, the bee has three following options:

1. It becomes an uncommitted follower after abandoning the food source (UF).
2. It dances and then recruits hive mates before returning to the same food source (EF1)
3. It continues to forage at the food source without recruiting other bees (EF2).

The performance of ABC algorithm has been compared with other optimization methods such as Genetic Algorithm (GA), Differential Evolution algorithm (DE), Evolution Strategies (ES), Particle Swarm Optimization (PSO), and Particle Swarm Inspired Evolutionary Algorithm (PS-EA) (Karaboga and Akay, 2009a; Karaboga and Basturk, 2008; Karaboga and Basturk, 2007). The comparisons were made based on various numerical benchmark functions, which consist of unimodal and multimodal distributions. The comparison results showed that ABC can produce a more optimal solution and thus is more effective than the other methods in several optimization problems (Singh, 2009; Kang, et al., 2009; Karaboga, 2009).

2.2.2 Bee Algorithm (Pham, et al., 2006)

A kind of neighborhood search combined with random search inspired the proposal of the Bee Algorithm. The algorithm starts with the scout bees being placed randomly in the search space. The fitness of the each site visited by the scout bees is then evaluated. Bees with the highest fitness are chosen as the "selected bees" and sites visited by them are chosen for neighborhood search. Then, the algorithm conducts searches in the neighborhood of the selected sites, assigning more bees to search near to the best sites. The fitness values are used to determine the probability of the bees being assigned. Searches in the neighborhood of the best sites that represent more promising solutions are made more detailed by recruiting more bees to follow them than the other selected bees. The remaining bees in the population are assigned randomly around the search space scouting for new potential solutions. These steps are repeated until a stopping criterion is met.

2.2.3 Virtual Bee Algorithm (Yang, 2005)

An algorithm called the Virtual Bee Algorithm (VBA) was introduced for solving engineering optimizations that have multi-peaked functions. In the VBA algorithm, the objectives or optimization functions are encoded as virtual foods. Virtual bees are used to search for virtual foods in the search space. The position of each virtual bee is updated via the virtual pheromone from the neighboring bees. The food with largest number of virtual bees or intensity of visiting bees corresponds to the optimal solution. However, the VBA algorithm was only tested using two-dimension functions.

2.2.4 Elite Bee (Sundareswaran and Sreedevi, 2008)

An optimization algorithm inspired by the honey bee foraging behavior based on the elite bee method was proposed to improve the algorithm performance. The bee whose solution is the best possible solution in each simulation iteration is considered to be the elite bee. A probabilistic approach is used to control the movement of the other bees, so the majority of bees will follow the elite bee's direction while a few bees may fly to other directions. This approach improves the capability of convergence to a global optimum.

2.2.5 Bee Swarm Optimization (Akbari, et al., 2010)

To improve the exploration and exploitation of foraging behavior of honey bees for numerical function optimization, an algorithm called Bee Swarm Optimization (BSO) was presented.

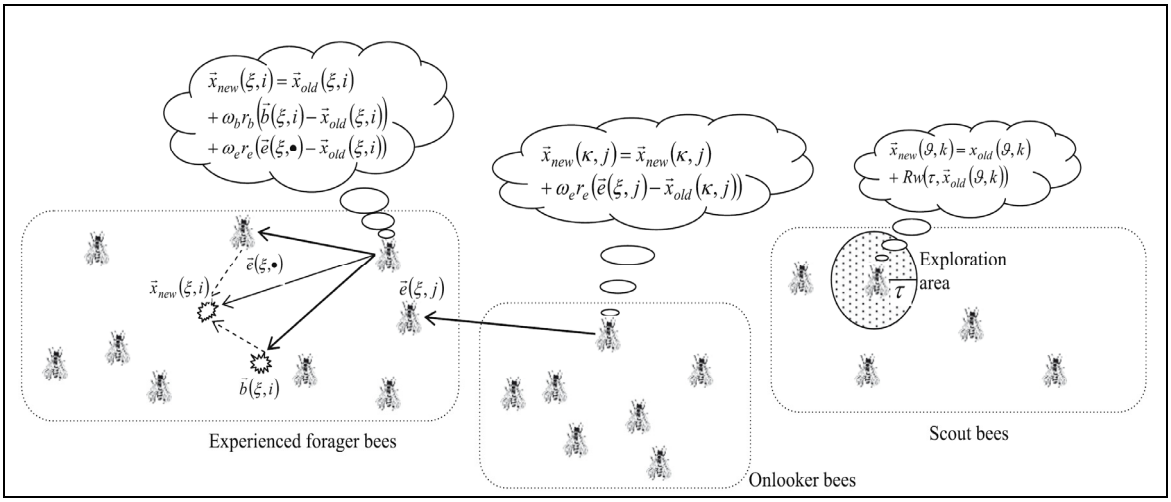


Figure 2.3 The structure of the bee swarm and the flying patterns of the bees (Akbari, et al., 2010)

In Figure 2.3, the bees of the swarm are sorted according to the fitness values of the most recently visited food source and these sorted bees are divided into three types. The bees with the worst fitness values are classified as scout bees, while the rest of bees are divided equally as experienced foragers and onlookers. Different flying patterns were introduced for each type of bee to balance the exploration and exploitation in this algorithm.

2.2.6 Summary of the Algorithms Based on Bee Foraging Behavior

In summary, the main objective of all algorithms mentioned in this section aim to find the optimal solution by using the method based on bee forging process concept. However, there are some differences in each algorithm. While VBA algorithm selects candidate solutions by randomly choosing from neighboring bees, other mentioned algorithms select candidate solutions based on fitness probability calculated from objective function. The best solution found by elite bee is also employed by BA, BSO, and Elite Bee algorithms to bias the solution's direction. Furthermore, all algorithms except VBA algorithm use a scout bee to improve the global search.

Although there are differences in the details of the algorithm variations, the same concept used on the most algorithms is that the bee agents in each algorithm tend to find the optimal solution based on the group decision making. This mechanism enables bee agents to find a solution by using the probability based on the fitness value on each candidate solution. The candidate solution with a higher fitness will have a larger chance to be selected by bee agents than ones with lower fitness values.

Accuracy of results and convergence speed are common criterions to indicate the algorithm performance. This performance depends on the balance between the exploration and exploitation process of bee agents in each algorithm. The difference in method to balance these processes makes one algorithm differ from another.

2.3 Current Applications of ABC Metaheuristic

Although the ABC algorithm is designed for solving the numerical optimization problems, it can be adapted and used to solve the combinatorial optimization problems. More clearly, in order to describe how ABC metaheuristic can be applied to optimization problems, this section provides a brief description of the main point to

consider when applying ABC metaheuristic to the practical optimization problems. Examples of these problems are listed in Table 2.1.

Table 2.1 Current application of ABC algorithm

Problem name	Problem type	Main references
Generalized assignment	Assignment	Baykasoğlu, et al., 2007
Flow shop scheduling	Scheduling	Pan, et al., 2011
Minimum spanning tree	Subset	Singh, 2009
Training neural networks	Machine learning	Karaboga, et al., 2007
Data clustering	Classification	Karaboga and Ozturk, 2011; Zhang, et al., 2010
Pattern recognition	Image processing	Xu and Duan, 2010; Banharnsakun, et al. 2011

2.3.1 Generalized Assignment Problem

The Generalized Assignment Problem (GAP) is the problem that intends to assign a set of tasks to a set of agents with minimum total cost. Each agent represents a single resource with limited capacity. Each task must be assigned to only one agent and it requires a certain amount of resources of the agent. This problem occurs in many application domains such as location problems, vehicle routing, group technology, scheduling etc. It can be formulated as an integer programming model as follows.

$$\min \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij}$$

subject to

$$\sum_{i=1}^{n} a_{ij} x_{ij} \leq b_j \quad \forall j, 1 \leq j \leq m$$

$$\sum_{j=1}^{m} x_{ij} \quad \forall i, 1 \leq i \leq n$$

$$x_{ij} \in \{0,1\} \qquad 1 \leq i \leq n \ \ \forall i, \qquad 1 \leq j \leq m \ \ \forall j$$

where I is set of tasks ($i = 1, \dots, n$); J is set of agents ($j = 1, \dots, m$); b_j is resource capacity of agent j ($b_j \geq 0$); a_{ij} is resource needed if task i is assigned to agent j ($a_{ij} \geq 0$); c_{ij} is cost of task i if assigned to agent j ($c_{ij} \geq 0$); x_{ij} is decision variable ($x_{ij} = 1$, if task i is assigned to agent j; 0, otherwise). The objective function represents the total assignment cost where the first constraint set is related to the resource capacity of agents and the second constraint set ensures that each task is assigned to only one agent.

In order to find the total minimum cost of GAP, a list of agents that contain random tasks represents as a food source in ABC algorithm. The objective function and both constraints are encoded as a fitness value. ABC process adjusts the tasks in each agent

based on Shift Neighborhood method. This method is related to the transferring of a task from one agent to another agent to improve a new solution.

2.3.2 Flow Shop Scheduling Problem

Flow shop scheduling problem is a problem found in the manufacturing systems. In this flow shop, a job is allowed to overlap its operations between successive machines by splitting it into a number of smaller subslots. Through the extensive use of just-in-time system in manufacturing, the performance measure related to both earliness and tardiness penalties has raised significant attention. The flow shop scheduling problem aims to solve an n-job and m-machine by finding a minimum of the total weighted earliness and tardiness penalties as follows.

$$\min f(\pi) = \sum_{j=1}^{n} (\alpha_j E_j + \beta_j T_j)$$

where π is a processing sequence of the jobs; α_j and β_j is the earliness and tardiness penalties of job j respectively; $E_j$is the earliness of job j and T_j is the earliness of job j.

The ABC algorithm can be used to solve this problem by mapping a food source as a discrete job permutation. The objective function $f(\pi)$ is encoded to represent as the fitness value for each food source. The neighboring food sources are generated based on insert and swap operators of a permutation π. The insert operator is defined by removing a job from π from its original position j and inserts it into another position k whereas the swap operator produces a neighbor of π by interchanging two jobs of π in the different positions. Employed bees, onlooker bees and scout bees then perform these operators based on the ABC-based searching process in order to find the optimal processing sequence of the jobs.

2.3.3 The Leaf-constrained Minimum Spanning Tree Problem

The leaf-constrained minimum spanning tree (LCMST) plays an important role in several practical applications such as facilities location, circuit and network design. Let $G = (V, E)$ be an undirected, connected, weighted graph where V denotes the set of n nodes and E denotes the set of edges. Given a non-negative weight function $w: E \rightarrow \mathbb{R}^+$ associated with its edges and positive integer $\ell\,(2 \leq \ell < n-1)$, LCMST problem

consists in seeking a spanning tree $T \subseteq E$ that constrained at least ℓ leaves and has minimum total weight among all such spanning trees of graph G as follows.

$$\min W(T) = \sum_{e \in T} w(e)$$

The subset encoding method is used to construct each leaf-constrained spanning tree (LCST) as a food source in ABC algorithm. This method represents a food source in a bit-string form. The weight cost of each LCST is considered a fitness value. The neighboring food sources of employed bees and onlooker bees are generated based on exchanging interior node between the LCST. After the exchange is finished, if the solutions are identical, it is called a collision. If a collision occurs while generating a neighboring food source for an employed bee then the original food source is abandoned and this employed bee will change its status to a scout bee and will perform to randomly generate a new food source. If a collision occurs while generating a neighboring food source for an onlooker bee then another food source is chosen randomly for another exchanging. This process is repeated until it has found a new food source that is different from the original food source.

2.3.4 Training Neural Networks Problem

This problem aims to find optimal weight set of Artificial Neural Networks (ANNs) in training process. ANNs are quite successful in modeling non-linearity and involved in so many applications in research fields. The finding of optimal weight values is an important key to make the successful design of ANNs.

Traditionally, ANN training has been carried out by using back-propagation gradient descent method. However, this technique has some drawbacks such as getting stuck in local minima and computational complexity. Global optimization methods have been proposed to overcome these disadvantages of gradient based algorithms.

To find the optimal weight set of ANN, it can be carried out by minimizing the network error function E as follows.

$$\min E\big(w(t)\big) = \frac{1}{n}\sum_{j=1}^{n}\sum_{k=1}^{K}(d_k - o_k)^2$$

where $E(w(t))$ is the error at the tth iteration, $w(t)$ is the weights in the connections at the tth iteration, d_k and o_k is the desired and actual value of the kth output node respectively. K is the number of output nodes and n is the number of patterns.

To map this problem with ABC algorithm, the set of network weights is presented as a food source. The error E produced by the ANNs is used to calculate the fitness value. The minimization process of the objective function E is performed based on ABC process in order to find the optimal set of network weights.

2.3.5 Data Clustering Problem

The data clustering problem occurs during gathering data into clusters so that the data in each cluster shares a high degree of similarity while being very dissimilar to data from other clusters. The problem happens in several applications such as data mining, statistical data analysis, data compression, and vector quantization.

Distance measurement is generally used for evaluating similarities between patterns. The objective of clustering is to minimize the sum squared Euclidean distances between each object and the center of the cluster belonging to every such allocated object as follows.

$$\min\ J(w,z) = \sum_{i=1}^{N}\sum_{j=1}^{K} w_{ij}\left\|x_i - z_j\right\|^2$$

where N is the number of patterns, K is the number of clusters, x_i is the location of ith pattern and z_j is the center of the jth cluster that is calculated as follows.

$$z_j = \frac{1}{N_j}\sum_{i=1}^{N} w_{ij}x_i$$

Where N_j is the number of patterns in the jth cluster, w_{ij} is the association weight of pattern x_i with cluster j which will be either 1 or 0 (if pattern i is allocated to cluster j , w_{ij} is 1, otherwise 0).

Given a database with C classes and N parameters, the clustering problem can be seen as that of finding the optimal positions of C centroids in an N-dimensional space. Thus, to represent this problem in term of ABC parameters, positions of centroids are mapped to the coordinates of food sources (x_i). The cost function calculated from the Euclidean distances between food sources (x_i) and the coordinates of objects is represented as

fitness value. The ABC process attempts to find the optimal positions of C centroids by minimizing the Euclidean distance value.

2.3.6 Pattern Recognition Problem

Pattern recognition problem is the task of finding a given pattern in an image. Human beings can recognize a multitude of patterns in images with little effort, but it is a complicated task and is still a challenge for computer vision systems. Similarity measuring is a very important aspect that has been used for solving this problem.

Several methods have been used to determine the similarity such as Hausdorff distance based matching (HD), Mutual Information (MI) and Edge Potential Function (EPF). However, the functions of these similarity measures with respect to transformation parameters are generally non-convex and irregular. A global optimization method is thus required to optimize these functions. They can be represented as follows.

For Hausdorff Distance Based Matching Method:

$$\min HD(A,B) = \max\left(h(A,B), h(B,A)\right)$$

where

$h(A,B) = \max_{a\in A} \min_{b\in B} \|a-b\|$

$h(B,A) = \max_{b\in B} \min_{a\in A} \|b-a\|$

Given two finite point set A and B, the function $h(A,B)$ is called the directed Hausdorff distance from A to B. It identifies the point $a \in A$ that is farthest from any point of B and measures the distance from a to its nearest neighbor in B.

For Mutual Information Method:

$$\max MI(I_R, I_T) = H(I_R) + H(I_T) - H(I_R, I_T)$$

where I_R is a reference image and I_T is a target image. $H(I_R)$ and $H(I_T)$ are the Shannon's entropies of a reference image and target image respectively, $H(I_R, I_T)$ is the joint entropy of the two images.

For Edge Potential Function Method:

$$\max EPF(x,y) = \frac{1}{4\pi\varepsilon_{eq}} \sum_i \frac{Q_{eq}(x_i,y_i)}{\sqrt{(x-x_i)^2+(y-y_i)^2}}$$

where ε_{eq} is a constant related with the image background situation. (x, y) represents the coordinates of any point of an image, $Q_{eq}(x_i,y_i)$ is the ith edge point in the image at coordinates (x_i,y_i).

When the pattern recognition problem is mapped to the ABC algorithm, transformation parameters including translating, rotating, and scaling are considered a food source. The fitness value will be related on the functions described above. If the Hausdorff distance based matching method is selected as a measure, the ABC algorithm will attempt to find the transformation parameters that can minimize HD value. On contrary, if Mutual Information or Edge Potential Function Method is selected as a measure, it will attempt to find the transformation parameters that can maximize MI or EPF value respectively.

In summary, based on the several application problem domains mentioned in this section, these problems are complex. Therefore, their solutions can be considered a very good indicator to show that the ABC is one of effective metaheuristic approaches which can be applied to solve the optimization problems consisting of both combinatorial and numerical function problems.

CHAPTER 3 ARTIFICIAL BEE COLONY OPTIMIZATION FRAMEWORK

In the previous studies, the mapping between the problem space and the ABC algorithm has not been clearly illustrated, which made it difficult for the researchers to adopt the algorithm in their problem domains. To improve the mapping description, we introduce the use of the set theory in this chapter.

Firstly, the basic optimization model is discussed. From this model, the optimization framework based on the Artificial Bee Colony including the objective determination, parameters mapping and balancing of exploitation and exploration are then addressed. Lastly, the examples of problem mapping are given.

3.1 Optimization Model

To design an effective optimization model, three key aspects must be considered. In the first aspect, the objective of the optimization problem that requires us to solve is pointed out. The next aspect is how to map the parameters between the optimization algorithm and the problem. The last aspect is how to balance the exploration and exploitation in the algorithm's search process. This optimization model is shown as Figure 3.1 below.

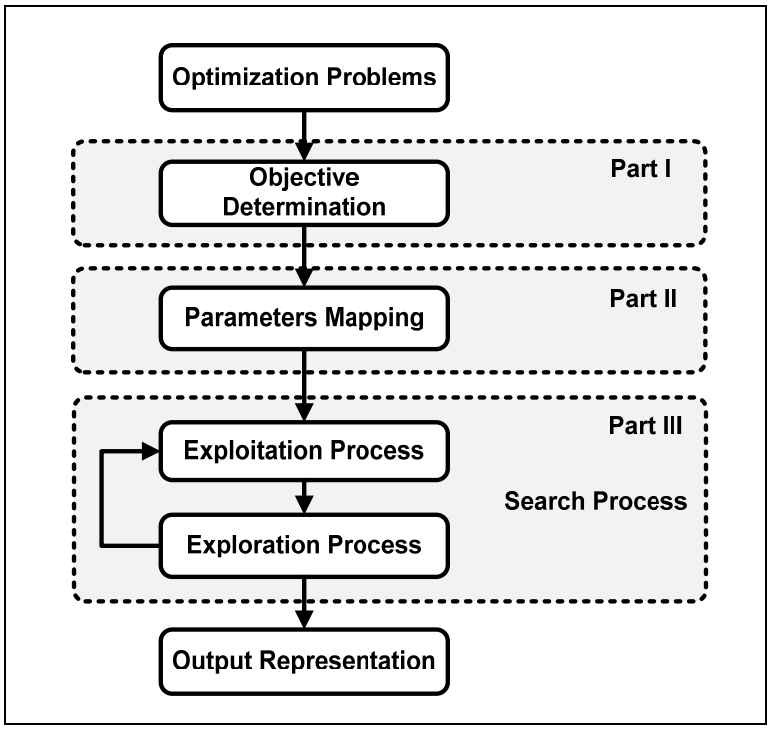


Figure 3.1 Optimization Model

3.2 ABC Based Optimization Framework

3.2.1 Objective Determination

Generally, the objective determination depends on each individual optimization problem. However, the objective of the optimization problems is to find either the maximum or the minimum solution from all feasible solutions. To show as a guideline, the objective determination on example optimization problems is illustrated in Table 3.1.

Table 3.1 Example of Objective Determination

Problem name	Objective Determination
Generalized assignment	To assign a set of tasks to a set of agents with minimum total cost.
Flow shop scheduling	To solve an *n*-job and *m*-machine by finding a minimum of the total weighted earliness and tardiness penalties.
Minimum spanning tree	To seek a spanning tree with constrained leaves with minimum total weight
Training neural networks	To find the optimal weight set of Artificial Neural Networks with minimum neural network error function.
Data clustering	To minimize the sum squared Euclidean distances between each object and the center of the cluster belonging to every such allocated object.
Pattern recognition	To find the transformation parameters with maximize the similarity function value.

3.2.2 Parameters Mapping

The ABC algorithm is designed for solving numerical optimization problems and the algorithm utilizes equations 2.2 and 2.4 for finding a solution in a continuous function domain. These equations cannot be used directly to solve problems in combinatorial optimization, which its solution is in a discrete domain. However, the algorithm can be expanded for this problem type with suitable modifications (Karaboga and Akay, 2009b).

There are several methods proposed to modify the ABC algorithm for solving discrete optimization problems. Singh (2009) employed a subset encoding to represent a solution in an artificial bee colony algorithm for solving the leaf-constrained minimum spanning tree problem. Pan et al. (2011) also proposed a discrete artificial bee colony (DABC) algorithm to minimize total weighted earliness and tardiness penalties for the lot-streaming flow shop scheduling problems.

In this section, we introduce a set theory to illustrate how to map a problem in consideration to a representation that ABC metaheuristic can be used to find the solutions in both numerical and combinatorial optimization problems.

We consider the search process in the ABC metaheuristic in terms of (S, f, Ω), where S is the set of candidate solutions, f is the fitness function which assigns a fitness value $f(s)$ to each candidate solution $s \in S$ and Ω is a set of constraints. Solutions $\tilde{s}$ belonging to the set $\tilde{S} \subseteq S$ that satisfy Ω are called feasible solutions.

In ABC, the initial solutions, which are the feasible solutions $(\tilde{s})$ in the feasible search space $\tilde{S}$, are randomly constructed and assigned to employed bees. After initialization, the artificial bees are subjected to repeat cycles of three major processes: updating feasible solutions, selecting feasible solutions, and avoiding suboptimal solutions.

In the process of updating feasible solutions, employed bees update their solutions with their neighbors by using the concept of equation (2.2). The old solutions in an employed bee's memory are replaced with new solutions of higher fitness.

During the selection of feasible solutions $(\tilde{s})$, each onlooker bee selects one of the proposed solutions depending on the information obtained from the employed bees. The probability of a solution being selected by an onlooker bee is proportional to its fitness as calculated by equation (2.3). After solutions are selected, the onlooker bees also update their selected solutions by using the concept of equation (2.2) based on the neighboring solution $s_n \in \widetilde{S}$ where $s_n \neq \tilde{s}$ for all $\tilde{s}$ obtained from employed bees in each iteration. Next, the old solutions in an onlooker bee's memory are replaced with new solutions of higher fitness from the same step of employed bee.

In the process of avoiding suboptimal solutions, solutions that do not improve the fitness are replaced with new solutions randomly constructed by the scout bees. The concept to generate the new solution is based on the equation (2.4).

These three major processes are repeated until a globally optimal feasible food source $s^* \in \tilde{S}$ where $f(s^*) \geq f(\tilde{s})$ for all $\tilde{s} \in \tilde{S}$ is found or the number of iteration reaches the Maximum Cycle Number (MCN).

Given this formal characterization, ABC metaheuristic can be applied to a wide variety of interesting optimization problem. It allows the artificial bees to find the solutions by

updating their solutions to optimal solutions on feasible search spaces. The examples of parameter mapping between the ABC algorithm and the problems can be shown in Table 3.2.

Table 3.2 Examples of parameter mapping

Problem name	**Fitness Value Representation** $(f(\tilde{s}))$	**Food Source Representation** $(\tilde{s})$
Generalized assignment	Total assignment cost	A list of agents containing assigned tasks
Flow shop scheduling	Total weighted earliness and tardiness penalties	A discrete job permutation
Minimum spanning tree	Weight cost of leaf-constrained spanning tree	A leaf-constrained spanning tree
Training neural networks	Neural network error	A set of neural network weights
Data clustering	Total weight cost of Euclidean distances	A position of centroids of clusters
Pattern recognition	Similarity value	A set of transformation parameters

3.2.3 Balancing of Exploitation and Exploration

In a robust search process, exploration and exploitation must be carried out together. While the exploration process is related to the independent search for an optimal solution, exploitation uses existing knowledge to bias the search space.

Candidate solutions will be converged more slowly to the optimal solution if the optimization model employs only the exploration search. Only using the exploitation search, optimization model will lead the candidate solutions to entrap easily in a local optimum. The success of an optimization algorithm then depends highly on the balancing mechanism between exploration and exploitation.

In the mentioned ABC algorithm, the exploitation will be handled by employed and onlooker bees while the exploration will be maintained by scout bees. The exploitation process depends on the group's decision making among bee agents. Onlooker bee agents use the knowledge and information from employed bee agents to bias the direction of their solutions. The exploration process, which is a random search process, occurs when the bee agents cannot find a better solution in a threshold period.

This mechanism enables ABC to have both the local and the global search ability. Based on this advantage, the ABC algorithm will be able to get out of a local optimum point in the search space and find the global optima better than other heuristic approaches such as Simulated Annealing and Tabu Search that only have a local search

ability. As a result, many researchers have applied the ABC algorithm to solve several science and mathematical applications (Rao et al., 2008; Singh, 2009; Sabat et al., 2010). The comparison results showed that the ABC algorithm performs better than other heuristic algorithms in terms of solution quality and computation efficiency.

The ABC based optimization framework can be illustrated in Figure 3.2.

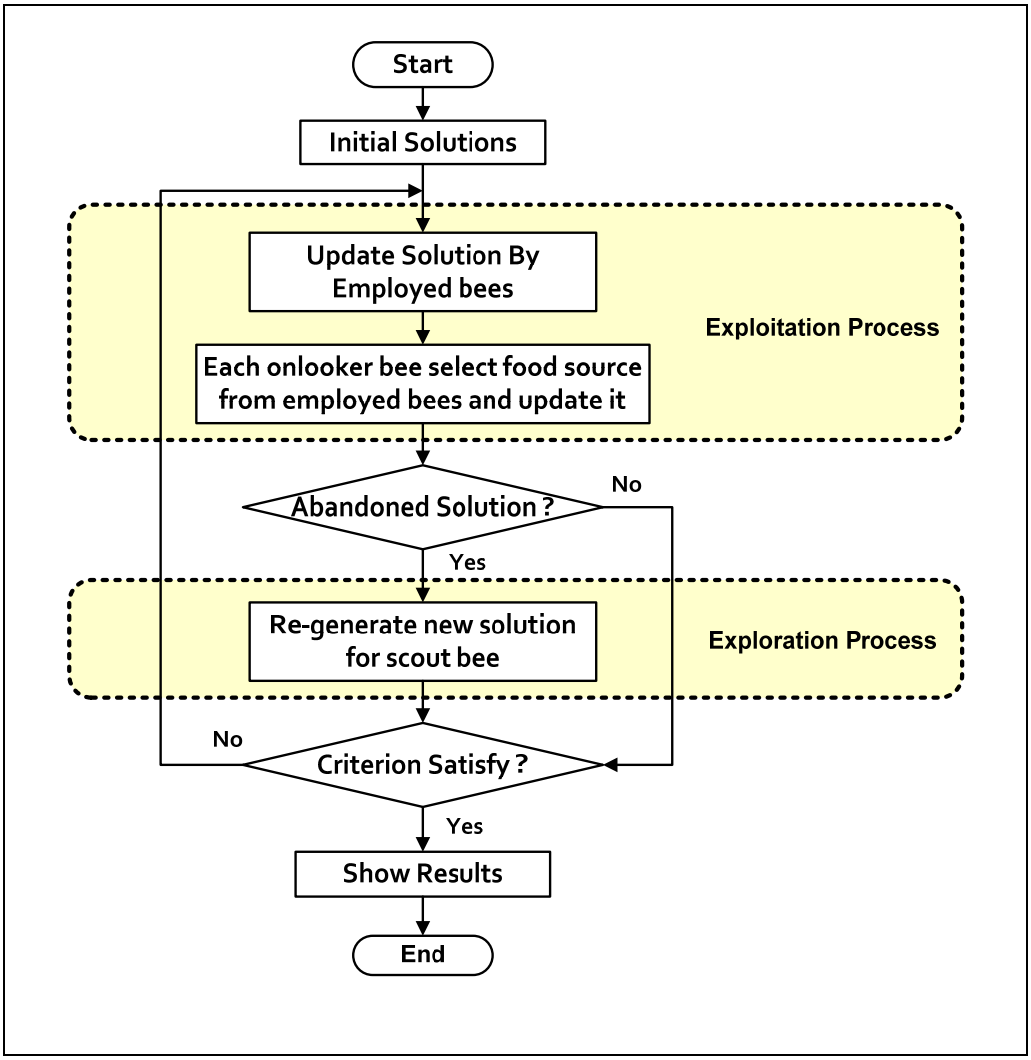


Figure 3.2 Exploitation and Exploration in ABC

3.3 The Example of the ABC Metaheuristic Representation

To illustrate how our mapping framework works, two problems are used as our examples. For the first, we will solve the Traveling Salesman Problem (TSP), a classical optimization problem. Next, the dimension reduction problem in bioinformatics data is presented.

3.3.1 The Traveling Salesman Problem (TSP)

The Traveling Salesman Problem (TSP) is an example of combinatorial optimization problems known to be NP-complete. It naturally arises as a sub-problem in various application domains such as network communication, transportation, manufacturing and logistics (Bella and McMullen, 2004; Fournier and Pierre, 2005; Bagchi, et al., 2006; Silvaa, et al., 2008).

Generally, the TSP states that in order for one salesman who wants to visit *n* different cities, his objective would be to find the sequence of tour that minimizes the cost of travelling to visit each city exactly once and finally returns to the starting point.

Given a complete undirected graph $G = (V, E)$ with V being the set of cities, E being the set of edges fully connecting the cities V, and each edge $\in E$ being assigned a value d_{ij} that is the cost of the traveling between cities i and j, with $i, j \in V$, the Traveling Salesman Problem is concerned with finding the closed path which visits each of the $n = |V|$ cities of G exactly once with minimal cost. The function of traveling cost $f(v)$ is represented by the equation below:

$$f(v) = \sum_{i=1}^{n-1} d_{v(i)v(i+1)} + d_{v(n)v(1)}$$

where v is a sequence of the tour in a possible closed path.

Figure 3.3 describes the steps that need to be taken to solve TSP using ABC.

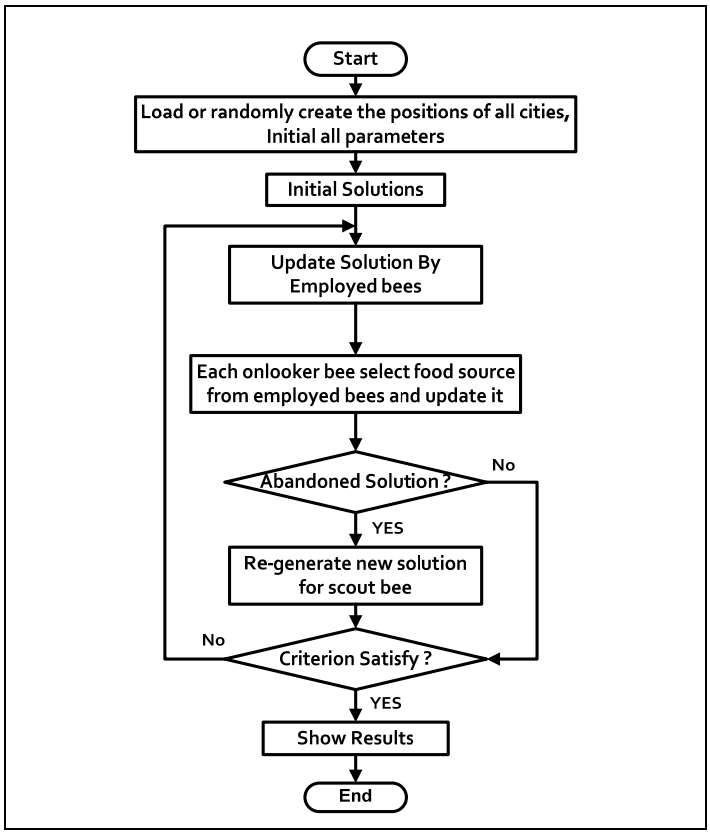


Figure 3.3 The ABC algorithm flowchart for TSP

First, the coordinates of all cities are either randomly generated or loaded from an input file. The cost of traveling between any two cities is determined by their Euclidean distance. A symmetric TSP instance is used in this representation. In symmetric TSP, the traveling cost between two cities is the same in either direction. Next, the initial parameters such as the number of employed bees, onlooker bees and maximum number of iteration (MCN) are set. New candidate solutions, which are sequences of the tour, are constructed and represented as the food sources. The fitness of each food source is determined by the inverse of its traveling cost. This mapping is shown in Figure 3.4 below.

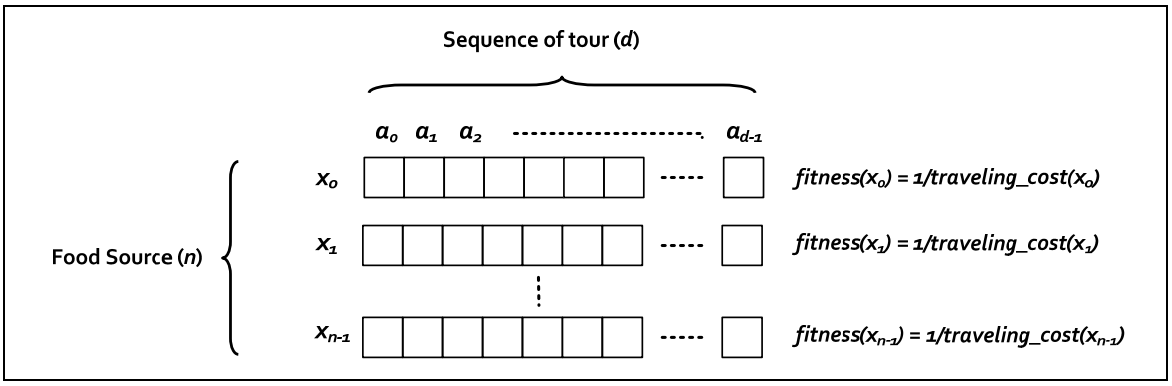


Figure 3.4 The mapping of the food sources and the sequence of the tour

In the second step, the food sources are updated by employed bees. Greedy Subtour Crossover (GSX) (Sengoku and Yoshihara, 1998) is the mechanism used to update the employed bees' old food sources based on their neighboring food sources. The algorithm and an example of GSX are shown in Figure 3.5 and 3.6 respectively.

```
Inputs:  the old food source                 x_old = {a_0, a_1, ..., a_{d-1}}
         the neighboring food source         x_neighboring = {b_0, b_1, ..., b_{d-1}}
Output:  The new food source                 x_new = {}

GSX_Algorithm(x_old, x_neighboring){
        s_a = true
        s_b = true
        Choose city c randomly
        Choose p, where a_p = c
        Choose q, where b_q = c
        x_new = x_new⊕c
        do{
                p = p − 1 (mod d)
                q = q + 1 (mod d)
                if (s_a = true){
                        if (a_p ∉ x_new){
                                x_new = a_p⊕x_new
                        }else{
                                s_a = false
                        }
                }
                if (s_b = true){
                        if (b_q ∉ x_new){
                                x_new = x_new⊕b_q
                        }else{
                                s_b = false
                        }
                }
        }while (s_a = true or s_b = true)
        if (|x_new| < d){
                add the rest of cities to x_new in the random order
        }
        return x_new
}
```

Figure 3.5 Greedy Subtour Crossover method

Note that "⊕" in $x_{new} = a_p \oplus x_{new}$ and $x_{new} = x_{new} \oplus b_q$ is the concatenation operator, and that sentence means to add a_p before x_{new} and to add b_q after x_{new} respectively.

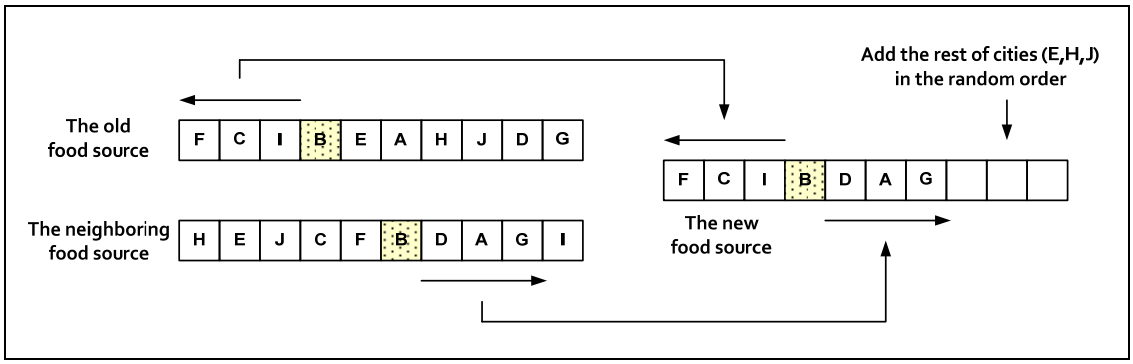


Figure 3.6 Example of Greedy Subtour Crossover method

Suppose that we have $x_{old} = \{F, C, I, B, E, A, H, J, D, G\}$ and $x_{neighboring} = \{H, E, J, C, F, B, D, A, G, I\}$. Start by choosing one city at random. In this example we will pick B. Then we derive $p = 3$ and $q = 5$ from the facts $a_3 = B$ and $b_5 = B$ respectively. Now $x_{new} = \{B\}$.

Next, alternate picking cities from x_{old} and $x_{neighboring}$. Start with a_2 (city I) because $p = 3 - 1 = 2$ and continue with $b_6 = D$ because $q = 5 + 1 = 6$. x_{new} becomes $\{I, B, D\}$.

Similarly, add a_1 (city C) and b_7 (city A) and add a_0 (city F) and b_8 (city G). x_{new} becomes $\{F, C, I, B, D, A, G\}$. Now the next city becomes $a_9 = G$ but since G has already appeared in x_{new}, we cannot add any more cities from x_{old}.

Instead we try to add cities from $x_{neighboring}$. The next city would be $b_9 = I$, but I is already visited in x_{new}. Thus we cannot add cities from $x_{neighboring}$ either.

Then, we add the rest of the cities, i.e. E, H and J , to x_{new} in random order. Finally, x_{new} becomes $\{F, C, I, B, D, A, G, H, E, J\}$.

After this updating process is completed, local search based on the 2-opt method (Croes, 1958) is also used to improve x_{new}. The 2-opt method is shown in Figure 3.7 below.

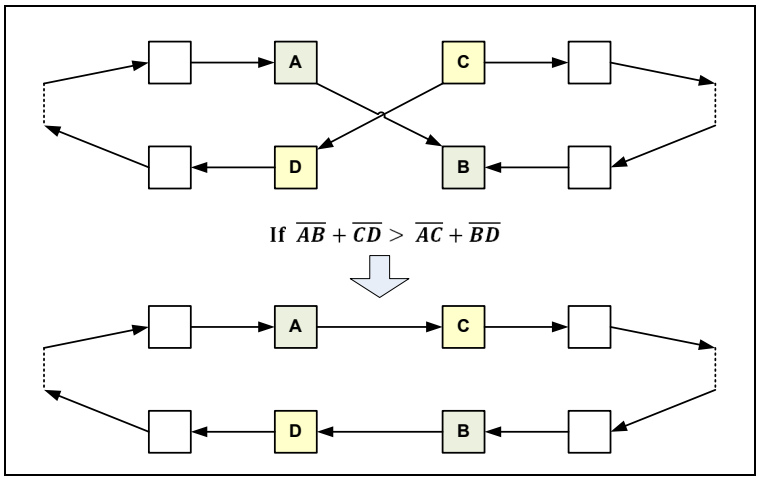


Figure 3.7 The 2-opt method

In the 2-opt method, every pair of edges in the tour sequence is compared to one another. From the example in Figure 3.7, if the sum of the traveling cost from A to B and from C to D exceeds the sum of the traveling cost from A to C and from B to D, we remove the two former edges (A to B and C to D) and add the two latter edges (A to C and B to D). This gives us the new tour sequence shown in the lower half of Figure 3.7.

An old food source (x_{old}) in an employed bee's memory will be replaced by a new food source (x_{new}) if the new solution has better fitness $(f(x))$.

In the third step, the employed bees share new tour sequences that they have found with the onlooker bees, who then select the tour sequences of higher fitness and update those sequences based on the same methods used by the employed bees including GSX and 2opt.

In the last step, tour sequences whose fitness has not been improved after a certain period are abandoned and replaced by new tour sequences constructed by the scout bee.

Steps 2 to 4 are repeated until the number of iteration reaches the MCN.

In this work, a hybrid method combining Artificial Bee Colony and Greedy Subtour Crossover (ABC-GSX) was proposed (Banharnsakun, et al. 2010a). The mapping of parameters of this algorithm for the combinatorial optimization problem domain was addressed. Traveling Salesman Problem, a classical optimization problem, was chosen to evaluate the effectiveness of this mapping and our proposed method. In this method, the exploitation process in the ABC algorithm is improved by combining GSX. The

results indicated that our hybrid method yielded more effective results for TSP, with an average relative error below 1% (See appendix A for more detail).

3.3.2 Dimension Reduction in Bioinformatics Data

In bioinformatics, data overload is a serious problem because the data analysis process requires extensive computational resources and time. Furthermore, the massive amount of data can make it difficult for the scientists to identify the interesting features. Hence, dimension reduction process is required to eliminate the irrelevant or noise features from all data in order to reduce computation time, resource usage and the complexity of identifying the significant features.

For a dimension reduction problem, the feature set of data is represented as a food source or solution in the ABC algorithm; each food source is encoded into a binary string which is similar to the characteristics of chromosome strings in genetic algorithm. For example, the food source which represents a feasible set of features can be expressed as $S = F_1F_2F_3F_4...F_d$; F is the feature or dimension of data and d is the total number of dimensions. Hence, F_1 represents the first dimension; F_2 represents the second dimension, and so on. The value of each feature F is binary[0, 1].

The feature value {1} indicates a selected feature, and the feature value {0} indicates a non-selected feature. In this case, the selected feature is determined as a required feature that cannot be eliminated from the original data whereas a non-selected feature is a feature that can be removed from the original data. The mapping of the food sources in ABC and a feasible set of features is shown in Figure 3.8 where n is the number of food sources (solutions).

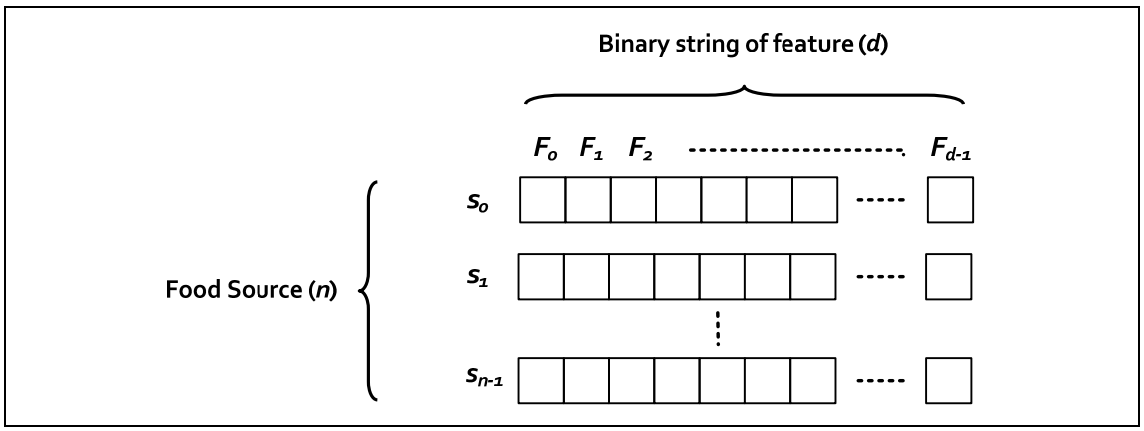


Figure 3.8 The mapping of the food sources and a feasible set of features

Our proposed method applies ABC wrapping with a k-nearest neighbors (kNN) classifier. kNN is used for fitness evaluation to estimate the fitness value of the ABC' food sources. After the employed bees and the onlooker bees generate new candidate food sources, which are a subset of selected features, kNN is performed to evaluate the classification accuracy of the new candidate food sources which is used as a criterion for selecting the optimal subset of features.

In this study, we set the number of nearest neighbors, *k*, to one so that each input data will simply be classified to the nearest-neighboring data class. The leave-one-out cross-validation (LOOCV) method is used to evaluate the accuracy of ABC-based feature selection. LOOCV will use one sample data item from the original dataset as the validation data; the remaining items will be used as the training data. This training process is repeated until all data items in the dataset have been used as the validation data item.

We applied the ABC-kNN method to feature selection and classification problem described above. The goal is to find the optimal subset of features from the original feature set while the resulting subset satisfies the highest classification accuracy. The feature selection and the classification process of the ABC-kNN are illustrated in Figure 3.9.

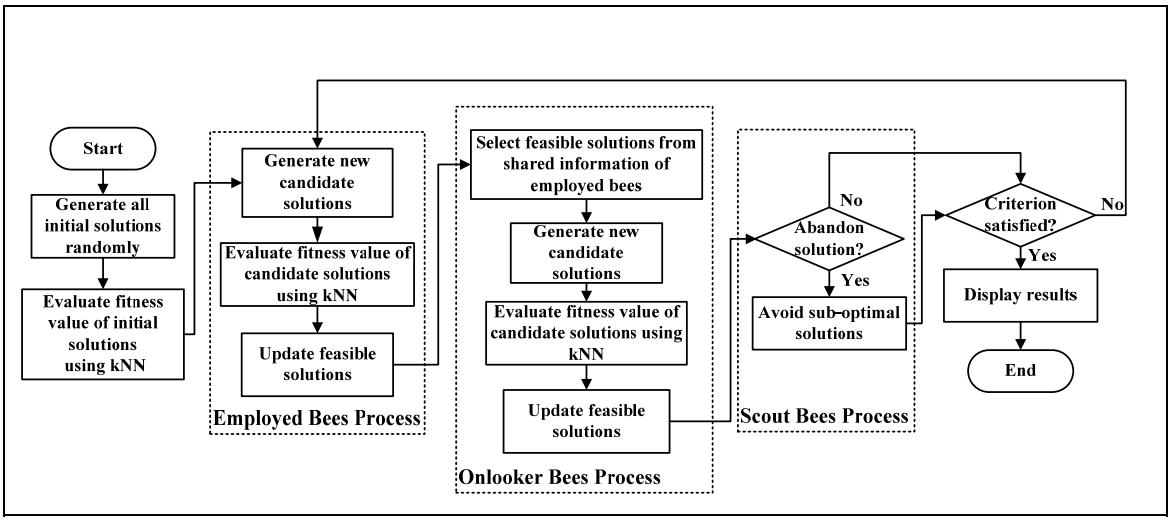


Figure 3.9 The flowchart of ABC-kNN method

Initial Process

At initialization, the initial parameters including the number of employed bees, the number of onlooker bees, the maximum number of iterations, and a predetermined

number of iterations called "*limit*" are set. The food sources (each one a feasible set of features) is randomly generated. Then, each solution is evaluated by using a kNN classifier. The classification accuracy value obtained from kNN classifier will be used to represent as a fitness value. Thus, the more accurate the classification, the higher the fitness value.

Employee Bees Process

In this process, each food source is modified based on the process of updating feasible solution by employed bees as expressed in equation (2.2). However, in our algorithm framework, the solutions are represented with binary numbers. We adapt the concept of the candidate solution update based on a neighboring solution using the bitwise AND operation (∧) as follows:

$$v_{ij} = \begin{cases} x_{ij} \wedge x_{kj} & if\ (x_{ij} = x_{kj}) \\ 0 & if\ (x_{ij} \neq x_{kj}\ and\ \phi \leq 0.5) \\ 1 & if\ (x_{ij} \neq x_{kj}\ and\ \phi > 0.5) \end{cases} \quad (3.1)$$

where

v_{ij} = The new candidate solution *i* feature *j*
x_{ij} = The current solution *i* feature *j*
x_{kj} = The neighboring solution *k* feature *j*
ϕ = Random real number between 0 to 1

In this method, if the value for the feature *j* obtained from the current solution and its neighboring solution are the same, this feature value will be kept in the new candidate solution (i.e. either 0 for not selected and 1 for selected). Otherwise, the feature will be randomly assigned with a new value (either 1 or 0).

The new candidate solution is then evaluated with a kNN classifier. If the new fitness value is better than the current one, the employed bees will replace their solution with this new candidate solution; otherwise, the new candidate solution will be ignored.

Onlooker Bees Process

After employed bees share information of their solutions, onlooker bees will select a food source to visit according to the probability of each food sources based on equation (2.3) and perform the process of updating feasible solution similar to employed bees by

using equation (3.1). The current solution in the onlooker bee's memory will be replaced by the new candidate solution if the new solution has a better fitness value.

Scout Bees Process

In this process, if the fitness value of the current food source has not been improved by a predetermined number of iterations, called the "*limit*", the food source will be abandoned. Then, the scout bees will randomly generate a new food source location in all dimensions. This process is performed in order to avoid selecting the sub-optimal solution.

Termination Process

Processes of employed bees, onlooker bees, and scout bees will be repeated until the number of iterations reaches the predefined maximum number of iterations.

In summary, the objective of the study is to provide a mapping framework of ABC method for data dimensionality reduction problem. Our proposed method called ABC-kNN (Prasartvit et al., 2011) embraces the strength of the heuristic search of ABC algorithm for classification problems.

The proposed algorithm is validated in gene expression analysis. The experimental results show that the proposed method can effectively reduce the data dimension while maintaining high classification accuracy. The remaining data features accurately indicate the underlying features that have the most effects on the system (See appendix B for more detail).

CHAPTER 4 THE BEST-SO-FAR ABC ALGORITHM

This Chapter outlines our version of the Artificial Bee Colony (ABC) model. This modified version is called *Best-so-far ABC*. The Best-so-far ABC algorithm changes the mechanism used to calculate new candidate food sources, and adopts an adjustable radius, as well as a new way to compare candidate food sources. The computational complexity is also addressed in order to compare best-so-far ABC with original ABC and BSO.

4.1 The Ideas of the Best-so-far ABC Algorithm

Although the activities of exploitation and exploration are well balanced and help to mitigate both stagnation and premature convergence in the ABC algorithm, the convergence speed is still an issue in some situations.

To enhance the exploitation and exploration processes, we are proposing to make three major changes by introducing the best-so-far method, an adjustable search radius, and an objective-value-based comparison method.

In our best-so-far ABC, the exploitation process is focused on the best-so-far food source. This food source is used in the comparison process for updating the new candidate food source to accelerate the convergence speed. The searching ability of the exploration process is also enhanced by the scout bee. It will randomly generate a new food source to avoid local optima.

The pseudo-code of the best-so-far ABC algorithm is illustrated in Figure 4.1. In this pseudo-code, the best-so-far method is shown in line number 37 where the onlooker bees will register the best selected food source position so far within the new candidate generation function. The adjustable search radius is performed by a scout bee in line number 50 and the objective-value-based comparison method is used in line number 14, 27, 39 and 52. The full details of each change are described in section 4.1.1-4.1.3.

Line	Initialization:
1	**For** $i = 1$ *to* $n(FS)$ //do for all food sources
2	**For** $d = 1$ *to* D
3	$x(FS, id) = x(min, d) + rand[0,1] * (x(max, d) - x(min, d))$ //initialize the feasible solutions in the search space
4	**Next** d
5	**Next** i
6	**While** $(iteration \leq MaxIteration)$
7	**For** $i = 1$ *to* $n(EB)$ //do for each employed bees
8	**For** $d = 1$ *to* D
9	$x(EB, id) = x(FS, id)$ //assign the food source position to the employed bee
10	**Next** d
11	**Select** d_s //Randomly select the dimension of the solution
12	**Select** $x_n(FS, d_s)$ //Randomly select the neighboring solution
	//update the position of the employed bee
13	$x(EB, id_s) = x(FS, id_s) + rand[-1,1] * (x(FS, id_s) - x_n(FS, d_s))$
14	**If** $f(x(EB, i)) < f(x(FS, i))$ //compare and select the better solution between the old and the new solution
15	$x(FS, id_s) = x(EB, id_s)$ //Replace the old solution with the new solution
16	*limit_count*$(x(FS, i)) = 0$
17	**Else**
18	*limit_count*$(x(FS, i)) + +$
19	**Next** i
20	**For** $i = 1$ *to* $n(EB)$ //do for all employed bees for selecting the best-so-far solution
21	**If** $i = 1$
22	**For** $d = 1$ *to* D
23	$x_b(FS, d) = x(FS, id)$
24	**Next** d
25	$f(x_b(FS)) = f(x(FS, i))$
26	**Else**
27	**If** $f(x(FS, i)) < f(x_b(FS))$
28	**For** $d = 1$ *to* D
29	$x_b(FS, d) = x(FS, id)$
30	**Next** d
31	$f(x_b(FS)) = f(x(FS, i))$
32	**Next** i
33	**For** $i = 1$ *to* $n(OB)$ //do for each onlooker bee
34	**Select** d_s //Randomly select the dimension of the solution
35	**Select** $x_s(FS, d_s)$ //Select the food source based on equation 2.3
36	**For** $d = 1$ *to* D
	//update the position of the onlooker bee
37	$x(OB, id) = x_s(FS, d_s) + rand[-1,1] * fitness(x_b(FS)) * (x_s(FS, d_s) - x_b(FS, d_s))$
38	**Next** d
39	**If** $f(x(OB, i)) < f(x_s(FS))$ //compare and select the better solution between the old and the new solution
40	**For** $d = 1$ *to* D
41	$x_s(FS, d) = x(OB, id)$ //Replace the old solution with the new solution
42	**Next** d
43	*limit_count*$(x_s(FS)) = 0$
44	**Else**
45	*limit_count*$(x_s(FS)) + +$
46	**Next** i
47	**For** $i = 1$ *to* $n(FB)$ //do for all food sources for abandoning the food source that cannot improve the further result
48	**If** *limit_count*$(x(FS, i)) >$ *limit*
49	**For** $d = 1$ *to* D
	//update the position of the scout bee
50	$x(SB, d) = x(FS, i) + rand[-1,1] * \left(\omega_{max} - \frac{iteration}{MaxIteration}(\omega_{max} - \omega_{min})\right) * x(FS, i)$
51	**Next** d
52	**If** $f(x(SB)) < f(x(FS, i))$ //compare and select the better solution between the old and the new solution
53	**For** $d = 1$ *to* D
54	$x(FS, id) = x(SB, d)$ //Replace the old solution with the new solution
55	**Next** d
56	*limit_count*$(x(FS)) = 0$
57	**Else**
58	*limit_count*$(x(FS)) + +$
59	**Next** i

Figure 4.1 The pseudo-code of the Best-so-far ABC algorithm

Note that the pseudo-code in Figure 4.1 is used to find the minimum objective value. To find the maximum objective value, the condition which is used to compare the old solution and the new solution, in line number 14, 27, 39 and 52, must be changed from

$$f(x_{new}(*)) < f(x_{old}(*))$$

to

$$f(x_{new}(*)) > f(x_{old}(*))$$

where $f(x(*))$ is the objective function value.

In order to investigate the effect of each proposed modification to the overall performance of the proposed algorithm, a set of experiments were conducted. The effect of each proposed modification to solution quality and convergence speed was evaluated using numerical benchmark functions shown in Table 4.1. The results are presented at the end of each section. Note that we only show the result on the Griewank function in this chapter. The complete experimental results are listed in Appendix C.

Table 4.1 Numerical benchmark functions

Function Name	Function	Characteristic	Ranges
Sphere	$f_1(\vec{x}) = \sum_{i=1}^{D} x_i^2$	Uni-modal, Separable	$-100 \leq x_i \leq 100$
Griewank	$f_2(\vec{x}) = \frac{1}{4000}\left(\sum_{i=1}^{D}(x_i^2)\right) - \left(\prod_{i=1}^{D}\cos\left(\frac{x_i}{\sqrt{i}}\right)\right) + 1$	Multi-modal, Non-Separable	$-600 \leq x_i \leq 600$
Rastrigin	$f_3(\vec{x}) = \sum_{i=1}^{D}(x_i^2 - 10\cos(2\pi x_i) + 10)$	Multi-modal, Separable	$-5.12 \leq x_i \leq 5.12$
Rosenbrock	$f_4(\vec{x}) = \sum_{i=1}^{D} 100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2$	Uni-modal, Non-Separable	$-30 \leq x_i \leq 30$
Ackley	$f_5(\vec{x}) = 20 + e - 20e^{(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2})} - e^{\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)}$	Multi-modal, Non-Separable	$-30 \leq x_i \leq 30$
Schaffer	$f_6(\vec{x}) = 0.5 + \frac{(\sin\sqrt{x_1^2 + x_2^2})^2 - 0.5}{(1 + 0.001(x_1^2 + x_2^2))^2}$	Multi-modal, Non-Separable	$-100 \leq x_i \leq 100$

4.1.1 The Best-so-far Method (Exploitation)

In order to improve the efficiency of the exploitation process by onlooker bees, we proposed to modify the parameters that are used to calculate new candidate food sources. In the original algorithm, each onlooker bee selects a food source based on a probability that varies according to the fitness function explored by a single employed bee. Then the new candidate solutions are generated by updating the onlooker solutions as shown in equation (2.2). In our best-so-far method, from the pseudo-code line number 20 to 32, all onlooker bees use existing information from all employed bees to make a decision on a new candidate food source. Thus, the onlookers can compare information from all candidate sources and are able to select the best-so-far position.

On the assumption that the best-so-far position will lead to the optimal solution, from the pseudo-code line number 33 to 46, we bias the solution direction by using the best-so-far solutions-based approach to update the new candidate solutions of onlooker bees. We then accelerate its convergence speed by using the fitness value of the best-so-far food source as the multiplier of the error correction for this solution update. The values in all dimensions of each food source are also updated in each iteration in order to increase the diversity of the feasible solutions in the search space.

The new method used to calculate a candidate food source is shown in Equation (4.1)

$$v_{id} = x_{ij} + \Phi f_b(x_{ij} - x_{bj}) \tag{4.1}$$

where v_{id} = The new candidate food source for onlooker bee position *i* dimension *d, d=1,2,3,...D*
x_{ij} = The selected food source position *i* in a selected dimension *j*
Φ = A random number between -1 to 1
f_b = The fitness value of the best food source so far
x_{bj} = The best so far food source in selected dimension *j*

This change should make the best-so-far algorithm converge more quickly because the solution will be biased towards the best-so-far solution. Figure 4.2 illustrates the effect of our Best-so-far method on convergence speed.

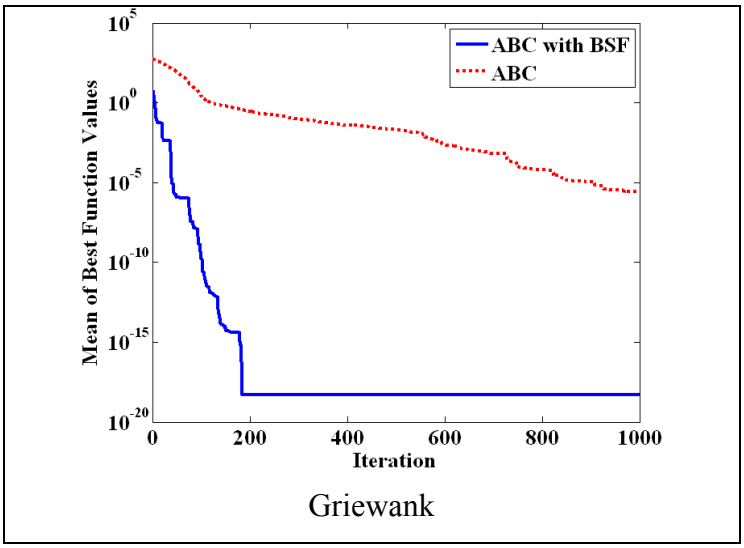


Figure 4.2 Iterations to convergence for ABC and ABC with Best-so-far method (ABC with BSF)

The plot shows that the solution obtained from ABC with Best-so-far method can quickly converge to the best solution found so far (in each iteration). Onlooker bees exploit the best-so-far solution provided by employed bees to bias their search direction. Consequently, when the number of iterations is increased, the solution quality of ABC with Best-so-far method is improved quickly.

4.1.2 The Adjustable Search Radius (Exploration and Exploitation)

Although our best-so-far method can increase the local search ability compared to the original ABC algorithm, the solution is easily entrapped in a local optimum. In order to resolve this issue, we introduce the improvement on both exploitation and exploration based on a global search ability of the scout bee.

From the pseudo-code line number 47 to 59, the scout bee will randomly generate a new food source by using equation (4.2) whenever the solution stagnates in the local optimum.

$$v_{ij} = x_{ij} + \emptyset_{ij}\left[\omega_{max} - \frac{iteration}{MCN}(\omega_{max} - \omega_{min})\right]x_{ij} \tag{4.2}$$

Where v_{ij} is a new feasible solution of a scout bee that is modified from the current position of an abandoned food source (x_{ij}) and $\emptyset_{ij}$ is a random number between [-1,1]. The value of ω_{max} and ω_{min} represent the maximum and minimum percentage of the position adjustment for the scout bee. The value of ω_{max} and ω_{min} are fixed to 1 and

0.2 respectively. These parameters were chosen by the experimenter. With these selected values, the adjustment of scout bee's position based on its current position will linearly decrease from 100 percent to 20 percent in each experiment round, i.e. a scout bee will utilize the exploration process in the early step and will employ the exploitation process by using existing information of solution in the later steps.

Based on the assumption that the solution of scout bee will be far from the optimal solution in the first iteration and it will converge closely to the optimal solution in later iterations, equation (4.2) will dynamically adjust the position of scout bee by allowing a scout bee in the first iteration to wander with a wider step size in the search space. As the number of the iteration increases, the step size for the wandering of a scout bee will decrease.

From the pseudo-code line number 7 to 19, the employed bees can thus still maintain diversity in producing a new food source (equation 2.2). Using both the local and the global search methods, the convergence speed increases and solutions can be globally optimized more quickly.

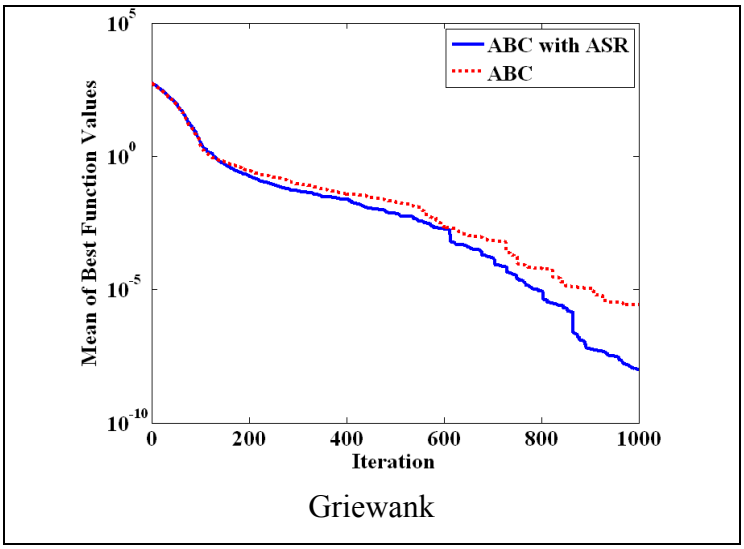


Figure 4.3 Iterations to convergence for ABC and ABC with Adjustable Search Radius (ABC with ASR)

The effect of adjustable search radius is illustrated in Figure 4.3. The result shows that the exploitation process introduced to a scout bee can improve solution quality of the ABC algorithm. The scout bee with adjustable search radius ability can also use existing information to derive a better solution when compared to a scout bee in original ABC that has only exploration ability.

4.1.3 Objective-value-based Comparison Method

In this work, we also focus on the method that is used to compare and to select between the old solution and the new solution in each iteration. Basically, the comparison of the new solution and the old solution is done by the fitness value. If the fitness of the new solution is better than the fitness of the old solution, we select the new one and ignore the old solution. The fitness value can be obtained from the following equation.

For finding the minimum objective value

$$Fitness\big(f(x)\big) = \begin{cases} \dfrac{1}{1+f(x)} & \text{if } f(x) \geq 0 \\ 1+|f(x)| & \text{if } f(x) < 0 \end{cases} \tag{4.3}$$

Based on equation 4.3, we can see that when $f(x)$ is larger than the zero but has a very small value, e.g. 1E-20, the fitness value of equation $\frac{1}{1+1E-20}$ is rounded up to be 1 (1E-20 is ignored). This will lead the fitness of all solutions to become equal to 1 in the later iterations. In other words, there is no difference between the fitness values that is equal to $\frac{1}{1+1E-20}$ and $\frac{1}{1+1E-120}$. Thus, a new solution that gives a better fitness value than the old solution will be ignored and the solution will stagnate at the old solution. In order to solve this issue, we directly use the objective value of function for comparison and selection of the better solution. The pseudo-code of the new comparison process is shown in Figure 4.4.

The old method for comparing the solutions	The objective-value-based comparison method
If ($Fitness(f_{new}(x)) > Fitness(f_{old}(x))$**)** **Replace the old solution with the new solution** **Else** **Keep the old solution**	**If (Finding Minimum)** **If (**$f_{new}(x) < f_{old}(x)$**)** **Replace the old solution with the new solution** **Else** **Keep the old solution** **Else If (Finding Maximum)** **If (**$f_{new}(x) > f_{old}(x)$**)** **Replace the old solution with the new solution** **Else** **Keep the old solution**

Figure 4.4 The comparison between the old and the new method for comparing the solutions

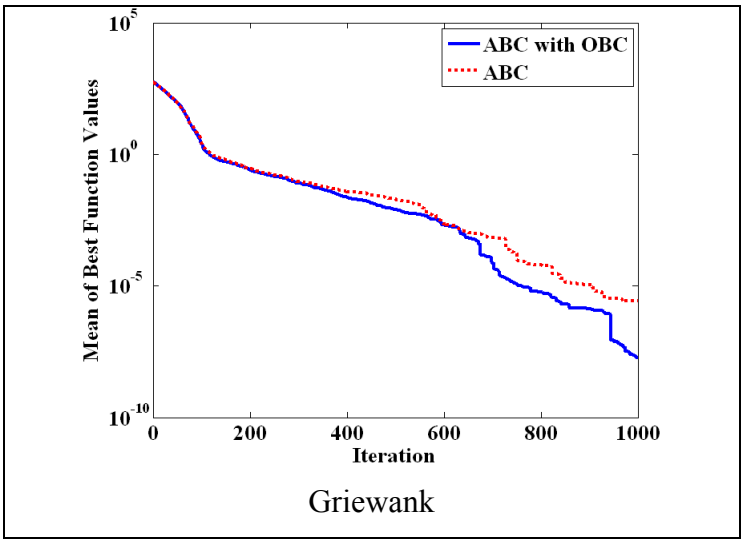


Figure 4.5 Iterations to convergence for ABC and ABC with Objective-value-based Comparison Method (ABC with OBC)

From Figure 4.5, it can be seen that our objective-value-based comparison method can help ABC to avoid an issue of solution stagnation. In original ABC, the solution can sometimes be hard to improve because the fitness values of the old and the new solution are too similar. With the objective-value-based comparison method, the new solution that gives the same fitness value but provides a better objective value will be selected and thus improve the overall solution quality.

From these 3 experiments, we found that the Best-so-far method helps onlooker bees to bias their search direction to the best solution found so far in each iteration of every test case. The improvement is thus obvious in most experiments. On the other hand, the benefit of adjustable search radius and objective-based comparison methods can only be observed when the solution is stagnated in a local optimum. Consequently, we can conclude that the Best-so-far method most affects the performance and solution quality of the algorithm.

4.2 Sensitivity of the Parameters Setting

In order to analyze the sensitivity of the parameters setting of Best-so-far ABC, we experiment with different population sizes and limit values. The population size refers to the colony size. A limit value is a parameter used to control scout production frequency. If a limit value is small, the scout production frequency is high.

Variation in Colony Sizes

In Table 4.2, the mean of the best function values with the colony sizes of 20, 40, and 100 are presented. The iterations to convergence are also shown in Figure 4.6 - 4.8.

The experimental results show the evidence that as the colony size increases, the Best-so-far ABC algorithm produces better results. The reason for the improvement is the fact that the diversity of the feasible solutions in the search space is increased with the population size. However, the improvement is not observed indefinitely (shown in Ackley function). In other words, there is a balanced point of population size for each problem configuration and an increment beyond that point can no longer improve the performance.

Table 4.2 The Mean of the best function values obtained with 1000 iterations using different colony sizes

Function	D	Colony Size					
		20 (n_e=n_o=*10*)		40 (n_e=n_o=*20*)		100 (n_e=n_o=*50*)	
		Mean	S.D.	Mean	S.D.	Mean	S.D.
Sphere	50	1.28E-40	5.74E-40	1.17 E-90	4.24 E-90	3.69E-221	2.76E-220
Griewank	50	4.46E-25	1.99E-24	1.44E-120	6.40E-120	4.65E-275	3.54E-274
Rastrigin	50	1.53E-28	6.84E-28	3.67E-93	1.64E-92	8.65E-272	2.05E-271
Rosenbrock	50	1.16514	2.24618	0.418848	1.07602	0.0757589	0.160013
Ackley	50	2.98E-7	1.33E-6	0	0	0	0
Schaffer	2	2.00E-179	4.23E-178	0	0	0	0

D, number of dimensions; n_e, number of employed bees; n_o ,number of onlooker bees

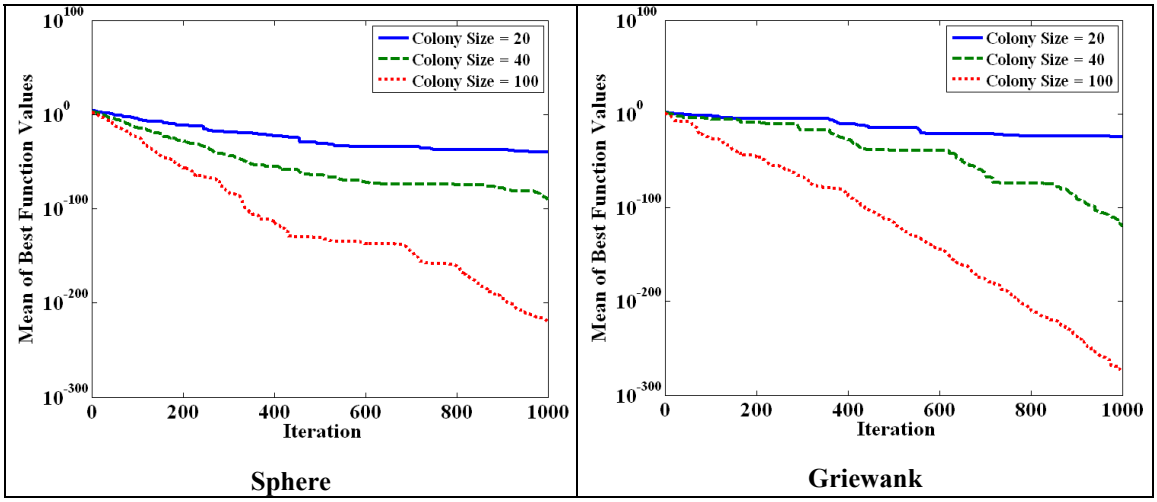


Figure 4.6 Iterations to convergence on Sphere and Griewank

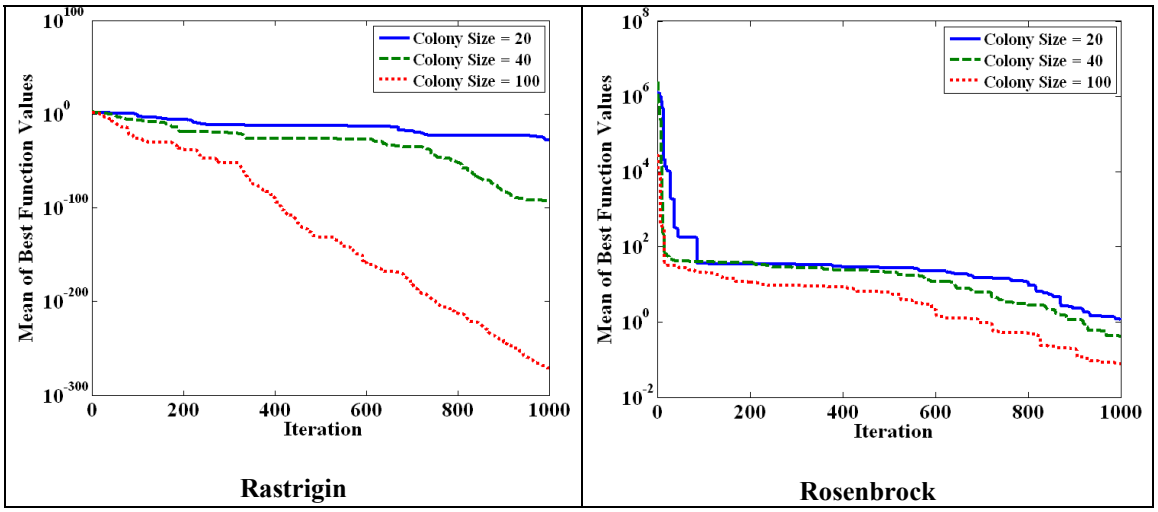


Figure 4.7 Iterations to convergence on Rastrigin and Rosenbrock

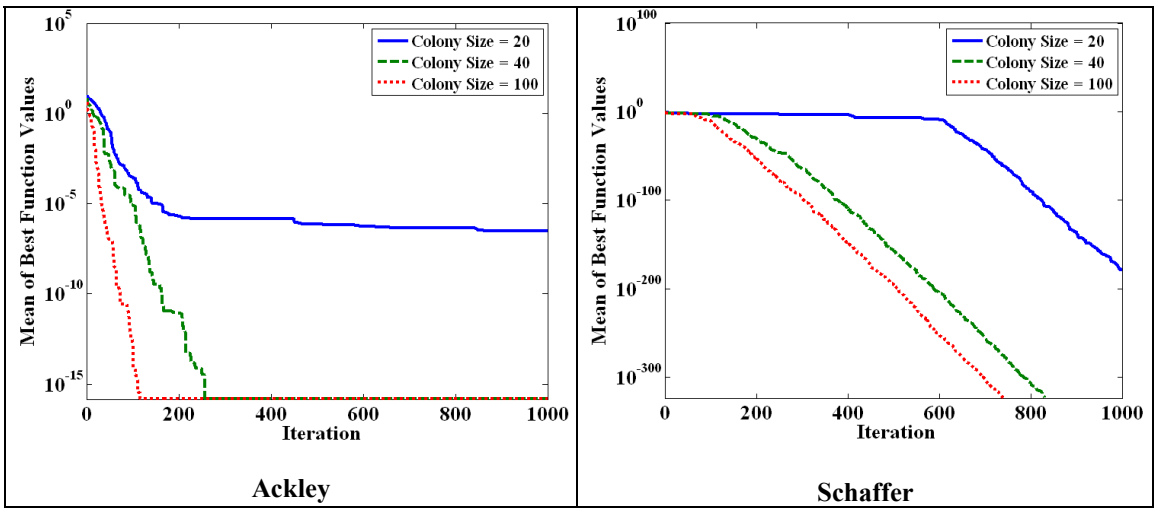


Figure 4.8 Iterations to convergence on Ackley and Schaffer

Variation in Limit Values

The "limit" values of the scout production can be defined with two variables; n_e is the number of employed bees, and D is the number of problem dimension. We analyze the effect of the scout production process on the performance of Best-so-far ABC with different "limit" values: 0.1 x n_e x D, 0.5 x n_e x D, and 1.0 x n_e x D. We also investigate these limit values on different colony sizes (20, 40, and 100).

Figure 4.9 presents the iterations to convergence of the mean best values obtained from different "limit" values. In this Chapter, only results of the Griewank function are discussed. The detailed information for other test functions is presented in Appendix C and is summarized in Table 4.3.

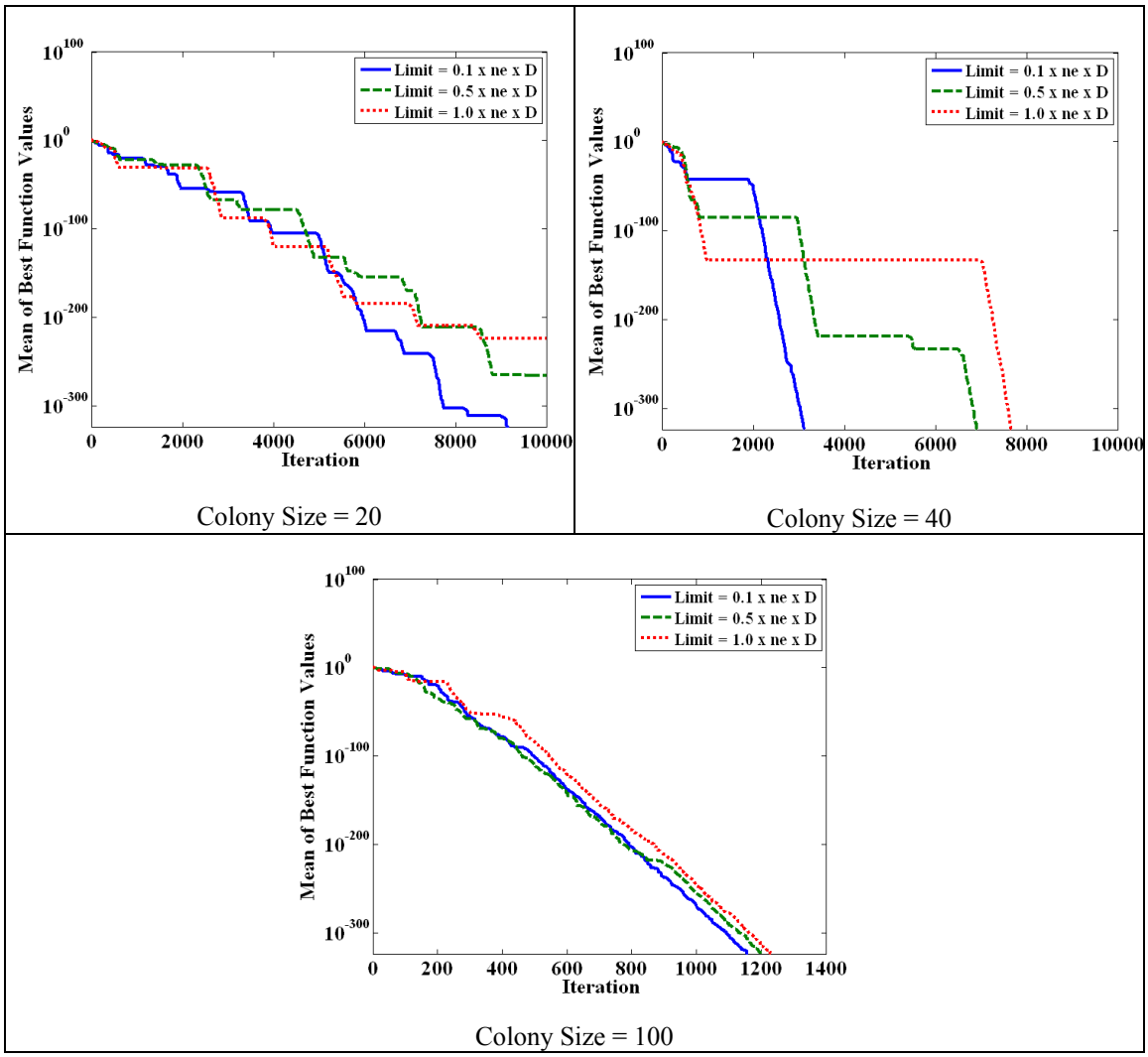


Figure 4.9 Iterations to convergence under different "limit" values on Griewank function

Figure 4.9 and Table 4.3 show that the scout production limit affects the performance of Best-so-far ABC for small colony sizes. The solution quality and convergence speed are improved when the scout production limit is high. However, the effect of the "limit" becomes much smaller when a large colony size is used. In other words, higher production limit only gives slightly better solutions in a case of a large colony size. It can be concluded that the diversity of the feasible solutions is sufficiently provided by the large population size. Thus, the diversity controlled by the scout production will give smaller effect to the algorithm in this case.

Table 4.3 Sensitivity of the parameters setting

Function	MCN	Dimension (*D*)		Colony Size								
				20 (n_e=n_o=*10*)			40 (n_e=n_o=*20*)			100 (n_e=n_o=*50*)		
				Limit = 0.1xn_ex*D*	Limit = 0.5xn_ex*D*	Limit = *1.0*xn_ex*D*	Limit = 0.1xn_ex*D*	Limit = 0.5xn_ex*D*	Limit = *1.0*xn_ex*D*	Limit = 0.1xn_ex*D*	Limit = 0.5xn_ex*D*	Limit = *1.0*xn_ex*D*
Sphere	**10000**	**50**	**Mean**	2.10E-217	1.19E-180	1.07E-57	0	0	0	0	0	0
			S.D.	3.12E-216	2.47E-179	4.77E-57	0	0	0	0	0	0
Griewank	**10000**	**50**	**Mean**	0	3.30E-266	1.66E-224	0	0	0	0	0	0
			S.D.	0	4.26E-265	2.12E-223	0	0	0	0	0	0
Rastrigin	**10000**	**50**	**Mean**	0	4.88E-121	2.43E-99	0	0	0	0	0	0
			S.D.	0	2.18E-120	1.09E-98	0	0	0	0	0	0
Rosenbrock	**10000**	**50**	**Mean**	0.79444	0.01888	0.00048	0.03283	5.55E-6	7.87E-17	5.32E-6	1.22 E-29	0
			S.D.	1.68993	0.03532	0.00130	0.08266	2.18E-5	2.96E-16	1.47E-5	5.46E-29	0
Ackley	**10000**	**50**	**Mean**	0	0	0	0	0	0	0	0	0
			S.D.	0	0	0	0	0	0	0	0	0
Schaffer	**2000**	**2**	**Mean**	0	0	0	0	0	0			
			S.D.	0	0	0	0	0	0			

n_e, Number of employed bees; n_o, Number of onlooker bees; D, dimension of the problem; runs = 20

4.3 Computational Complexity

We can evaluate the computational complexity of the best-so-far ABC algorithm in comparison with the original ABC and the BSO algorithms. Let n be the total number of bees, d be the dimension of solutions, and m be the maximum iteration. Considering each component of the algorithm in turn:

1. *Initialization*: The time used in this step is dominated by the time used to assign the initial solution on all employed bees ($\frac{n}{2}$). The time required, T_1, is

$$T_1 = (\frac{n}{2})d$$

2. *Updating solutions for all employed bees*: The computation associated with the updating solutions of employed bees is performed over the all employed bees. The time required, T_2, is

$$T_2 = (\frac{n}{2})d$$

3. *Selecting the best-so-far from all employed bees*: This step involves finding the best-so-far solution from all employed bees. The time required, T_3, is

$$T_3 = (\frac{n}{2})d$$

4. *Updating solutions for all onlooker bees*: The computation associated with the updating solutions of onlooker bees is performed over the all onlooker bees ($\frac{n}{2}$) based on the best-so-far solution obtained from previous step. The time required, T_4, is

$$T_4 = \left(\frac{n}{2}\right)(2d)$$

5. *Finding and updating abandoned solution by a scout bee*: This computation involves finding and updating a solution when it is considered as abandoned solution by a scout bee. The time required, T_5, is

$$T_5 = \left(\frac{n}{2}\right)(2d)$$

The total time to compute for one iteration, T, is thus $T_1 + T_2 + T_3 + T_4 + T_5$. The total execution time for iteration of maximum iteration m can then be defined as

$$T_{Best-so-far\ ABC} = T_1 + m(T_2 + T_3 + T_4 + T_5)$$

$$T_{Best-so-far\ ABC} = \left(\frac{n}{2}\right)d + m\left[\left(\frac{n}{2}\right)d + \left(\frac{n}{2}\right)d + \left(\frac{n}{2}\right)(2d) + \left(\frac{n}{2}\right)(2d)\right] = \left(\frac{n}{2}\right)d + m(3nd)$$

When we consider the computational complexity of the original ABC, we have found that it uses the computational time T_1 and T_2 the same as Best-so-far ABC but T_3 is not used in the original ABC. T_4 and T_5 of the original ABC are equal to $\left(\frac{n}{2}\right)d$ and $\left(\frac{n}{2}\right)d + d$ respectively. The computational complexity of the ABC thus can be described as

$$T_{ABC} = T_1 + m(T_2 + T_4 + T_5)$$

$$T_{ABC} = \left(\frac{n}{2}\right)d + m\left[\left(\frac{n}{2}\right)d + \left(\frac{n}{2}\right)d + \left(\frac{n}{2}\right)d + d\right] = \left(\frac{n}{2}\right)d + m(\frac{3}{2}nd + d)$$

In summary, the best-so-far ABC algorithm must spend time necessary to find the best-so-far solution in each iteration, so it uses the computational time equal to $\left(\frac{n}{2}\right)d + m(3nd)$. This is larger than the computational time of the original ABC algorithm which is equal to $\left(\frac{n}{2}\right)d + m(\frac{3}{2}nd + d)$. The BSO algorithm must spend time to sort all bees based on their fitness in each of iteration of algorithm, so the computational time is equal to $nd + m(n + nd \log n + \frac{5}{2}nd))$. However, when we focus on the upper bound of the complexity of problems, both Best-so-far ABC and original ABC algorithms use computational time $O(mnd)$ while the BSO algorithm uses $O(mnd \log n)$, which is the highest complexity.

Although the computational complexity of the best-so-far ABC is higher than that of the original ABC at the same maximum iteration, the best-so-far ABC should be able to find the optimal solution before it reaches the maximum iteration. Thus the actual computation time of best-so-far ABC may be reduced in relative to original ABC. The execution of the results will be discussed in the next Chapter.

CHAPTER 5 ALGORITHMS COMPARISON AND EVALUATION

In this Chapter, the performance of the proposed technique is compared with the original technique and the state of the art algorithms using numerical benchmarking functions. The experimental methodology is presented and the results are illustrated.

5.1 Numerical Experiment Methodology

The aim of this experiment is to compare the performance of the search process for the best-so-far method with the original algorithm and the Bee Swarm Optimization that is the state-of-the-art algorithm for solving numerical optimization based on behavior of bees proposed by (Akbari, et al., 2010). Unimodal and multimodal benchmark functions as shown in Table 4.1 were used in this experiment. The objective of the search process is to find the solutions that can produce the minimum output value from these benchmark functions. In other words, the aim is to minimize $f_i(\vec{x})$ in Table 4.1. In order to make a performance comparison, we implemented the original ABC based on an algorithm given in (Karaboga, 2009) as well as our best-so-far method. Our original ABC code was also validated by comparing the benchmark results with the previous work proposed in (Karaboga and Basturk, 2007). Since we could not obtain the source code of the BSO algorithm, we only used the results reported in (Akbari, et al., 2010) to compare with our algorithm. As a result, the BSO algorithm will not be included in some comparison experiments.

The experiments were conducted in a similar fashion as described in (Karaboga and Basturk, 2007; Akbari, et al., 2010). The number of employed and onlooker bees were set to 100. The values of ω_{max} and ω_{min} are set to 1 and 0.2 respectively. Each of the experiments was repeated 100 times with different random seeds. All the experiments in this paper were run on the same hardware (Intel Core 2 Quad with 2.4 GHz CPU and 4 GB memory). The number of iterations to convergence, the runtime (execution time), and the mean and standard deviation of the output values of benchmark functions were recorded. Note that lower mean objective values indicate better solutions.

In order to investigate the effect of the iteration increment on the output quality, we divided the experiments into two sets, called ABC1 and ABC2. The maximum numbers of iterations proposed in Karaboga and Basturk, 2007; Akbari, et al., 2010 were also used in our work. In the ABC1 experiment, for all benchmark functions with exceptions of Schaffer function, the numbers of maximum iterations were 500, 750 and 1000 for the dimensions of 10, 20 and 30, respectively. In ABC2, the maximum numbers of iterations were 5000, 7500 and 10000 for the dimensions of 10, 20 and 30, respectively. For Schaffer function in both of ABC1 and ABC2 experiment, the numbers of maximum iterations was set to 2000. Note that one iteration of our method requires a longer execution time than the original method because the values in all dimensions of each candidate food source of onlooker bees are updated on every iteration and the algorithm spends time to find the best-so-far solution in each iteration. However, the number of iterations to convergence is smaller. Thus, we will look at both the number of iterations and runtime when we consider convergence speed.

5.2 Numerical Results and Discussion

The results obtained by the original and the best-so-far methods based on the numerical benchmark functions are shown in Table 5.1. Note that there is no result of the BSO algorithm to compare in the ABC1 experiment.

The “Convergence Iteration” column shows the number of iterations needed for each algorithm to converge towards the optimal solution (value not exceeding the maximum number of iterations indicated in section 5.1). The ‘‘Runtime’’ column is the average runtime needed to reach the convergence state across a hundred runs. The “Mean” column is the average output values of benchmark functions. The “SD” column shows the standard deviation of the results. Mean and SD values which are smaller than 1E-304 are considered as 0.

Table 5.1 Result obtained in the ABC1 experiment

Function	dimension	Max iteration	ABC				Best-so-far ABC			
			Convergence Iteration	Runtime (s)	Mean of Objective values	SD	Convergence Iteration	Runtime (s)	Mean of objective values	SD
Sphere	**10**	500	500	0.234	1.426E-16	8.212E-17	500	0.281	**3.303E-188**	**3.299E-187**
	20	750	750	0.593	2.353E-12	2.198E-12	750	0.704	**1.118E-239**	**1.109E-238**
	30	1000	1000	1.109	2.649E-10	2.136E-10	1000	1.546	**5.6413E-302**	**3.967E-301**
Griewank	**10**	500	500	0.453	1.040E-03	2.741E-03	500	0.500	**2.157E-214**	**2.092E-213**
	20	750	750	1.203	4.734E-09	2.177E-08	750	1.344	**2.920E-302**	**2.801E-300**
	30	1000	1000	2.344	5.369E-09	2.664E-08	845	2.672	**0**	**0**
Rastrigin	**10**	500	500	0.344	4.911E-16	1.073E-15	500	0.391	**1.414E-210**	**1.410E-209**
	20	750	750	0.922	2.933E-11	5.728E-11	750	0.985	**3.567E-300**	**3.569E-299**
	30	1000	1000	1.782	3.593E-07	3.565E-06	863	1.953	**0**	**0**
Rosenbrock	**10**	500	500	0.469	0.07022	0.09035	462	0.465	**0**	**0**
	20	750	750	1.375	0.427111	0.611375	750	1.484	**7.198E-22**	**4.941E-21**
	30	1000	1000	2.704	0.681802	0.774655	1000	2.969	**3.604E-15**	**3.585E-14**
Ackley	**10**	500	500	0.390	1.400E-10	7.657E-11	63	0.047	**0**	**0**
	20	750	750	0.968	3.681E-07	1.302E-07	85	0.094	**0**	**0**
	30	1000	1000	1.782	5.246E-06	1.702E-06	109	0.234	**0**	**0**
Schaffer	**2**	2000	2000	0.672	7.895E-11	4.014E-10	947	0.468	**0**	**0**

Table 5.2 Result obtained in the ABC2 experiment

Function	dimension	Max iteration	ABC				BSO		Best-so-far ABC			
			Convergence Iteration	Runtime (s)	Mean of Objective values	SD	Mean of Objective values	SD	Convergence Iteration	Runtime (s)	Mean of objective values	SD
Sphere	**10**	5000	5000	2.390	9.472E-17	1.615E-17	8.475E-123	3.953E-122	833	0.578	**0**	**0**
	20	7500	7500	5.859	2.627E-16	8.796E-17	2.361E-115	7.348E-115	1003	1.234	**0**	**0**
	30	10000	10000	10.922	3.217E-16	5.142E-17	4.728E-102	1.372E-101	1115	2.141	**0**	**0**
Griewank	**10**	5000	5000	4.437	7.396E-05	7.396E-04	3.823E-46	6.679E-46	754	0.797	**0**	**0**
	20	7500	7500	12.037	2.054E-18	4.227E-19	8.402E-47	3.928E-46	828	1.735	**0**	**0**
	30	10000	10000	23.359	4.638E-18	7.744E-19	4.161E-47	7.523E-47	845	2.672	**0**	**0**
Rastrigin	**10**	5000	5000	3.453	9.934E-18	2.992E-18	4.171E-64	7.834E-64	767	0.672	**0**	**0**
	20	7500	7500	9.172	1.928E-17	3.579E-18	7.899E-61	2.215E-60	824	1.313	**0**	**0**
	30	10000	10000	17.406	2.962E-17	4.618E-18	6.228E-59	8.496E-59	863	1.953	**0**	**0**
Rosenbrock	**10**	5000	5000	4.672	0.00490144	0.0061382	3.617E-07	5.081E-05	462	0.465	**0**	**0**
	20	7500	7500	13.453	0.00541073	0.0090507	9.201E-07	1.102E-05	1191	2.469	**0**	**0**
	30	10000	10000	26.625	0.00771188	0.0151573	5.865E-06	8.519E-05	2169	6.469	**0**	**0**
Ackley	**10**	5000	5000	3.906	4.763E-15	1.636E-15	7.105E-19	5.482E-18	63	0.047	**0**	**0**
	20	7500	7500	9.657	1.546E-14	2.183E-15	2.131E-19	4.603E-18	85	0.094	**0**	**0**
	30	10000	10000	17.860	2.789E-14	2.843E-15	1.460E-18	8.130E-17	109	0.234	**0**	**0**
Schaffer	**2**	2000	2000	0.672	7.895E-11	4.014E-10	1.069E-49	3.170E-49	947	0.468	**0**	**0**

In Table 5.1, we compare the results using approximately the same runtime value. The best-so-far method gives better solutions than the original algorithm in all cases of benchmark functions. Especially, the global minimal values of Ackley and Schaffer functions were found by the best-so-far method in ABC1 experiment.

The results from the ABC1 experiment showed that several benchmark functions did not converge within the specified number of iterations. The maximum number of iterations was then increased in the ABC2 experiment. The results are shown in Table 5.2.

In the ABC2 experiment in Table 5.2, when the number of iterations was increased, the solution quality was also improved in the original ABC as shown in mean of objective values column. However, our best-so-far method was still able to generate a better solution on all the benchmark functions. The average improvement on runtime for the best-so-far method when compared with the original ABC was 82%, and the maximum and minimum runtime improvement were 99% and 30% for Ackley and Schaffer function respectively. For several benchmark functions, the original ABC never reached the mean value achieved by our best-so-far method even when the runtime was increased.

The results from the BSO method reported in (Akbari, et al., 2010) were also used to compare with the Best-so-far method in the ABC2 experiment. The results showed that the mean value of Best-so-far method is better than that of the BSO method at the same number of the iterations on all benchmark functions.

Figure 5.1 shows the comparison of the convergence speed (Iterations) in terms of number of iterations between the original and the best-so-far method. Notice that the best-so-far method converges substantially faster with a much smaller number of iterations needed.

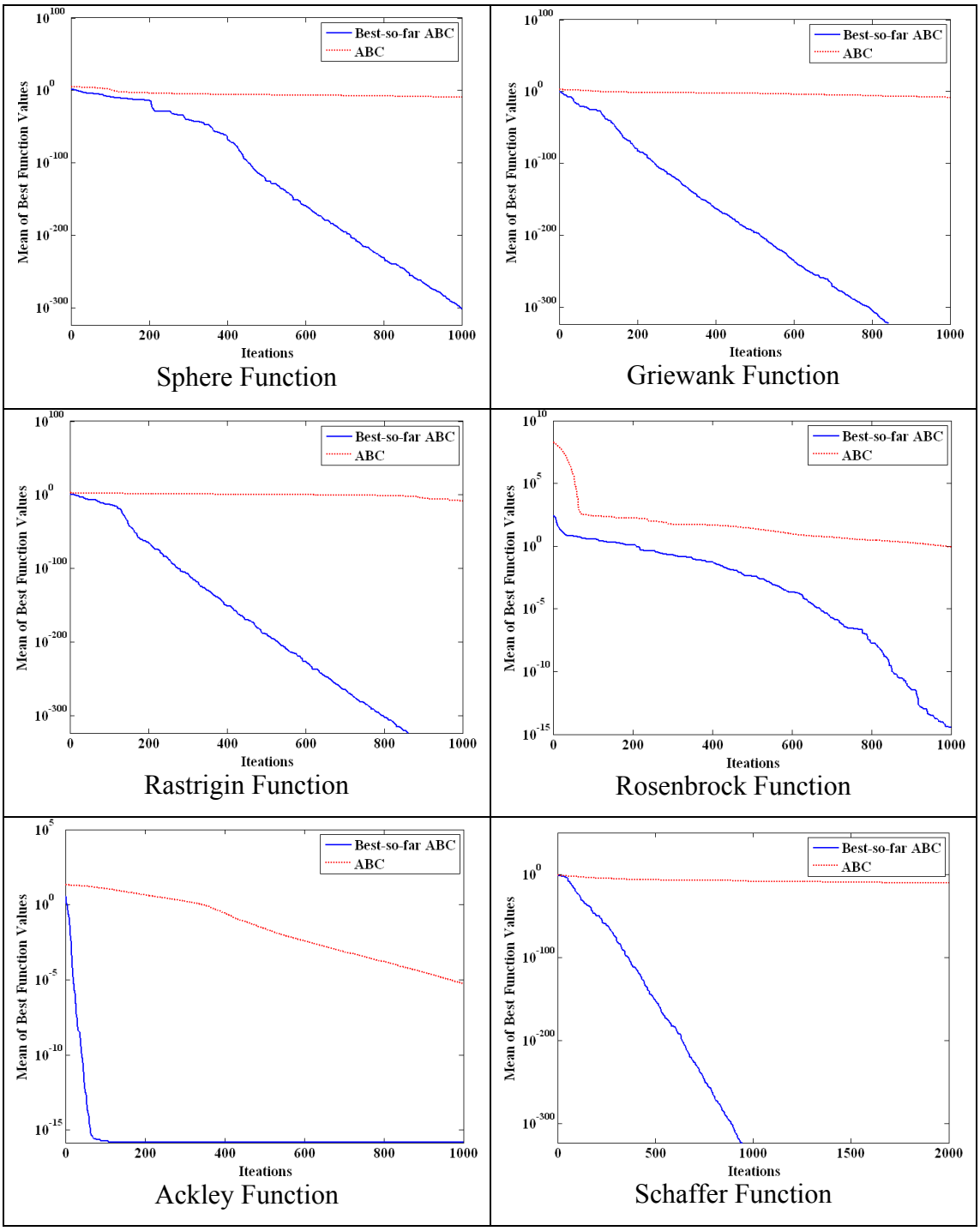


Figure 5.1 Iterations to convergence for original and best-so-far algorithms when dimension = 30 (dimension of Schaffer Function = 2)

Note that there is no result of the BSO algorithm to illustrate the convergence speed, so the comparison with the BSO algorithm is not included here.

The rate of convergence is used to indicate the speed at which a converging sequence approaches its limit. A smaller value of the rate of convergence implies that the solution converges to an optimal value more quickly. Suppose that the sequence $\{x_k\}$ converges to the number L, so the rate of convergence can be calculated from equation as follows

$$rate\ of\ convergence = \lim_{k\to\infty} \frac{|x_{k+1} - L|}{|x_k - L|} \tag{5.1}$$

Table 5.3 Rates of convergence in the ABC2 experiment

Function	Dimensions	Rate of convergence	
		ABC	Best-so-far ABC
f1	10	0.9265	0.5856
Sphere	20	0.9942	0.6691
	30	0.9954	0.7124
f2	10	0.9966	0.5358
Griewank	20	0.9890	0.6032
	30	0.9908	0.5829
f3	10	0.9270	0.5622
Rastrigin	20	0.9662	0.6073
	30	0.9787	0.6246
f4	10	0.9965	0.8705
Rosenbrock	20	0.9972	0.9492
	30	0.9978	0.9722
f5	10	0.9928	0.5683
Ackley	20	0.9948	0.6791
	30	0.9964	0.7445
f6	2	0.9907	0.6849
Schaffer			

The results in Table 5.3 show that both the ABC and the Best-so-far ABC converge linearly to the limit (global minimum). However, when we compare the rates of convergence, we found that the Best-so-far ABC gives better results than the ABC on all benchmark functions, especially in the Griewank and Rastrigin functions.

In summary, the results in the numerical experiments indicate that the best-so-far ABC method can converge to the optimal solution more quickly on almost all benchmark functions when it is compared with the original ABC method. The results also show that the best-so-far ABC method can produce better solutions than the BSO algorithm that is the state-of-the-art algorithm. In other words, fewer iterations were needed to converge and thus, less computational time was required. Notice that the SD is relatively low, which implies consistency among experimental runs.

Although the results show that our Best-so-far ABC overcomes other aforementioned methods, it still requires a lot of time when the problem size is large. To solve this problem, in the next Chapter, the distributed version of the Best-so-far ABC will be introduced to handle a large problem size while keeping the solution quality at a reasonable level.

CHAPTER 6 BEST-SO-FAR ABC ON A DISTRIBUTED ENVIRONMENTS

In Best-so-far ABC described previously, the best feasible solutions found so far are shared globally among the entire bee population. Thus, the new candidate solutions are more likely to be close to the current best solution. This will make the algorithm converge more quickly than the original ABC.

However, many real-world problems are extremely time sensitive. In these domains, finding a good quality solution is not enough; the algorithm must produce a good solution in a limited amount of time. To reduce computation time, we propose using a variant of ABC that allows calculations to be distributed to multiple computational units. Since different parts of the overall search can be executed in parallel, the overall convergence time should decline.

6.1 Best-so-far ABC with Multiple Patrilines

Our distributed approach adopts the concept of “patrilines”, which is another bee-inspired idea. In nature, a honey bee queen may mate with multiple males from her colony. Offspring from the same male will form a cluster within a colony, called patriline (from the Latin *pater* for "father"). Worker bees from different patrilines will thus be genetically distinct. Research has shown that such diversity enhances foraging productivity (Girard et al., 2011).

Inspired by this multiple patrilines concept, we designed an algorithm to further improve the diversity of solutions in the search space of our Best-so-far ABC. The colony of artificial bees will be divided into multiple patrilines, where each patriline consists of its own set of employed, onlooker, and scout bees. The exploitation (by employed and onlooker bees) and exploration (by scout bees) processes are handled independently within each patriline. Thus, our work has some conceptual similarity to the multi-colony ACO approach of Yang and Kamel (2006).

Moreover, patrilines exchange solution information in order to select the best possible solution. In our work, we use the hypercube concept (González et al., 1995) as an information exchange model in search spaces. In this concept, a hypercube of dimension

d consists of 2^d patrilines labeled from 0 to $2^d - 1$. Every patriline communicates only with its d adjacent neighbors, one in each dimension of the d-cube.

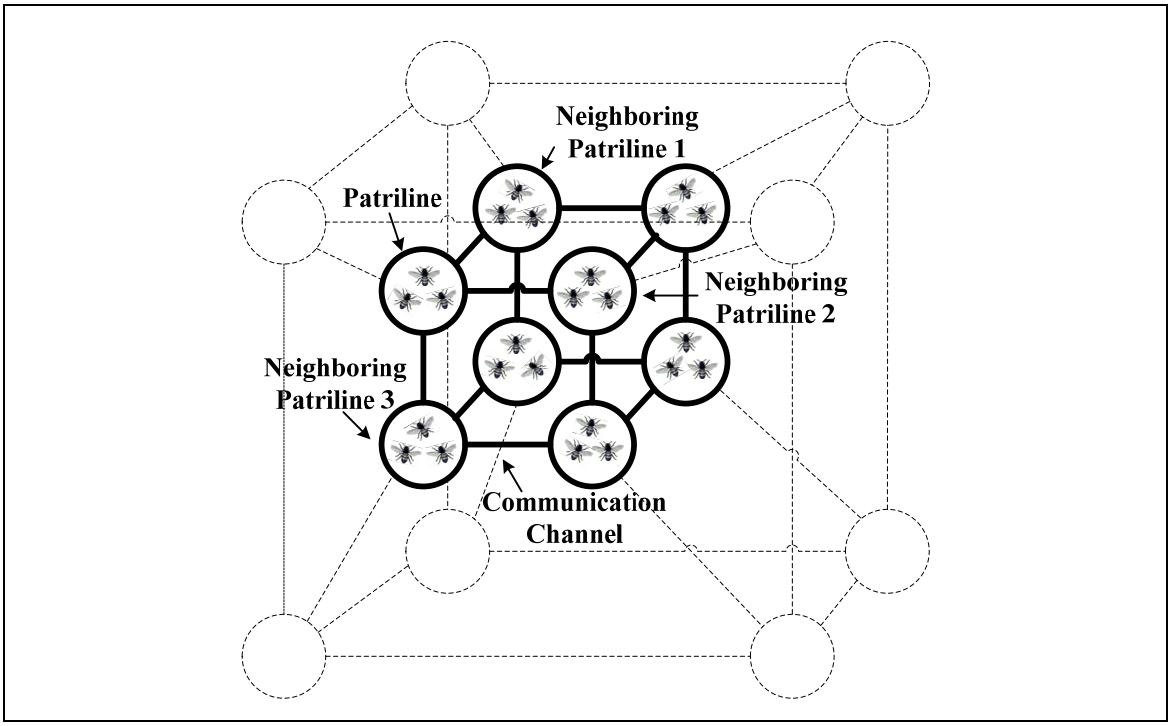


Figure 6.1 The patriline cube

Figure 6.1 shows the example of an information exchange model in a cube, where the number of patrilines is set to 8 (d = 3). If the number of patrilines used is increased, the information exchange network will form a hypercube. For example, a 4-D hypercube will be used for 9-16 patrilines as illustrated with dotted-line in Figure 6.1.

Information on the local best food source found by a patriline will be distributed to the neighboring patrilines. Each patriline then selects the best food source from the solutions provided by its neighbors and replaces its own local worst food source with the newly selected one. We select only one best solution from all neighbors in order to avoid stagnation that may occur from premature convergence to a local optimum. The information exchange model is shown in Figure 6.2.

This distributed approach should improve the search quality because various patrilines maintain some degrees of independence and thus potentially explore different regions of the search space.

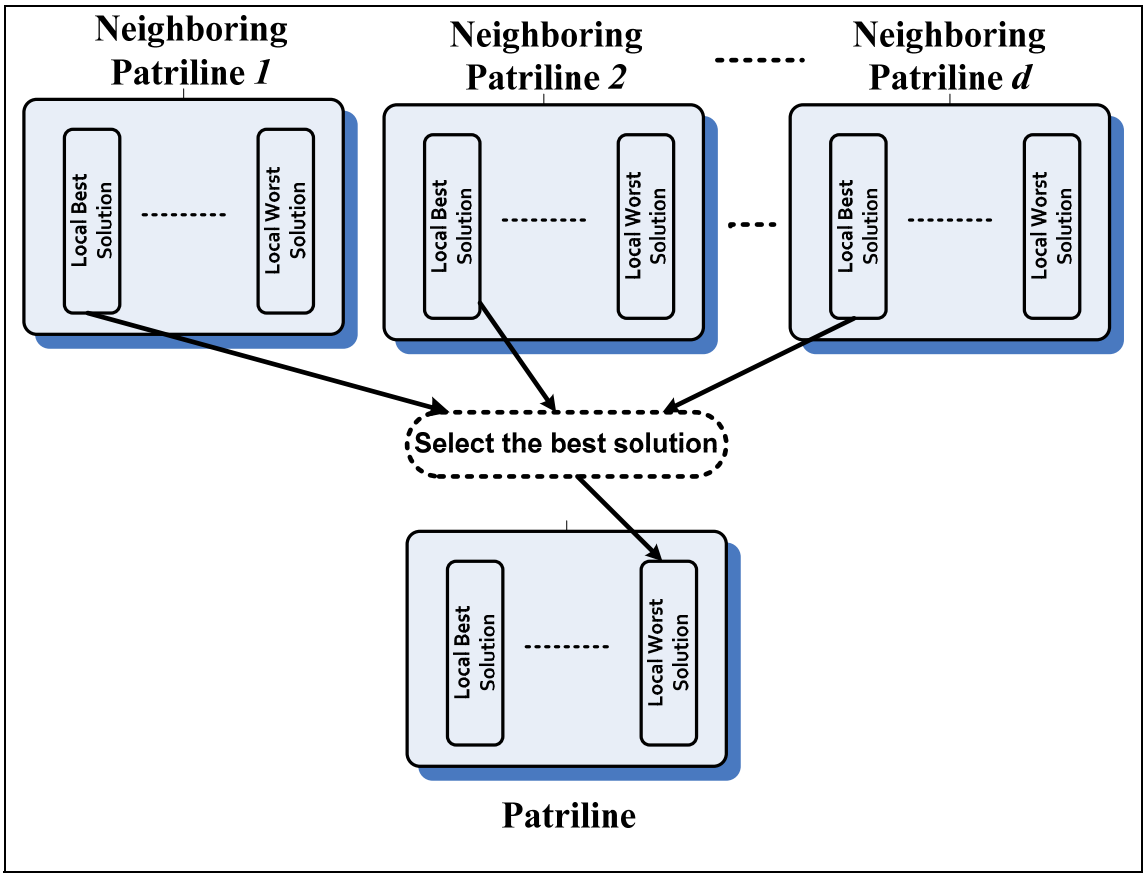


Figure 6.2 The information exchange method

In addition to the independence property, each patriline can also function concurrently with other patrilines. This should enhance the speed of the overall computation. A separate computational process can be assigned to each patriline allowing parallel execution across computing units. A message passing mechanism (Quinn, 2004) is used to support interaction among the processes.

We based our implementation on the commonly used Manager-worker model (Grama, 2003). One of the patrilines will be selected as the foraging manager. The remaining patrilines are called the workers. The manager is responsible for initializing and coordinating the computation and displaying the global results, as well as acting like a worker to perform its own search and information exchange. Inter-processor communication is implemented based on the message-passing interface API (MPI), which is a de facto standard in distributed computing (Quinn, 2004).

The details of Best-so-far ABC with multiple patrilines are presented in the flowchart shown in Figure 6.3.

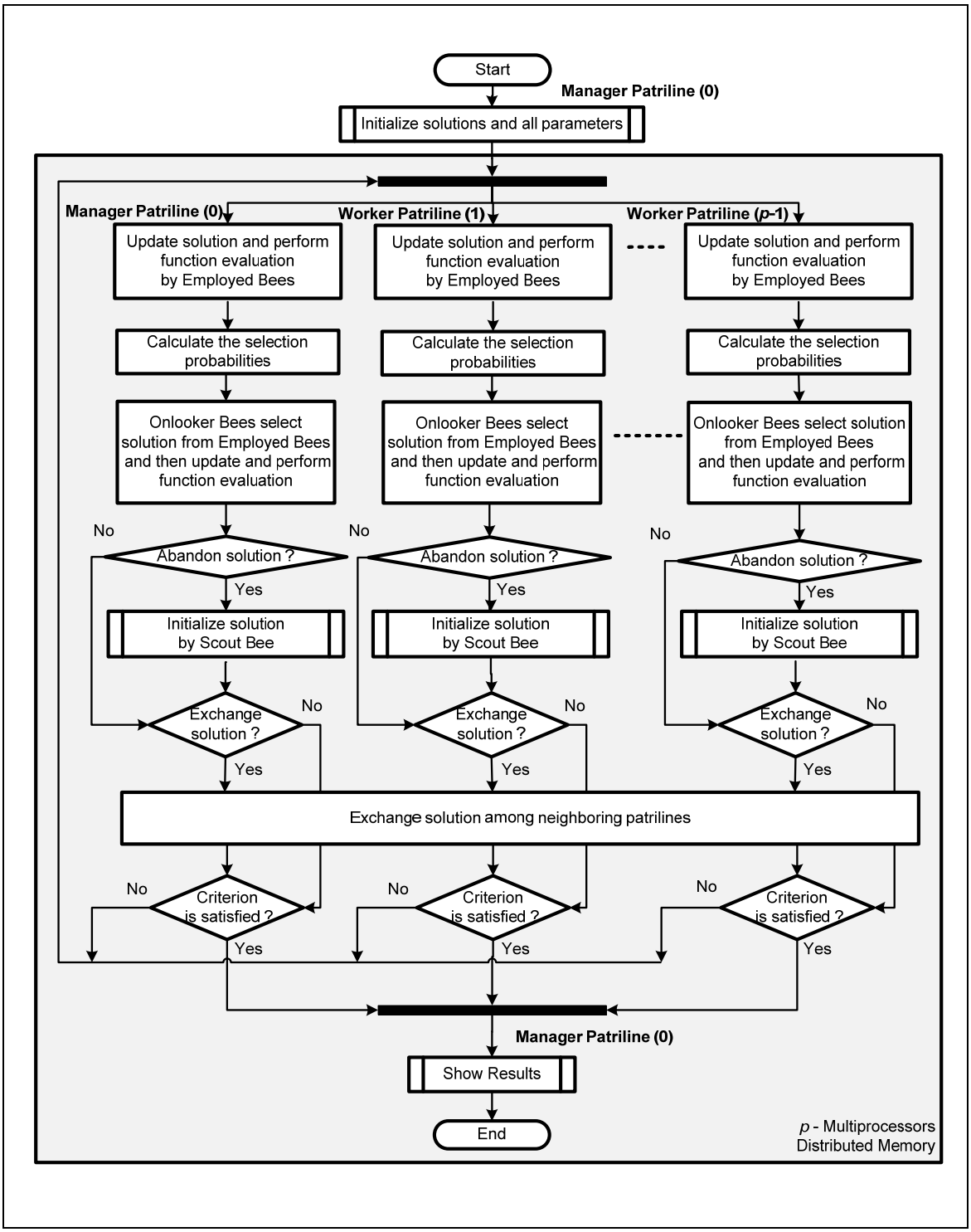


Figure 6.3 The Best-so-far ABC with multiple patrilines flowchart

Let P be the number of patrilines, N be the total number of food sources, and N_p be the number of food sources in the search space of each patriline. The initial food sources are allocated equally among all patrilines using the equation, $N_p = N/P$.

The algorithm first randomly initializes solutions consisting of a set of food source positions. To insure that the different sets of initial solutions represent different areas of the search space, we initialize food sources using a different random seed number in each patriline. The objective function that determines how good a food source is denoted by $F(x_i)$.

$$F(x_i)\,, x_i \in R^D\,, i \in \{1,2,3..,N\}, \tag{6.1}$$

where x_i is the position of a food source as a D-dimensional vector and R^D is the search space. The definition of $F(x_i)$ depends on the type of problem. For clustering, it will be related to the sum of intra-cluster distances, which should be as small as possible.

Each patriline incorporates a set of food sources, employed bees and onlooker bees.

The operational steps below are then carried out in parallel for all patrilines.

1. **Solution Updating by Employed Bees:** Employed bees in each patriline independently update their local food sources using equation (6.2) below.

$$\tilde{x}_{ij} = x_{ij} + \emptyset_{ij}(x_{ij} - x_{kj}) \tag{6.2}$$

   In the equation (3.2), $\tilde{x}_{ij}$ is a new feasible position of a food source that is modified from its previous position value (x_{ij}) based on a comparison with a randomly selected position from its neighboring food source (x_{kj}). $\emptyset_{ij}$ is a random number between [-1,1] which is used to adjust the old food source to become a new food source in the next iteration. i represents a particular employed bee. $k \in \{1,2,3..,N_p\} \wedge k \neq i$ and $j \in \{1,2,3..,D\}$ are randomly chosen indexes. The difference between x_{ij} and x_{kj} is a difference of position in one particular, randomly chosen dimension.

   If a new food source ($\tilde{x}_{ij}$) is better than an old food source (x_{ij}), the new food source will replace the old food source.

2. **Solution Sharing:** When the employed bees return to the hive, they share information with the onlooker bees in their own patriline about the quality of the candidate food sources they found. The onlooker bees then probabilistically select these food sources. Food sources of higher fitness have a larger chance of

being selected by onlooker bees than ones of lower fitness. The probability that a food source will be selected can be obtained from equation (6.3) below.

$$P_i = \frac{fit_i}{\sum_{n=1}^{N_p} fit_n} \tag{6.3}$$

where fit_i is the fitness value of the food source *i,* which is related to the objective function value ($F(x_i)$) of the food source *i*.

3. **Solution Updating by Onlooker Bees:** Onlooker bees in each patriline select food sources from their employed bees based on the probabilities calculated from step 2 and update those food sources for all dimensions d based on our Best-so-far method by using equation (6.4) below.

$$\tilde{x}_{id} = x_{id} + \Phi fit_b(x_{id} - x_{bd}) \tag{6.4}$$

where v_{id} = The new candidate food source for onlooker bee *i* dimension *d, d=1,2,3,...D*
x_{id} = The selected food source *i* in dimension *d*
Φ = A random number between -1 to 1
f_b = The fitness value of the best food source so far
x_{bd} = The best so far food source in dimension *d*

The old food source will be replaced by the new food source if the new food source has a better fitness value.

4. **Radius Searching by Scout Bee:** An employed bee in each patriline whose food source is rejected will change to be a scout bee and will search randomly for a new food source. A new food source will be generated by using equation (6.5) below.

$$\tilde{x}_{ij} = x_{ij} + \emptyset_{ij}\left[\omega_{max} - \frac{iteration}{MCN}(\omega_{max} - \omega_{min})\right]x_{ij} \tag{6.5}$$

where $\tilde{x}_{ij}$ is a new feasible food source of a scout bee that is modified from the abandoned food source (x_{ij}) and $\emptyset_{ij}$ is a random number between [-1,1]. *MCN* stands for the Maximum Cycle Number, the criterion for halting the search process. The parameters ω_{max} and ω_{min} represent a range used to adjust the scout bee's search radius. The values of ω_{max} and ω_{min} are fixed at 1 and 0.2 respectively. These parameters were empirically chosen by the experimenter (Banharnsakun et al., 2011). With these selected values, the amount of position change seen in the scout bee's new food source will linearly decrease from the

first to the final iteration. The employed bee that switches to be a scout will go back to being an employed bee in the next iteration.

5. **Solution Exchanging:** Each patriline periodically exchanges its local best food sources with its three neighboring patrilines. In this work, we perform the exchange process every 20 iterations. This value was empirically chosen by the experimenter (Banharnsakun et al., 2010b). Note that if the exchange period is too small, solutions may converge too quickly to local optima. On the other hand, if the period is too large, solutions may take a long time to converge to global optima.

Steps 1 to 5 will be repeated until the number of iteration equals to the Maximum Cycle Number or MCN (Karaboga and Basturk, 2007).

6. **Solution Gathering:** In the last iteration, the manager patriline gathers the local best food sources from all worker processes and calculates the global best food source.

In the algorithm described above, the foraging behavior of a patriline is influenced by both positive and negative feedback from its neighbors. Communication among patrilines contributes to the patrilines' diversity. These mechanisms suggest that the foraging behavior may be more adaptable to real world problems. Performance scalability of the distributed algorithm is described next.

6.2 Performance Scalability

This section aims to evaluate the performance of the proposed distributed algorithm in comparison to the sequential implementation of the best-so-far ABC in both the perspectives of result accuracy and algorithm's efficiency. We implemented our algorithm in C++ with the MPI library and executed it on a three-machine cluster running Windows XP. The processor on each machine was the Intel Core 2 Quad CPU Q6600 with a speed of 2.40 GHz and 2 GB of RAM. Thus, the entire cluster provides a total of 12 computing units (four cores times three machines). The numbers of employed and onlooker bees were set to 72. The number of patrilines used (denoted as "#p" in Table 6.1) ranged from a single patriline to 12 patrilines (matching the number

of available processors). Each experiment was repeated 30 times with different random seeds.

6.2.1 The Evaluation of Solution Accuracy

To evaluate the solution accuracy, the Best-so-far ABC with Multiple Patrilines algorithm was tested on six basic benchmark functions, which are described in Chapter 5. The dimension on each benchmark function was set to 30 except for the Schaffer function, where the dimension is set to 2. The MCN was set to100.

The mean and the standard deviation of the output values were recorded. The results obtained from the experiment sets are shown in Table 6.1. Note that lower values in both "Mean" and "SD" columns indicate better solutions.

Table 6.1 Solution quality obtained from Best-so-far ABC with multiple patrilines

Algorithm	Function Name					
	Sphere		Griewank		Rastrigin	
	Mean	SD	Mean	SD	Mean	SD
Best-so-far ABC – 1p	1.45E-35	8.21E-35	1.80E-37	9.88E-37	3.45E-35	1.88E-34
Best-so-far ABC – 2p	2.76E-42	1.51E-41	1.54E-51	8.42E-51	6.47E-48	3.54E-47
Best-so-far ABC – 4p	9.46E-43	5.18E-42	1.03E-52	5.65E-52	1.65E-49	9.02E-49
Best-so-far ABC – 8p	2.58 E-43	1.34 E-42	9.45E-50	5.17E-49	6.05E-50	2.46E-49
Best-so-far ABC – 12p	5.34E-42	2.06E-41	9.00E-47	3.23E-46	3.51E-44	9.73E-44
Algorithm	**Function Name**					
	Rosenbrock		**Ackley**		**Schaffer**	
	Mean	**SD**	**Mean**	**SD**	**Mean**	**SD**
Best-so-far ABC – 1p	7.45E-02	2.92E-01	7.11E-16	1.45E-15	3.24E-04	1.77E-03
Best-so-far ABC – 2p	5.20E-02	2.47E-01	2.37E-16	9.01E-16	1.62E-03	3.68E-03
Best-so-far ABC – 4p	8.44E-03	4.39E-02	4.74E-16	1.23E-15	3.56E-03	4.76E-03
Best-so-far ABC – 8p	8.63E-03	4.35E-02	4.74E-16	1.23E-15	4.86E-03	4.94E-03
Best-so-far ABC – 12p	7.72E-03	3.51E-02	5.92E-16	1.35E-15	4.86E-03	4.94E-03

From the results, we can see that the mean and the standard deviation of the output values obtained from our distributed Best-so-far ABC were in the same range as those obtained from sequential best-so-far ABC on all benchmark functions. In other words, using multiple patrilines can improve the execution time without scarifying the solution quality.

6.2.2 The Evaluation of Algorithm's Efficiency

In this section, the proposed algorithm's efficiency was evaluated on a large problem size and the parallel scalability analysis is discussed. To ensure that the workload in a single iteration is substantially high, we chose to use with Griewank as a benchmark

function because it has a large range of search space ($-600 \leq x_i \leq 600$). The dimension on this function was set to 10000, and the MCN was set to 1000. The most common measurements of parallel algorithm effectiveness are speedup and efficiency. These values can be calculated from equation (6.6) and (6.7) respectively.

$$\boldsymbol{Speedup} = \frac{\text{Execution time on one processor core}}{\text{Execution time on } m \text{ processor cores}} \tag{6.6}$$

$$\boldsymbol{Efficiency} = \frac{Speedup}{m} \tag{6.7}$$

The results of the execution time, speedup and efficiency for different number of processors are shown in Table 6.2 and the scalability plot of these results are illustrated in Figure 6.4 below.

Table 6.2 Speedup and efficiency for given number of processors

Number of processors	Execution Time (Sec)	Speedup	Efficiency
1	358.766	-	-
2	187.422	1.914215	0.957107
4	95.922	3.740185	0.935046
8	48.376	7.416198	0.927025
12	32.524	11.03081	0.919234

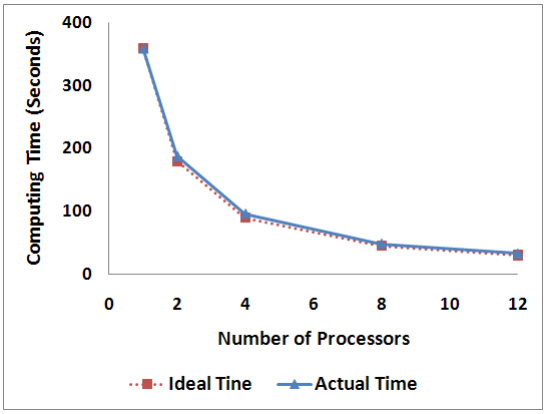


Figure 6.4 Scalability Plot

With a large problem size, our distributed Best–so-far approach performed very close to ideal time. The execution times dropped almost linearly as more processors were added to the computation, while the solution quality was maintained. The computation to communication ratio was also high. In other words, low communication overhead was observed. The efficiency could thus be measured at as high as 90% in all cases. The smallest and the biggest performances dropped from the ideal are 4% and 9 % when the number of processors used are set to 2 and 12 respectively. It can be concluded that by

adding more processors, a very large problem size can be solved in a relatively short amount of time using a cluster of state of the art machines.

To summarize, our Best-so-far ABC with Multiple Patrilines algorithm is scalable and produces good solutions while significantly improving the processing time. The next Chapter, the adoption of the distributed algorithm will be presented in details in the context of clustering application. Performance analysis will also be provided.

CHAPTER 7 APPLYING BEST-SO-FAR ABC TO PRACTICAL APPLICATIONS

In order to confirm the algorithm performance when used with the real world problems, we have implemented best-so-far ABC with several application domains including image registration and job shop scheduling. In addition, to demonstrate the performance of the distributed version of our proposed algorithm, we also apply the distributed Best-so-far ABC algorithm to clustering application. In this chapter, each section starts with a brief background on the application domain, then the adoption of the best-so-far model to these problems, and finally, the experiments and results.

The scalability and speed up of the distributed version can only be tested with large data sets. In this chapter, the scalability result is thus only presented with the clustering problem with data from actual manufacturing process, where the number of data records and parameters are large sufficiently.

7.1 Image Registration

7.1.1 Background

Image registration is a process used to align two or more images of a scene. It plays an important role in many research fields such as the computer vision, remote sensing and medical imaging (Chen, et al., 2003; Jacquet, et al., 2009). Usually a target image must be transformed to be geometrically coincident with a reference image. To do this, corresponding features must be identified in the two images or else the global intensity distributions must be compared. A similarity or fitness measure (Goshtasby, 2005) is also necessary. The registration process searches for a transformation from the target to the reference feature geometry that maximizes this similarity measure.

The global optimization of the similarity measurement is the key to automate the registration process. There are two common approaches to measure the similarity, which are geometric distance and statistical similarity. Hessian and Hausdorff geometric distance have been used to measure the distance in computer vision researches (Mount, et al., 1999). These methods measure the nearest neighbor of rank k^{th}. The weakness of the methods is the fact that the presence of outliers may

affect the accuracy. Mismatch (Ratanasanya, et al., 2008) is then introduced. It is a new measure that makes use of a Gaussian distribution as a weight function in order to be tolerant of outliers.

Least Square Error is the method that statistically determines the smallest deviation or distance between two data sets, such as the reference and target images. The drawback of this measure is its sensitivity to the presence of outliers. Cross Correlation measures similarity of the two images by convolving them. Unfortunately, this measure is also not robust in the presence of noise or outliers. Currently, most of the researchers in this field work with Mutual Information (MI) as a measure of similarity. It is quite similar to correlation except that the concept of MI stems from conditional probability. It is a measure of the presence of the information from one set in another set. It is now becoming a standard to measure the similarity in image registration.

In this study, we used a statistical similarity approach where the Mutual Information (Cover and Thomas, 2006; Maes, et al., 1997; Viola and Wells, 1997) has been used for measuring the similarity of the two images. An optimization solution in the mutual information approach can be expressed mathematically as follows:

$$\alpha^* = \arg\max_{\alpha}(MI(I_R, I_T)) \tag{7.1}$$

Where α is the set of transformation parameters, *MI* is an objective function, I_R is a reference image and I_T is a target image.

The mutual information of I_R and I_T can be expressed in terms of Shannon's entropy as follows:

$$MI(I_R, I_T) = H(I_R) + H(I_T) - H(I_R, I_T) \tag{7.2}$$

Where $H(I_R)$ and $H(I_T)$ are the Shannon's entropies, and $H(I_R, I_T)$ is the joint entropy of the two images.

Shannon's entropy can be defined as:

$$H(I_R) = -\sum_{a} p_R(a)\log\left(p_R(a)\right) \tag{7.3}$$

$$H(I_T) = -\sum_{b} p_T(b)\log\left(p_T(b)\right) \tag{7.4}$$

$$H(I_R, I_T) = -\sum_{a}\sum_{b} p_{R,T}(a,b)\log\left(p_{R,T}(a,b)\right) \tag{7.5}$$

Where $p_R(a)$ and $p_T(b)$ are the marginal probability mass functions of the reference and the target image respectively, and $p_{R,T}(a,b)$ is the joint probability mass function of two images.

The joint histogram of the image pair can be used to calculate these marginal probability mass functions:

$$p_{R,T}(a,b) = \frac{h(a,b)}{\sum_a \sum_b h(a,b)} \tag{7.6}$$

$$p_R(a) = \sum_b p_{R,T}(a,b) \tag{7.7}$$

$$p_R(b) = \sum_a p_{R,T}(a,b) \tag{7.8}$$

Where $h(a,b)$ is a number of corresponding pairs having intensity value *a* in the reference image and intensity value *b* in the target image.

Various types of transformations can be applied in image registration. In our work, we used rigid transformations involving rotation and translation along the x-axis and y-axis. The transformation of images can be written as:

$$x' = x_c + (x - x_c) * cos\theta + (y - y_c) * sin\theta + \Delta x \tag{7.9}$$

$$y' = y_c - (x - x_c) * sin\theta + (y - y_c) * cos\theta + \Delta y \tag{7.10}$$

Where $x'$and y' are new coordinates, x and y are the current coordinates, x_c and y_c are the center coordinates of the target image, Δx and Δy are the displacements of a target image, and θ is the rotation angle of the target image around its center.

7.1.2 Optimization Solution with Best-so-far ABC

We applied the best-so-far ABC method to the registration problem described above. The goal was to find a global optimization of the similarity measure. In other words, we tried to find the transformation variables that can maximize the mutual information values of the images. The adopted algorithm is illustrated in Figure 7.1.

In Figure 7.1, initial solutions consisting of a rotation, and x and y-axis translation values are generated. These parameter sets are treated as the food sources for the employed bees. Each solution is used to transform the target image by re-sampling. Then we calculate the mutual information score between the transformed target and the

reference image. The onlooker bees will then select the solutions that produce higher mutual information and update those solutions based on our best-so-far method. The process will be repeated until the mutual information reaches a threshold value or the number of iteration equals the MCN. Solutions that cannot improve the mutual information within a certain period will be abandoned and new solutions will be regenerated by the scout bee.

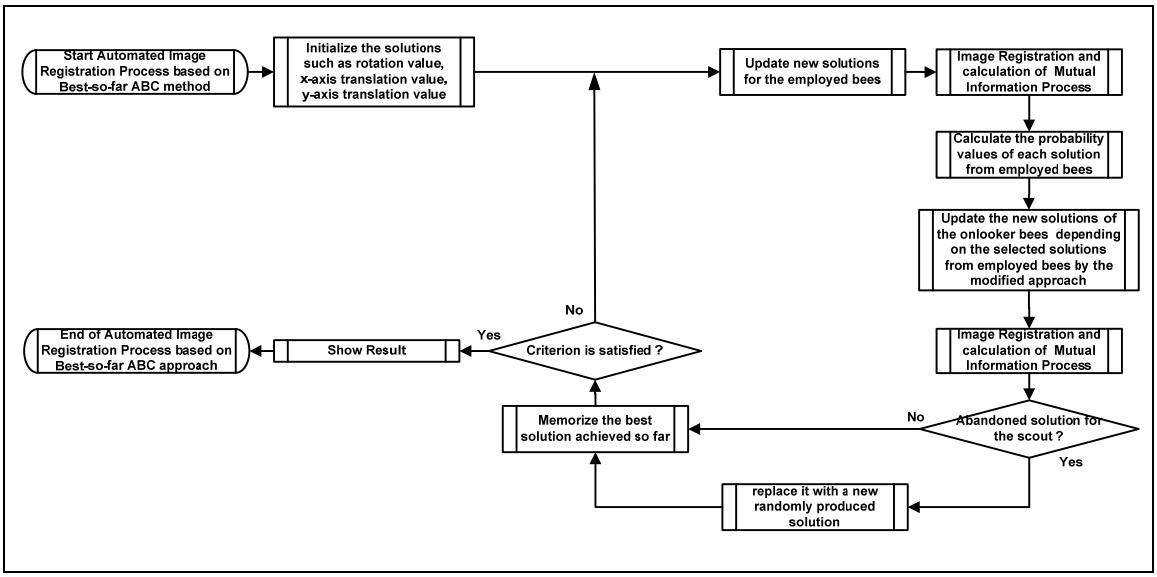


Figure 7.1 Automated Image Registration Based on The Best-so-far ABC method

Equation (2.2) is used by employed bees in order to update variables and improve the solution quality in each iteration. However, due to the independence among the transformation parameters in image registration application, equation (4.1), which uses a randomly chosen dimension, is slightly adjusted. A fixed dimension is used instead, as shown in equation (7.11).

$$v_{id} = x_{id} + \Phi f_b(x_{id} - x_{bd}) \tag{7.11}$$

In our algorithm, equation (7.11) is introduced to improve the solutions in each iteration. When the solutions cannot be further improved, equation (2.2) is used to calculate the new candidate solutions. Moreover, equation (4.2) is used by scout bees to randomly generate the new solutions when the mutual information cannot be improved.

7.1.3 Image Registration Application Experiment Methodology

In our image registration experiments, once again we are to compare both the solution quality and execution performance between our best-so-far and original ABC implementation. The objective function of the image registration application is *Mutual Information* or MI. MI is used to measure the similarity of the two input images after registration. Thus, the higher the MI value, the more accurate the registration process.

In our experiments, we used four sample image pairs. For each pair, the experiments were repeated 30 times. The number of employed and onlooker bees used were 50 and the maximum number of iterations was 200. The search space of each variable for rigid transformation process in terms of rotation degree (θ), translation in x-axis (x) and y-axis (y) were defined as shown in Table 7.1 below. The ranges of these parameters were chosen based on a visual estimate of the maximum amount of change required.

Table 7.1 The search space of each variable for rigid transformation process

Image Pair	(θ)	(x)	(y)
I	$-25^o \leq \theta \leq 25^o$	$-20 \leq x \leq 20$	$-20 \leq y \leq 20$
II	$0^o \leq \theta \leq 50^o$	$-210 \leq x \leq 210$	$-160 \leq y \leq 160$
III	$-1^o \leq \theta \leq 1^o$	$-10 \leq x \leq 10$	$-50 \leq y \leq 0$
IV	$-90^o \leq \theta \leq 90^o$	$-256 \leq x \leq 256$	$-256 \leq y \leq 256$

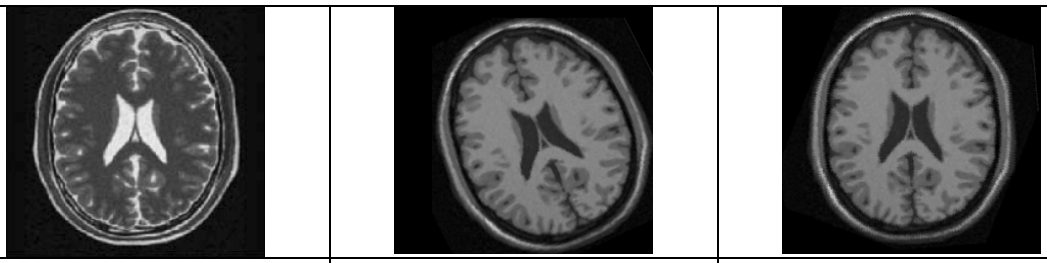
We tested the algorithms with two cases. In the first case, we used a controlled initial solution with the fixed initial values of θ, x, and y. We offset all the initial values from the lower bound displayed in the Table above. However, the initial values might accidentally be fixed close to the optimum solution. Thus, in the second test case, we used randomly generated initial values.

In both test cases, we investigated the solution quality of the two algorithms when the runtime was approximately the same as well as studying the algorithm performance when the solution quality was approximately the same. In other words, we would like to know which algorithm uses less computation time to create a solution of the equivalent quality.

7.1.4 Image Registration Results and Discussion

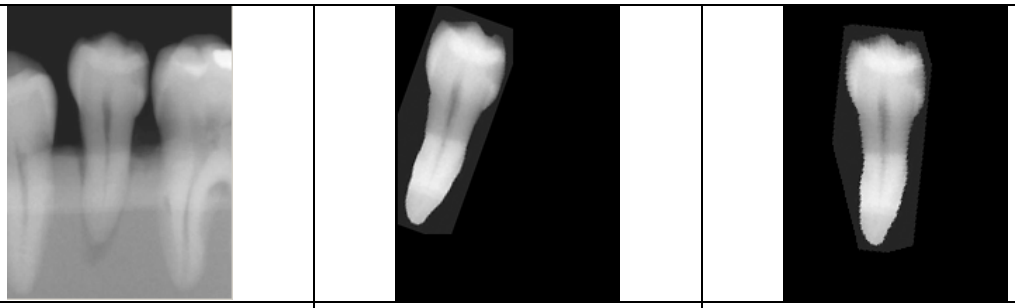
In this experiment, we used both the original and the best-so-far methods to register four sets of images. The two methods produced results that appeared indistinguishable to human eyes. However, the analysis of MI for registered images suggests that the best-so-far method offered advantages.

The registered images are shown in Figure 7.2 to Figure 7.5. However, if we investigate the resulting numbers closely, the best-so-far produces better results faster as discussed next.



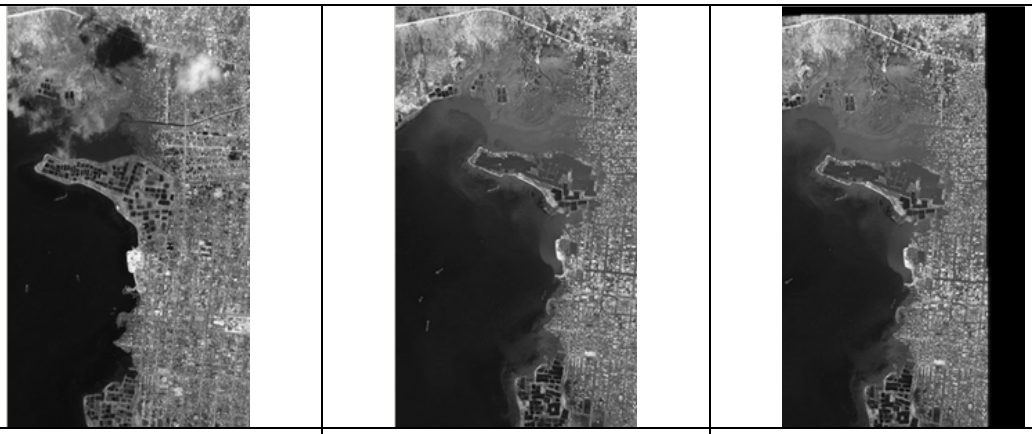
Reference Image	Target Image	Resampling of the Target Image

Figure 7.2 Image Pair I: Brain image for registration



Reference Image	Target Image	Resampling of the Target Image

Figure 7.3 Image Pair II: Dentistry image for registration



Reference Image	Target Image	Resampling of the Target Image

Figure 7.4 Image Pair III: Satellite image for registration

Reference Image

Target Image

Resampling of the Target Image

Figure 7.5 Image Pair IV: Camera Man image for registration

First, we discuss the results of the experiments for the fixed initial solution. Table 7.2 and Figure 7.6 show the results in a case where the runtime of both methods is approximately the same. In Table 7.3, the means of MI values are fixed to be approximately the same.

Table 7.2 The results obtained from the original ABC and best-so-far ABC method based on fixed initial solutions at approximately the same runtime value.

Image Pair	Image size (H x W)	ABC				Best-so-far ABC			
		Convergence Iteration	Runtime (s)	Mean (MI)	SD (MI)	Convergence Iteration	Runtime (s)	Mean (MI)	SD (MI)
I	253x255	186	687.699	1.25875	9.55E-05	179	684.106	1.25883	8.46E-05
II	210x160	199	413.281	0.552284	3.31E-02	200	411.116	0.581292	1.76E-02
III	299x176	200	636.353	0.957611	9.28E-04	196	635.344	0.959218	2.37E-04
IV	256x256	200	723.533	1.3449	3.00E-02	188	719.947	1.36046	7.57E-05

Table 7.3 The results obtained from the original ABC and best-so-far ABC method based on fixed initial solutions at approximately the same mean MI value.

Image Pair	Image size (H x W)	ABC				Best-so-far ABC			
		Convergence Iteration	Runtime (s)	Mean (MI)	SD (MI)	Convergence Iteration	Runtime (s)	Mean (MI)	SD (MI)
I	253x255	186	687.699	1.25875	9.55E-05	108	412.756	1.25875	8.53E-05
II	210x160	199	413.281	0.552284	3.31E-02	32	65.778	0.553137	1.81E-02
III	299x176	200	636.353	0.957611	9.28E-04	44	142.628	0.957646	3.14E-04
IV	256x256	200	723.533	1.3449	3.00E-02	37	141.691	1.34501	7.66E-05

In Table 7.2, we compare the results using approximately the same runtime value. The best-so-far method gives slightly better solutions than the original algorithm in all cases of sample image pairs, with somewhat faster convergence. Table 7.3 shows the convergence speed at approximately the same mean value of MI. The results indicate that the best-so-far method solutions are converged to an optimal solution more quickly than the original method in all image pairs. The maximum runtime improvement of 84% was found in the experiment with image pair II and the minimum of 39% was with image pair I. The average runtime improvement for all image pairs was 70%.

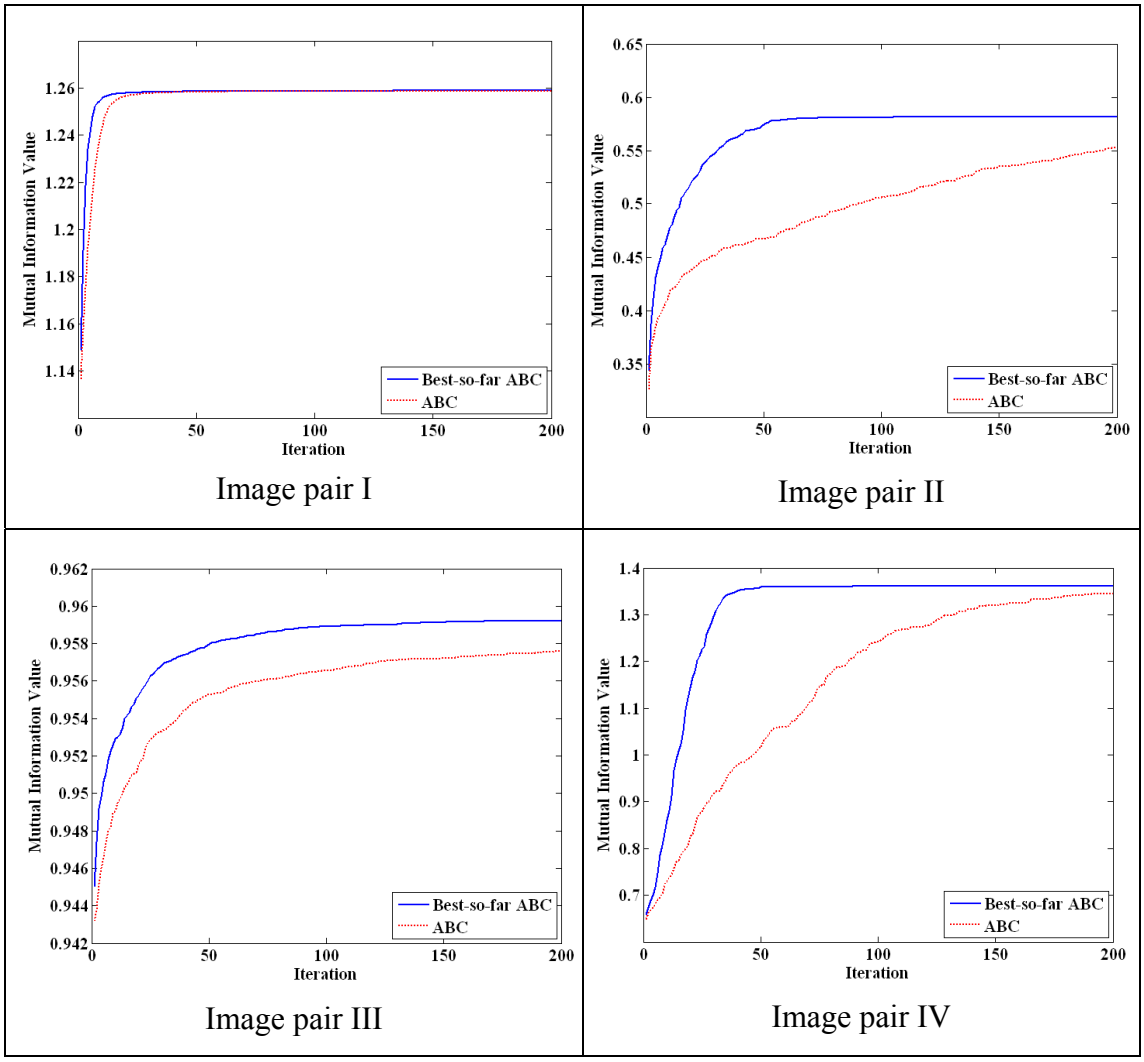


Figure 7.6 The mean of the mutual information across iterations in Image Pair I-IV based on fixed initial solutions

For the random initial solution experiments, in some cases, if the initial solution is close to the optimal solution, the results from the first few iterations of the original algorithm generate better results. However, when the number of iterations increases, the best-so-far method will adjust the solutions and converge to the optimal solutions more quickly. The results of these experiments are shown in Tables 7.4 and 7.5 and Figure 7.7.

Table 7.4 The results obtained from the original ABC and best-so-far ABC method based on random initial solutions by using approximately the same runtime value.

Image Pair	Image size (H x W)	ABC				Best-so-far ABC			
		Convergence Iteration	Runtime (s)	Mean MI	SD MI	Convergence Iteration	Runtime (s)	Mean MI	SD MI
I	253x255	189	714.005	1.25867	9.53E-05	196	712.789	1.25878	1.13E-04
II	210x160	199	400.781	0.580997	2.89E-03	187	399.604	0.58439	3.26E-04
III	299x176	199	641.630	0.958047	8.09E-04	200	629.072	0.959175	6.32E-04
IV	256x256	199	759.200	1.35800	5.38E-03	200	758.391	1.36049	4.54E-05

Table 7.5 The results obtained from the original ABC and best-so-far ABC method based on random initial solutions by using approximately the same mean MI value.

Image Pair	Image size (H x W)	ABC				Best-so-far ABC			
		Convergence Iteration	Runtime (s)	Mean MI	SD MI	Convergence Iteration	Runtime (s)	Mean MI	SD MI
I	253x255	189	714.005	1.25867	9.53E-05	82	298.207	1.25867	1.17E-04
II	210x160	199	400.781	0.580997	2.89E-03	65	138.899	0.581029	3.44E-04
III	299x176	199	641.630	0.958047	8.09E-04	38	119.524	0.958108	6.56E-04
IV	256x256	199	759.200	1.35800	5.38E-03	59	223.725	1.35838	4.62E-05

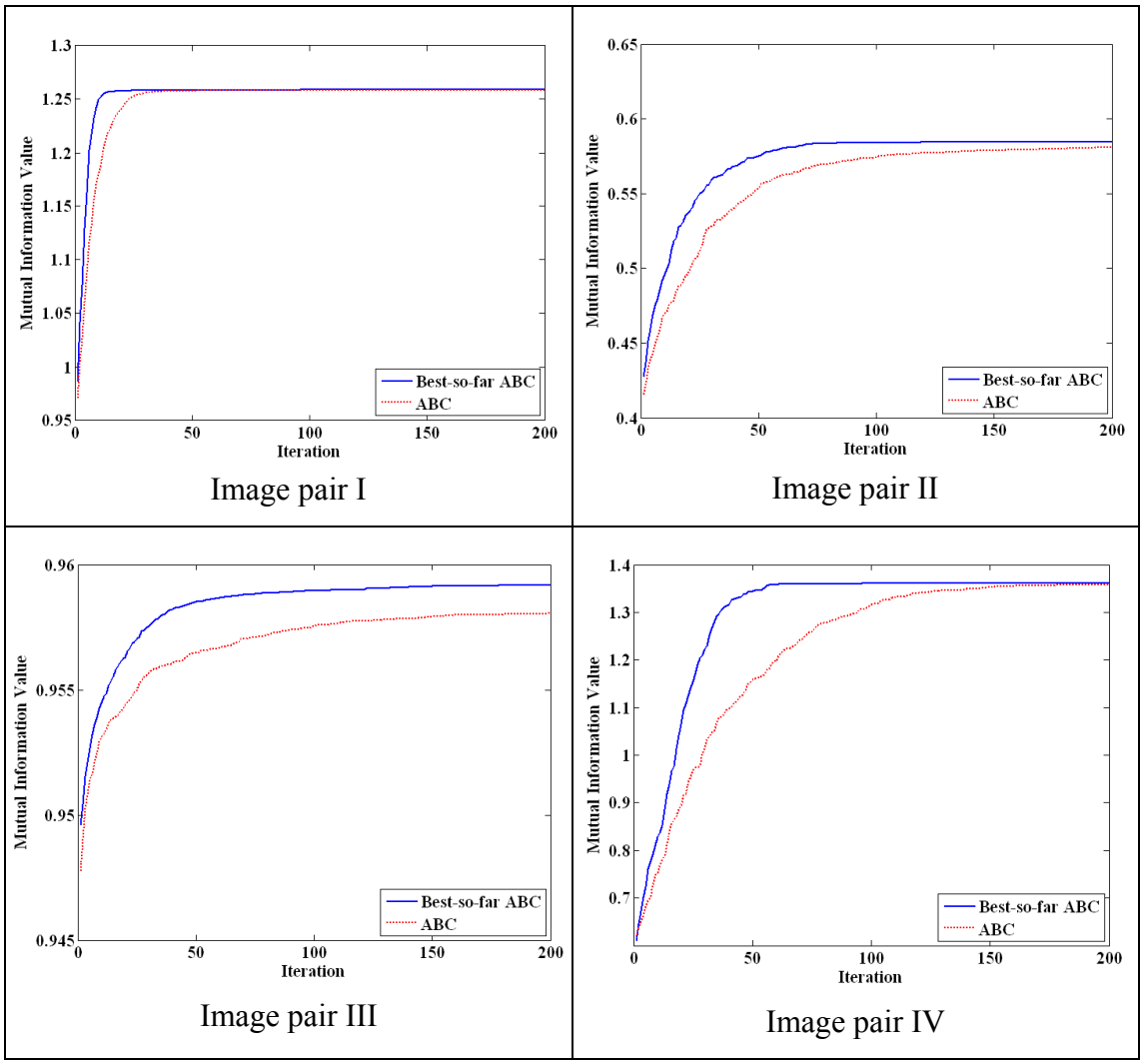


Figure 7.7 The mean of the mutual information across iterations in Image Pair I-IV based on random initial solutions

In the random initial solutions experiment, although the MI values from the original method are equivalent to the MI values from the best-so-far method at approximately the same runtime value as shown in Table 7.4, the results in Table 7.5 show that the solutions in the same case are still converged to an optimal solution faster than the original algorithm when the approximate mean value was the same. The maximum and

minimum percentage of the runtime improvement, found on the experiment of image pair III and image pair I, were equal to 81% and 58% respectively. The average improvement on the runtime for the best-so-far method compared with the original model on all experiments of image pairs in this case was 68%.

7.2 Job Shop Scheduling

7.2.1 Background

The Job Shop Scheduling Problem (JSSP) is a real-world problem in a field of production management. To survive in the modern competitive marketplace, which requires lower cost and shorter product life cycles, a corporation must respond quickly and precisely to the customer's demands. Effective scheduling plays an important role in this adaptation.

French (1982) described the Job shop scheduling problem (JSSP) as a set of n jobs denoted by J_j where $j = 1,2,\ldots, n$ which have to be processed on a set of m machines denoted by M_k where $k = 1,2,\ldots,m$. Operation of j^{th} job on the k^{th} machine will be denoted by O_{jk} with the processing time P_{jk}. Each job must be processed through all machines in a particular order also known as the technological constraint and processing time varies with each job. A machine can process only one job at a time. The required order of machines also varies from one job to another. Once a machine starts to process a job, no interruption is allowed. The time required for all operations to complete their processes is called makespan. The objective of JSSP aims to minimize the makespan value. A solution can be expressed by equation (7.12) below.

$$\text{Min } C_{max} = \max (C_1, C_2, C_3, \ldots C_n) \tag{7.12}$$

$$C_j = \sum_{k=1}^{n} (w_{jk} + p_{jm(k)})$$

where

C_j is the completion time of job j

w_{jk} is the waiting time of job j at sequence k

$p_{jm(k)}$ is the processing time needed by job j on machine m at sequence k

The difficulty in finding good solutions to JSSP is due to the number of feasible solutions. For large problem size, the number of possible solutions makes it nearly impossible to explore the entire solution space. The total number of all possible solutions is $(n!)^m$.

Many approaches using both mathematical formulations and heuristic methods have been developed to solve this problem. For a JSSP situation of small size, mathematical formulations such as integer programming techniques (Fisher, 1981) can be used to solve this problem in reasonable computational time. However, Garey et al. (1976) provided a proof that this problem is NP-complete, that is, as the problem size increases, the computational time to find the best schedule using the mathematical formulation methods grows exponentially. To handle this issue, heuristic methods such as branch-and-bound (Carlier and Pinson, 1989) have been considered.

Metaheuristics (Yang, 2008) are one of many approximation methods widely used to solve practical optimization problems. In recent years, several algorithms employing a metaheuristic approach such as Genetic Algorithm (GA), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), Bee Colony Optimization (BCO), and Artificial Bee Colony (ABC) have been applied to solve JSSP.

Watanabe et al. (2005) introduced a genetic algorithm (GA) with a modified crossover operator and a search area adaptation method for controlling the tradeoff balance between global and local searches. Goncalves et al. (2005) presented a hybrid method called Hybrid Genetic Algorithm (HGA). The method combined GA and local search based on a disjunctive graph model and a neighborhood approach to improve the solution. To enhance the performance of GA, Asadzadeh and Zamanifar (2010) proposed an agent-based parallel GA approach. This parallel approach is based on a coarse-grained model. The initial population is divided into sub-populations, and each sub-population is evolved separately. Communication between sub-populations is restricted to the migration of chromosomes. Heinonen and Pettersson (2007) developed a hybrid approach based on an Ant Colony Optimization algorithm (ACO) and a post-processing algorithm to enhance the ACO performance for solving the JSSP.

To improve the solution quality in JSSP, Tasgetiren et al. (2006) presented a hybrid method (PSO-VNS) based on the Particle Swarm Optimization (PSO) and the Variable Neighboring Search (VNS). To further improve efficiency of PSO, a new hybrid swarm

intelligence algorithm (MPSO) consisting of particle swarm optimization, Simulated Annealing (SA) and a multi-type individual enhancement scheme was developed by Lin et al. (2010). Ge et al. (2007) employed a high global search efficiency of PSO with a powerful ability to avoid being trapped in local minima of SA by introducing an algorithm called Hybrid Evolutionary Algorithm (HEA). Ge et al. (2008) exploited the capabilities of distributed and parallel computing in swarm intelligence approaches by proposing a computationally efficient algorithm for combining PSO with an Artificial Immune System (AIS) to find the minimum makespan for job-shop scheduling.

Most of these aforementioned approaches aim to improve their algorithm's performance by introducing a hybrid method. They combined the advantageous features of each algorithm to improve both the local search and global search capability of their algorithms.

Inspired by the decision making capability of bee swarms, Chong and Low (2006) explored an evolutionary computation based on Bee Colony Optimization (BCO) to solve JSSP. The scheduling construction in this approach was done based on a state transition rule. Yao et al. (2010) presented an Improved Artificial Bee Colony algorithm (IABC) to enhance the search performance of the Artificial Bee Colony algorithm (ABC) for solving the JSSP. The mutation operation was also utilized in this method for exploring the search space and avoiding local optima.

In this section, we propose a modified version of the ABC algorithm called Best-so-far ABC. The research aims to improve the solution quality, which is measured based on "Best", "Average", "Standard Deviation (S.D.)", and "Relative Percent Error (RPE)" of the objective value. The algorithm is presented and applied to solve the JSSP.

7.2.2 Mapping Best-so-far ABC Algorithm to the JSSP

We applied the Best-so-far ABC method to JSSP described above. The goal is to find a global optimization of the makespan, i.e. we try to find the job operation scheduling list that minimizes the makespan valcue. The adopted algorithm is illustrated in Figure 7.8.

The steps of operation can be described as follows:

The First Step

The initial parameters such as the number of employed bees, onlooker bees and maximum cycle number (MCN) are set. Next, the job's processing time on each machine and the job's machine sequence will be given at this step. Examples can be found in Table 7.6 and Table 7.7 respectively.

In our solution representation, a solution in JSSP is an operation scheduling list which is represented as a food source (x) in our Best-so-far ABC algorithm. Each dimension in a food source represents one operation of a job. Each job appears exactly m times in an operation scheduling list. For the n-job and m-machine problem, each food source contains $n \times m$ dimensions corresponding to $n \times m$ operations. This representation is illustrated in Figure 7.9.

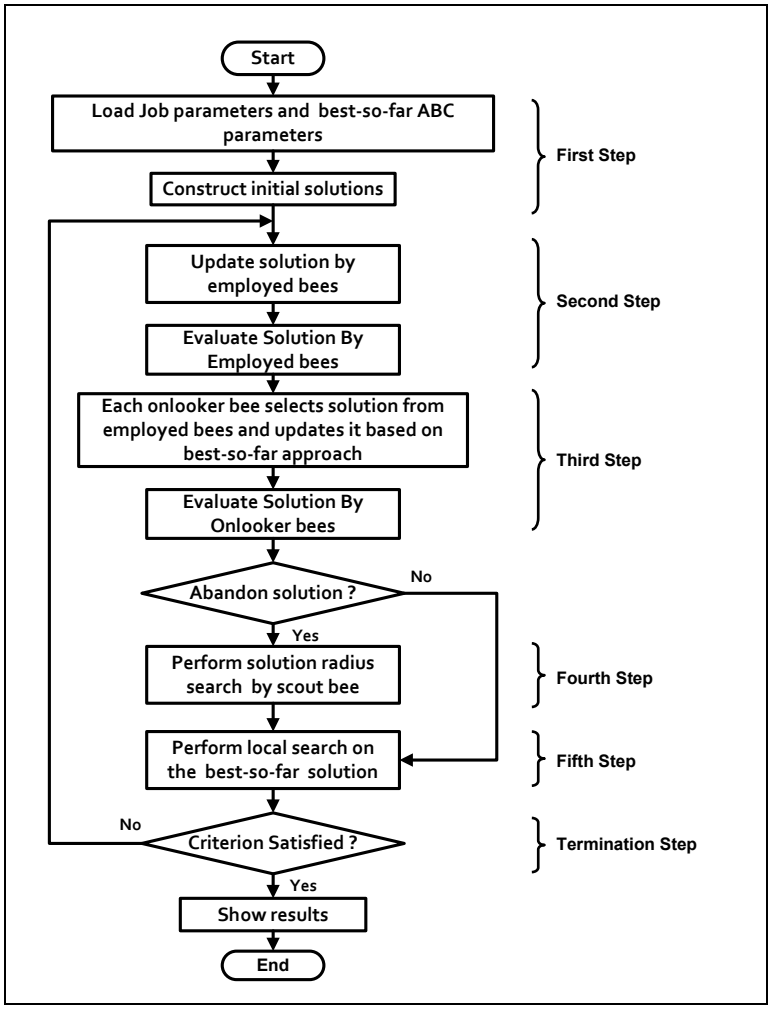


Figure 7.8 The Best-so-far ABC algorithm flowchart for JSSP

Table 7.6 An example of job processing time on each machine for 3-job x 3-machine

	M_1	M_2	M_3
J_1	15	25	18
J_2	7	10	20
J_3	12	10	8

Table 7.7 An example of job machine sequence for 3-job x 3-machine

	O_1	O_2	O_3
J_1	M_1	M_2	M_3
J_2	M_2	M_3	M_1
J_3	M_3	M_1	M_2

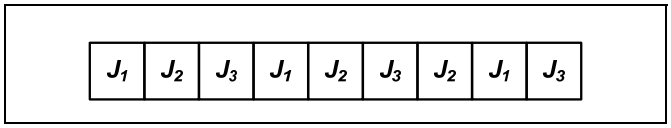


Figure 7.9 The example of operation scheduling list representation for 3-job x 3-machine

For Figure 7.9, J_i stands for the operation of job *i*. Since each job has three operations, it occurs three times in the operation scheduling list.

The interpretation of the example above is as follows. As we scan the operation scheduling list in Figure 7.9 from left to right with following constrained problems in Table 7.6 and Table 7.7, the first J_1 corresponds to the first operation of job J_1 which will be processed on machine M_1, the first J_2 corresponds to the first operation of job J_2 which will be processed on M_2, the first J_3 corresponds to the first operation of job J_3 which will be processed on M_3, the second J_1 corresponds to the second operation of job J_1 which will be processed on machine M_2, the second J_2 corresponds to the second operation of job J_2 which will be processed on M_3, the second J_3 corresponds to the second operation of job J_3 which will be processed on M_1, the third J_2 corresponds to the third operation of job J_2 which will be processed on machine M_1, the third J_1 corresponds to the third operation of job J_1 which will be processed on M_3, finally, the third J_3 corresponds to the third operation of job J_3 which will be processed on M_2. Thus, the feasible schedule can be constructed as shown in Figure 7.10.

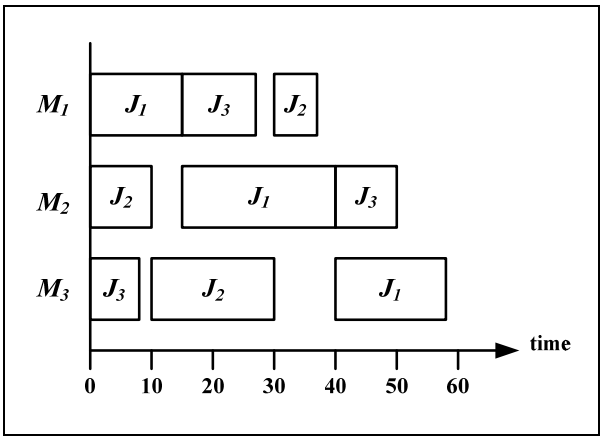


Figure 7.10 The feasible schedule constructed from the operation scheduling list in Figure 7.9

The fitness of each food source $f(x)$ is determined by the inverse of its makespan value ($C_{max}(x)$) which is calculated during the selection of feasible solutions. This mapping is shown in Figure 7.11.

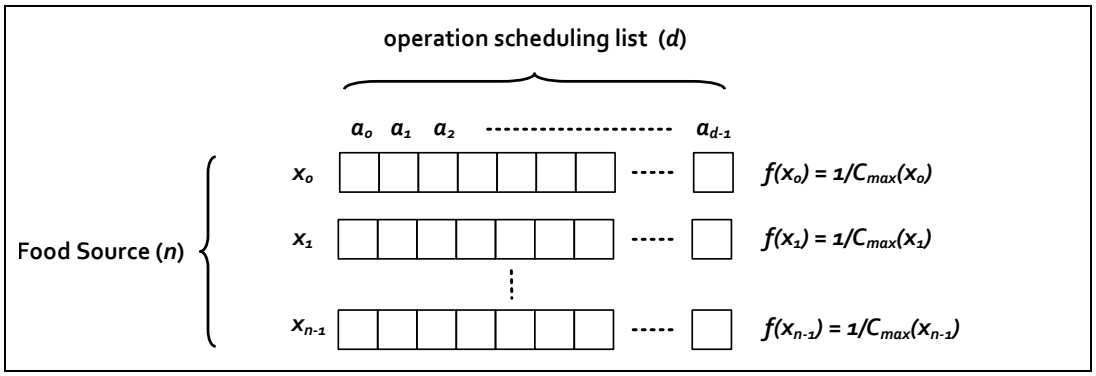


Figure 7.11 The mapping between the food sources and operation scheduling list

The Second Step

We employ the concept of the updating of a candidate food source based on a neighboring food source using equation (7.13) below.

$$v_{ij} = x_{ij} + \emptyset_{ij}(x_{ij} - x_{kj}) \tag{7.13}$$

The candidate food sources are updated by employed bees. Position Base Crossover (PBX) (Kellegoz and Toklu, 2008) is the mechanism used to update the employed bees' old food sources (x_{ij}) to new food sources (v_{ij}) based on their neighboring food sources (x_{kj}) randomly taken from other employed bees. An example of PBX is shown in Figure 7.12.

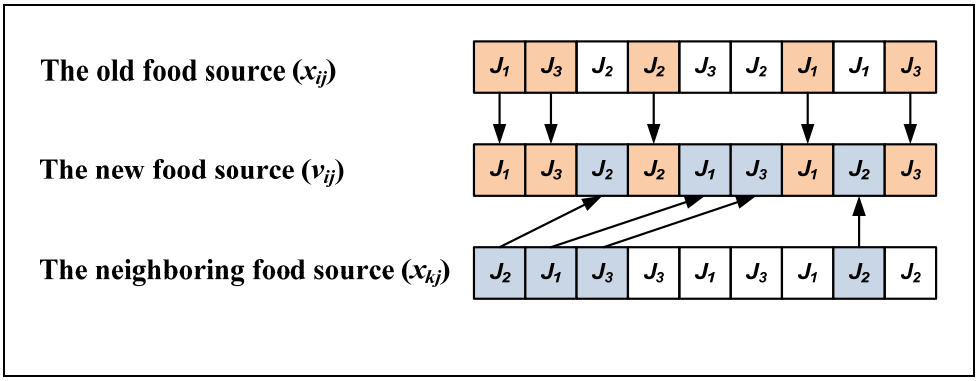


Figure 7.12 Exchanging information with a neighboring food source based on PBX method

Based on the PBX method, a set of job operations from the old food source is selected randomly. Each dimension in the operation scheduling list of the old food source is selected to produce the new position with the probability of 0.5. This probability is empirically chosen by the experimenter to allow the new food source to be generated from both the old food source and the neighboring food source in the same proportion. The jobs already selected from the old food source are ignored in being selected from the neighboring food source. The job operations on the neighboring food source that are not selected from the old food source will be selected and placed into empty positions from the left to the right of the operation scheduling list in the new food source. To guarantee that each job in the new food source is still included exactly *m* times, if any job already selected *m* times from either the old food source or the neighboring food source, we skip it and look at the next job.

The old food source (x_{ij}) in the employed bee's memory will be replaced by the new candidate food source (v_{ij}) if the new position has a better fitness value.

The Third Step

The employed bees share new solutions (v_{ij}) that they have found with the onlooker bees, who then select these solutions based on probability (P_i) calculated from an equation below.

$$P_i = \frac{f(v_i)}{\sum_{n=1}^{SN} f(v_i)} \tag{7.14}$$

where $f(v_i)$ is the fitness value of the food source i and SN is the number of food sources.

After onlooker bees have selected the solutions *(x_{ij})* from employed bees, then the best solution (x_{bj}) is identified from those selected solutions. The best solution (x_{bj}) will replace the best-so-far solution (x_{bj} of the previous iteration) if the new best solution is better and its fitness value (f_b) will be used for guiding the direction of the search space as shown in equation below.

$$v_{id} = x_{ij} + \Phi f_b(x_{ij} - x_{bj}) \tag{7.15}$$

The PBX method is also used to update the old solution (x_{ij}) of the onlooker bees to the new solution (v_{ij}) based on the best-so-far food source (x_{bj}) instead of the neighboring food sources (x_{kj}).

The old food source (x_{ij}) in the onlooker bee's memory will be replaced by the new candidate food source (v_{ij}) if the new position has a better fitness value.

The Fourth Step

To avoid suboptimal solutions, the scout bees ignore the old solution and randomly search for new solutions by using the concept of the equation below.

$$v_{ij} = x_{ij} + \emptyset_{ij}\left[\omega_{max} - \frac{iteration}{MCN}(\omega_{max} - \omega_{min})\right]x_{ij} \tag{7.16}$$

Based on this concept, the operation scheduling lists (x_{ij}) whose fitness values have not been improved after a certain period are abandoned and replaced by new operation scheduling lists (v_{ij}) updated by the scout bees by using the radius search method as shown in Figure 7.13.

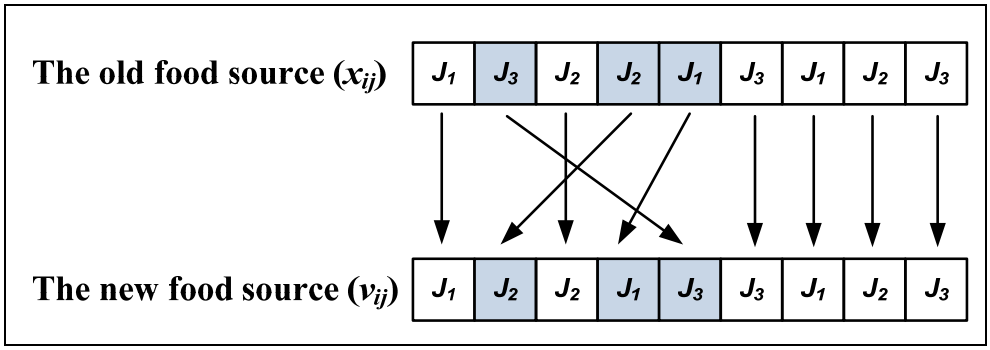


Figure 7.13 The updating food source based on radius search by scout bee to find v_{ij}

In the radius search method, a set of job operations from the old food source is selected randomly. Each dimension in the operation scheduling list is selected with the probability of ω_{max} in the first iteration and will linearly decrease to ω_{min} in the last

iteration. The value of ω_{max} and ω_{min} are fixed to 1 and 0.2 respectively. These parameters were chosen by the experimenter (Banharnsakun et al., 2011). So the first iteration will replace every dimension, that is, will swap the randomly selected solution for the one being abandoned.

In this swapping, the dimensions that are not marked as selected dimension from the old food source will be placed into the same dimension in the new food source. Next, the dimensions marked as selected dimension from the old food source will be placed into the empty positions from the left to right of the operation scheduling list in the new food source.

The Fifth Step

A local search based on the Variable Neighboring Search method (VNS) (Hansen et al., 2008) is performed on the best-so-far solution (x_b of eqation 7.15) to improve the solution quality. The pseudo code of VNS method is shown in Figure 7.14.

Although it seems that VNS would actually find the best solution by itself, it sometimes takes a long time to reach useful solutions whilst solving large scale job shop scheduling. To overcome this shortcoming, our Best-so-far ABC helps VNS to find solutions more quickly by substantially narrowing down the search space.

Procedure VNS_Procedure
    Get Initial Solution, $x' = x_b$
    Set *step* = 0 and p = 1
    $n = number\ of\ jobs$
    $m = number\ of\ machines$
    $\alpha = random\_integer\_number[1, nm]$
    $\beta = random\_integer\_number[1, nm]\ , \beta \neq \alpha$
    $x' = Exchanging\_Process(x', \alpha, \beta)$
    $\alpha = random\_integer\_number[1, nm]$
    $\beta = random\_integer\_number[1, nm], \beta \neq \alpha$
    $x' = Inserting\_Process(x', \alpha, \beta)$
    $\alpha = random\_integer\_number[1, nm]$
    $\beta = random\_integer\_number[1, nm]\ , \beta \neq \alpha$
    $x' = Exchanging\_Process(x', \alpha, \beta)$
    While $(step \ \leq nm * (nm - 1))$
        $\alpha = random\_integer\_number[1, nm]$
        $\beta = random\_integer\_number[1, nm], \beta \neq \alpha$
        If (p = 1) **then** $x'' = Exchanging\_Process(x', \alpha, \beta)$
        Else if (p = 0) **then** $x'' = Inserting\_Process(x', \alpha, \beta)$
        If $(fitness(x'') \geq\ fitness(x'))$ then $x' = x''$
        Else $p = |p - 1|$
        $step = step + 1$
    End while
    If $(fitness(x') \geq\ fitness(x_b))$ then $x_b = x'$
End-Procedure

Figure 7.14 Variable Neighboring Search (VNS) Procedure

From Figure 7.14, let α and β are the random integer numbers between 1 and nm, $Exchanging\_Process(x', \alpha, \beta)$ means exchanging the job operations in solution x' between α^{th} and β^{th} dimensions, $\alpha \neq \beta$. $Inserting\_Process(x', \alpha, \beta)$ means removing the job operation in solution x' from the α^{th} dimension and inserting it in the β^{th} dimension.

The example of the exchanging process and the inserting process are shown in Figure 7.15 and Figure 7.16 respectively.

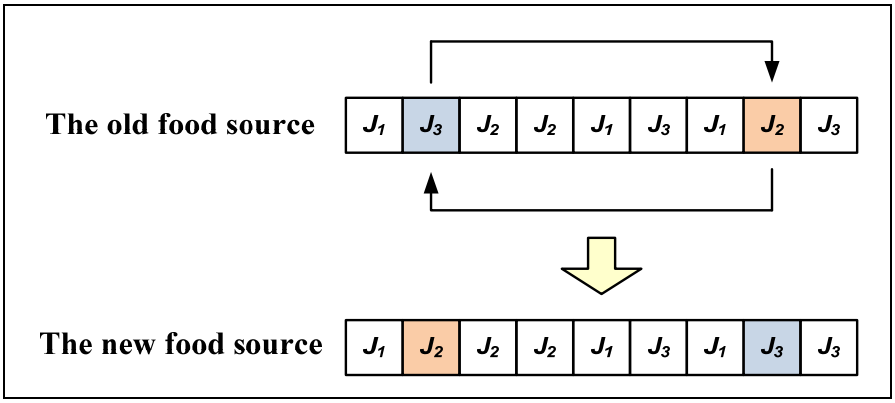


Figure 7.15 The exchanging process in VNS method (for new x_b)

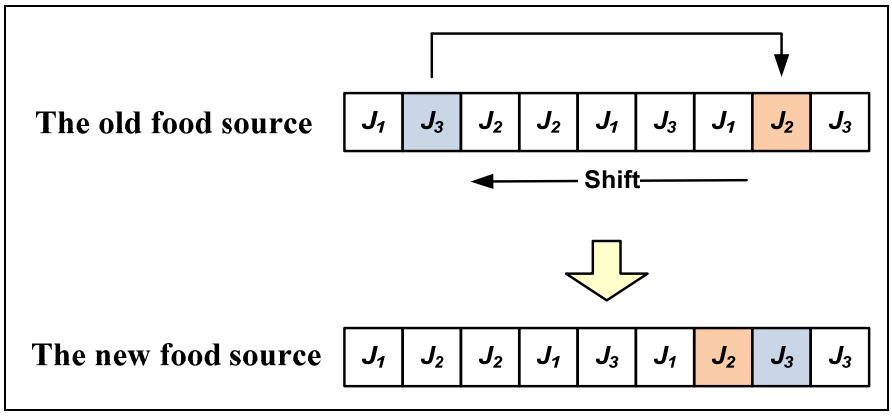


Figure 7.16 The inserting process in VNS method (for new x_b)

In Figure 7.15, for the exchanging process, suppose that the two random numbers α and β generated are 2 and 8. In this case, the job operations between the 2^{nd} dimension and 8^{th} dimension will be interchanged. In this example, the job operation J_3 at the 2^{nd} dimension and the job operation J_2 at the 8^{th} dimension will be swapped.

In Figure 7.16, for the inserting process, suppose that the two random numbers α and β generated are 2 and 8. In this case, the job operation J_3 at the 2^{nd} dimension will be removed, the job operations at 3^{rd} dimension through 8^{th} dimension will be shifted to the 2^{nd} dimension, and then inserted job operation J_3 in the 8^{th} dimension.

The Termination Step

Steps 2 to 5 are repeated until the number of iteration reaches the MCN.

7.2.3 Job Shop Scheduling Experiment Methodology

The aim of this experiment is to evaluate the solution quality based on the same parameter settings, i.e. the number of populations and the number of iterations. The best-so-far algorithm is compared with other approaches including a hybrid Particle Swarm Optimization with Variable Neighboring Search (PSO-VNS) (Tasgetiren et al., 2006), a multiple-type individual enhancement PSO (MPSO) (Lin et al., 2010), a Hybrid Intelligent Algorithm (HIA) (Ge et al., 2008), a Hybrid Evolutionary Algorithm (HEA) (Ge et al., 2007), a Hybrid Genetic Algorithm using parameterized active schedules (HGA-Param) (Goncalves et al., 2005), and an Improved Artificial Bee Colony algorithm (IABC) (Yao et al., 2010). The performance of our Best-so-far ABC algorithm is evaluated by testing them on the following 62 benchmark problems taken from the Operations Research Library (OR-Library) (Beasley, 1990).

- 3 problems from Fisher and Thompson (1963): referred as *ft06, ft10* and *ft20*.
- 40 problems from Lawrence (1984): referred as *la01-la40*.
- 5 problems from Adams, Balas, and Zawack (1988): referred as *abz05-abz09*.
- 10 problems from Applegate and Cook (1991): referred as *orb01-orb10*.
- 4 problems from by Yamada and Nakano (1992): referred as *yn01-yn04*.

The size of these problem instances range from 6 to 30 jobs and 5 to 20 machines.

The objective is to find the minimum makespan value from these benchmark problems. The numbers of employed and onlooker bees were set to 25. The MCN was set to 200. Each of the experiments was repeated 20 times with different random seeds. Our algorithm was coded in C++ and run on a PC with a 2.83-GHz Intel Core 2 Quad CPU and 4 GB of memory. We used the results reported in Goncalves et al. (2005), Tasgetiren et al. (2006), Lin et al. (2010), Ge et al. (2008) , Ge et al. (2007), and Yao et al. (2010) for the comparison. As a result, some problem instances cannot be evaluated in all aforementioned algorithms.

7.2.4 Job Shop Scheduling Results and Discussion

We compare our results with the results in other aforementioned studies in Table 7.8 to Table 7.10. In the Tables , "instance" means the problem name, "size" means the problem size of *n* jobs on *m* machines, "BKS" means the best known solution for the instance reported by Jain and Meeran (1999), "Best" means the best solution found by each algorithm, "Average" and "S.D." mean the average and standard deviation, respectively, of the results over 20 runs, and "RPE" means the relative percent error with respect to the best known solution which is calculated from the equation below.

$$RPE = \frac{(Best - BKS)}{BKS} * 100 \tag{7.17}$$

In each table of comparison results, boldface represents problems in which our Best-so-far ABC reached a higher quality solution than at least one of the previously reported algorithms. It can be noticed from the "Best", "Average", "S.D.", and "RPE" values in Table 7.8-7.10.

The first comparison is based on the results of an Improved Artificial Bee Colony algorithm (IABC) proposed by Yao et al. (2010). The comparison results are shown in Table 7.8.

Table 7.8 Comparison results of the Best-so-far ABC and IABC

Instance	Size (n x m)	BKS	Best-so-far ABC		IABC	
			Best	RPE	Best	RPE
ft06	6x6	55	55	0.00	55	0.00
ft10	10x10	930	930	0.00	930	0.00
ft20	20x5	1165	1165	0.00	1165	0.00
la01	10x5	666	666	0.00	666	0.00
la06	15x5	926	926	0.00	926	0.00
la11	20x5	1222	1222	0.00	1222	0.00
la16	10x10	945	**945**	**0.00**	977	3.39
la21	15x10	1046	**1046**	**0.00**	1051	0.48
la26	20x10	1218	1218	0.00	1218	0.00
la31	30x10	1784	1784	0.00	1784	0.00
la36	15x15	1268	**1268**	**0.00**	1275	0.55
Mean				0.00		1.47

From Table 7.8, it can be seen that the Best-so-far ABC generates better results in terms of the best makespan (Best) and relative percent error (RPE) than the IABC. The Best-so-far ABC method is able to find all of the best known solutions for the 11 instances reported and average of RPE for Best-so-far ABC is 0% whereas the IABC is able to find only 8 best known solutions among 11 instances reported with an average RPE of 1.47 %.

The second comparison is based on the results of a multiple-type individual enhancement PSO (MPSO) proposed by Lin et al. (2010), a Hybrid Intelligent Algorithm (HIA) proposed by Ge et al. (2008), a Hybrid Evolutionary Algorithm (HEA) proposed by Ge et al. (2007), and a Hybrid Genetic Algorithm using parameterized active schedules (HGA-Param) proposed by Goncalves et al. (2005). The comparison results are shown in Table 7.9 and Table 7.10.

According to Table 7.9 and Table 7.10, for instances *ft06, ft10, ft20, and la01-la40*, our Best-so-far ABC can find the best known solution (Best) for 41 of 43 instances. These results are better than MPSO, HIA, HEA, and HGA, which can only find 35, 32, 29, and 31 best known solutions among 43 instances respectively. Moreover, in terms of the average RPE, the Best-so-far ABC generates better results than these aforementioned approaches. The average RPE for the Best-so-far ABC is 0.03% whereas MPSO, HIA, HEA, and HGA got 0.15%, 0.33%, 0.39%, and 0.44% respectively.

Table 7.9 Comparison results of the Best-so-far ABC, MPSO, and HIA

Instance	Size ($n \times m$)	BKS	Best-so-far ABC		MPSO		HIA	
			Best	RPE	Best	RPE	Best	RPE
ft06	6x6	55	55	0.00	55	0.00	55	0.00
ft10	10x10	930	930	0.00	930	0.00	930	0.00
ft20	20x5	1165	**1165**	0.00	**1165**	0.00	**1165**	0.00
la01	10x5	666	666	0.00	666	0.00	666	0.00
la02	10x5	655	655	0.00	655	0.00	655	0.00
la03	10x5	597	597	0.00	597	0.00	597	0.00
la04	10x5	590	590	0.00	590	0.00	590	0.00
la05	10x5	593	593	0.00	593	0.00	593	0.00
la06	15x5	926	926	0.00	926	0.00	926	0.00
la07	15x5	890	890	0.00	890	0.00	890	0.00
la08	15x5	863	863	0.00	863	0.00	863	0.00
la09	15x5	951	951	0.00	951	0.00	951	0.00
la10	15x5	958	958	0.00	958	0.00	958	0.00
la11	20x5	1222	1222	0.00	1222	0.00	1222	0.00
la12	20x5	1039	1039	0.00	1039	0.00	1039	0.00
la13	20x5	1150	1150	0.00	1150	0.00	1150	0.00
la14	20x5	1292	1292	0.00	1292	0.00	1292	0.00
la15	20x5	1207	1207	0.00	1207	0.00	1207	0.00
la16	10x10	945	945	0.00	945	0.00	945	0.00
la17	10x10	784	784	0.00	784	0.00	784	0.00
la18	10x10	848	848	0.00	848	0.00	848	0.00
la19	10x10	842	842	0.00	842	0.00	842	0.00
la20	10x10	902	902	0.00	902	0.00	902	0.00
la21	15x10	1046	1046	0.00	1046	0.00	1046	0.00
la22	15x10	927	927	0.00	932	0.54	932	0.54
la23	15x10	1032	1032	0.00	1032	0.00	1032	0.00
la24	15x10	935	**935**	0.00	941	0.64	950	1.60
la25	15x10	977	**977**	0.00	**977**	0.00	979	0.20
la26	20x10	1218	1218	0.00	1218	0.00	1218	0.00
la27	20x10	1235	**1235**	0.00	1239	0.32	1256	1.70
la28	20x10	1216	**1216**	0.00	**1216**	0.00	1227	0.90
la29	20x10	1152	**1164**	1.04	1173	1.82	1184	2.78
la30	20x10	1355	1355	0.00	1355	0.00	1355	0.00
la31	30x10	1784	1784	0.00	1784	0.00	1784	0.00
la32	30x10	1850	1850	0.00	1850	0.00	1850	0.00
la33	30x10	1719	1719	0.00	1719	0.00	1719	0.00
la34	30x10	1721	1721	0.00	1721	0.00	1721	0.00
la35	30x10	1888	1888	0.00	1888	0.00	1888	0.00
la36	15x15	1268	**1268**	0.00	1278	0.79	1281	1.03
la37	15x15	1397	**1397**	0.00	1411	1.00	1415	1.29
la38	15x15	1196	**1196**	0.00	1208	1.00	1213	1.42
la39	15x15	1233	**1233**	0.00	**1233**	0.00	1246	1.05
la40	15x15	1222	**1224**	0.16	1225	0.25	1240	1.47
Mean				0.03		0.15		0.33

Table 7.10 Comparison results of the Best-so-far ABC, HEA, and HGA-Param

Instance	Size (n x m)	BKS	Best-so-far ABC		HEA		HGA-Param	
			Best	RPE	Best	RPE	Best	RPE
ft06	6x6	55	55	0.00	55	0.00	55	0.00
ft10	10x10	930	930	0.00	930	0.00	930	0.00
ft20	20x5	1165	**1165**	0.00	1169	0.34	**1165**	0.00
la01	10x5	666	666	0.00	666	0.00	666	0.00
la02	10x5	655	655	0.00	655	0.00	655	0.00
la03	10x5	597	597	0.00	597	0.00	597	0.00
la04	10x5	590	590	0.00	590	0.00	590	0.00
la05	10x5	593	593	0.00	593	0.00	593	0.00
la06	15x5	926	926	0.00	926	0.00	926	0.00
la07	15x5	890	890	0.00	890	0.00	890	0.00
la08	15x5	863	863	0.00	863	0.00	863	0.00
la09	15x5	951	951	0.00	951	0.00	951	0.00
la10	15x5	958	958	0.00	958	0.00	958	0.00
la11	20x5	1222	1222	0.00	1222	0.00	1222	0.00
la12	20x5	1039	1039	0.00	1039	0.00	1039	0.00
la13	20x5	1150	1150	0.00	1150	0.00	1150	0.00
la14	20x5	1292	1292	0.00	1292	0.00	1292	0.00
la15	20x5	1207	1207	0.00	1207	0.00	1207	0.00
la16	10x10	945	945	0.00	945	0.00	945	0.00
la17	10x10	784	784	0.00	784	0.00	784	0.00
la18	10x10	848	848	0.00	848	0.00	848	0.00
la19	10x10	842	842	0.00	–	–	842	0.00
la20	10x10	902	902	0.00	–	–	907	0.55
la21	15x10	1046	1046	0.00	1046	0.00	1046	0.00
la22	15x10	927	927	0.00	935	0.86	935	0.86
la23	15x10	1032	1032	0.00	1032	0.00	1032	0.00
la24	15x10	935	**935**	0.00	–	–	953	1.93
la25	15x10	977	**977**	0.00	–	–	986	0.92
la26	20x10	1218	1218	0.00	1218	0.00	1218	0.00
la27	20x10	1235	**1235**	0.00	1272	3.00	1256	1.70
la28	20x10	1216	**1216**	0.00	1227	0.90	1232	1.32
la29	20x10	1152	**1164**	1.04	1192	3.47	1196	3.82
la30	20x10	1355	1355	0.00	1355	0.00	1355	0.00
la31	30x10	1784	1784	0.00	1784	0.00	1784	0.00
la32	30x10	1850	1850	0.00	1850	0.00	1850	0.00
la33	30x10	1719	1719	0.00	1719	0.00	1719	0.00
la34	30x10	1721	1721	0.00	1721	0.00	1721	0.00
la35	30x10	1888	1888	0.00	1888	0.00	1888	0.00
la36	15x15	1268	**1268**	0.00	1287	1.50	1279	0.87
la37	15x15	1397	**1397**	0.00	1415	1.29	1408	0.79
la38	15x15	1196	**1196**	0.00	1213	1.42	1219	1.92
la39	15x15	1233	**1233**	0.00	1245	0.97	1246	1.05
la40	15x15	1222	**1224**	0.16	1242	1.64	1241	1.55
Mean				0.03		0.39		0.44

"-" stands for "not available".

The last comparison is based on the results of hybrid Particle Swarm Optimization with Variable Neighboring Search (PSO-VNS) by Tasgetiren et al. (2006). The comparison results are shown in Table 7.11.

Table 7.11 Comparison results of the Best-so-far ABC and PSO-VNS

Instance	Size ($n\ x\ m$)	BKS	Best-so-far ABC				PSO-VNS			
			Best	Average	S.D.	RPE	Best	Average	S.D.	RPE
abz05	10x10	1234	1234	1234.75	1.48	0.00	1234	1236.25	2.31	0.00
abz06	10x10	943	943	943.00	0.00	0.00	943	943.00	0.00	0.00
abz07	20x15	656	661	669.60	4.31	0.76	**659**	670.10	5.75	0.46
abz08	20x15	665	674	678.45	3.58	1.35	674	682.30	5.52	1.35
abz09	20x15	679	684	693.85	5.12	0.74	688	697.55	5.79	1.33
ft10	10x10	930	930	931.45	2.98	0.00	930	938.45	9.71	0.00
ft20	20x5	1165	1165	1169.45	5.81	0.00	1165	1175.25	5.30	0.00
la16	10x10	945	945	945.35	0.49	0.00	945	948.50	9.08	0.00
la19	10x10	842	842	842.00	0.00	0.00	842	844.00	3.78	0.00
la21	15x10	1046	**1046**	1050.00	3.23	0.00	1047	1053.80	6.01	0.10
la22	15x10	927	927	928.35	2.37	0.00	927	930.55	2.95	0.00
la24	15x10	935	935	939.10	1.89	0.00	935	939.70	3.63	0.00
la25	15x10	977	977	981.40	2.87	0.00	977	981.45	4.74	0.00
la27	20x10	1235	1235	1245.10	5.98	0.00	1235	1248.10	10.09	0.00
la28	20x10	1216	1216	1216.10	0.45	0.00	1216	1216.25	0.64	0.00
la29	20x10	1152	1164	1173.80	5.63	1.04	1164	1176.70	10.55	1.04
la36	15x15	1268	1268	1275.05	3.95	0.00	1268	1279.30	6.98	0.00
la37	15x15	1397	1397	1409.15	7.53	0.00	1397	1410.90	7.74	0.00
la38	15x15	1196	1196	1204.95	4.35	0.00	1196	1212.50	14.96	0.00
la39	15x15	1233	1233	1237.95	3.73	0.00	1233	1240.00	4.10	0.00
la40	15x15	1222	1224	1226.25	2.53	0.16	1224	1227.60	3.80	0.16
orb01	10x10	1059	1059	1073.00	8.96	0.00	1059	1076.05	10.58	0.00
orb02	10x10	888	**888**	889.05	0.39	0.00	889	889.45	1.79	0.11
orb03	10x10	1005	1005	1018.30	8.57	0.00	1005	1034.55	22.95	0.00
orb04	10x10	1005	1005	1010.90	3.75	0.00	1005	1011.30	6.42	0.00
orb05	10x10	887	887	890.10	2.00	0.00	887	892.45	4.81	0.00
orb06	10x10	1010	**1010**	1015.65	5.06	0.00	1013	1018.80	5.73	0.30
orb07	10x10	397	397	397.60	1.85	0.00	397	398.60	2.56	0.00
orb08	10x10	899	899	906.85	6.29	0.00	899	913.40	14.19	0.00
orb09	10x10	934	934	938.45	3.73	0.00	934	939.45	4.27	0.00
orb10	10x10	944	944	944.00	0.00	0.00	944	944.00	0.00	0.00
yn01	20x20	888	**891**	897.60	3.25	0.34	893	901.15	6.56	0.56
yn02	20x20	909	911	919.90	5.50	0.22	**910**	925.85	6.43	0.11
yn03	20x20	893	**897**	904.05	3.46	0.45	902	908.40	5.23	1.01
yn04	20x20	968	**972**	984.05	4.21	0.41	973	987.05	8.87	0.52
Mean				993.85	3.60	0.16		996.94	6.40	0.20

As shown in Table 7.11, the Best-so-far ABC can find the best known solutions (Best) for 26 of 35 instances reported where the PSO-VNS only finds the best known solutions with 23 of 35 instances reported. Moreover, the average RPE in the 35 instances reported for PSO-VNS is 0.20% whereas the average RPE for Best-so-far ABC is 0.16%. This shows that the Best-so-far ABC does better in finding the best known solution than the PSO-VNS. The mean of the average of makespan for the Best-so-far ABC is 993.85 whereas PSO-VNS is 996.94. The results indicate that the Best-so-far ABC gives slightly better makespan values on average than the PSO-VNS. It is also clear that the Best-so-far algorithm is more robust than the PSO-VNS algorithm as

shown by the lower standard deviations in the S.D. column. The average of the standard deviations for the Best-so-far ABC is only 3.60 whereas it is 6.40 for the PSO-VNS. Note that we do not report standard deviations for other comparisons because standard deviations are not reported in the papers presenting the other algorithms.

7.3 Clustering Application

7.3.1 Background

Clustering is an important technique applied in many applications domains, including machine learning (Fan et al., 2008; Anaya and Boticario, 2011), data mining (Yun et al., 2006; Ci et al., 2007), pattern recognition (Yuan and Kuo, 2008; Bassiou and Kotropoulos, 2011), image analysis (Das and Konar, 2009; Tan and Isa, 2011), information retrieval (Chan, 2008; Dhanapal, 2008;), and bioinformatics (Bhattacharya and De, 2010; Macintyre et al., 2010).

Clustering is a general term for a type of unsupervised learning. Clustering groups a set of data instances into categories (clusters) based on some similarity metric, such that data instances within the same cluster are similar to one another and dissimilar from items in other clusters. A distance measure is generally used for evaluating similarities between objects. Clustering is usually an iterative process in which data items are assigned to categories, and then possibly reassigned in each cycle, until some quality criterion is reached.

"Unsupervised" means that the categorization is developed based on the intrinsic structure of the data, without any need to supply the process with training items, that is, exemplars already associated with specific categories. The corresponding supervised procedure is known as classification. Classification methods learn to categorize test instances based on a training set of labeled instances. In this research, we focus on the clustering problem in which the number of clusters (K) is known a priori.

Let $O = \{o_1, o_2, \dots, o_n\}$ be a set of N objects to be clustered, let $C = \{c_1, c_2, \dots, c_k\}$ be a set of K cluster centers, and let D be the number of dimensions in the multidimensional space defining the instances. That is, each data instance o_i and cluster center c_j are defined by a vector of D values, one for each dimension. The clustering problem is stated as follows: given N objects, allocate each object to one of K clusters. The

objective of clustering is to minimize the sum of squared Euclidean distance between each object o_i and the center of the cluster c_j to which it belongs. This objective function can be expressed by equation (7.18) below.

$$\text{Min } D(w,c) = \sum_{i=1}^{N}\sum_{j=1}^{K}\sum_{d=1}^{D} w_{ij}\left\|o_{id} - c_{jd}\right\|^2 \tag{7.18}$$

where w_{ij} is the association weight of objects o_i in cluster j, i.e. w_{ij} is 1 if object i is allocated to cluster j otherwise 0.

The center of the cluster j ($c_j = \{c_{j1}, c_{j2}, \dots, c_{jd}\}$) is the set of mean of each dimension across all the objects assigned to the j^{th} cluster. It can be calculated by equation (7.19) below.

$$c_{jd} = \frac{1}{N_j}\sum_{i=1}^{N} w_{ij}o_{id} \quad for\ d = 1\ to\ D \tag{7.19}$$

where N_j is the number of objects in the j^{th} cluster.

The clustering problem can be viewed as a process of searching for an optimal assignment of objects or data instances to clusters. Each possible assignment is a potential solution. The greater the intra-cluster similarity, the better the solution.

K-means, a center-based clustering approach, is one of the most popular clustering algorithms because of its simplicity and efficiency in dealing with a large amount of data. However, the clusters resulting from the K-means algorithm are very sensitive to positions of the initial cluster centers in the problem space. Furthermore, the method always converges to the nearest local optimum from the starting position of the search.

To reduce sensitivity to the initial state in K-means, Zhang et al. (1999, 2000) proposed an algorithm called “K-harmonic means (KHM)”. This algorithm uses the harmonic mean of the distance from each data point to the cluster centers as the classification criterion, instead of the Euclidean distance usually used in K-means. Zhang et al. found that this approach produced solutions that were more robust than K-means with different initial configurations. However, their algorithm becomes trapped in local optima.

To solve the local optimum issue, many heuristic clustering algorithms have been introduced over the last decade. Selim and Alsultan (1991) applied a Simulated Annealing approach (SA) to avoid local optima. Sung and Jin (2000) proposed a heuristic algorithm for clustering that employed the tabu search method combined with two new procedures called "packing" and "releasing". Bandyopadhyay and Maulik (2002) introduced a genetic algorithm-based clustering algorithm called “KGA-clustering” to improve the searching capability of the K-Means algorithm. This method helps K-means to avoid getting stuck at local optimal values by using mutation mechanism in GA to improve global search ability.

Swarm intelligence (SI) is a class of meta-heuristic methods in the field of artificial intelligence, based on the collective behavior of social insects, flocks of birds, or schools of fish. In recent years, various SI methods have been successfully applied to address the local optima issue in clustering problems.

Shelokar et al. (2004) presented an Ant Colony Optimization (ACO) method for optimizing clustering. The algorithm emulates the real-life search behavior of ants. Shelokar et al. added a local search on the best 20% of the total solutions to improve solutions discovered by the ants. Yang and Kamel (2006) presented a multi-colony ACO approach to improve the effectiveness of the ACO algorithm. In this approach each ant colony provides ants moving at different speeds and uses different probability functions to generate diversity in the clustering results.

To improve the robustness and efficiency of K-means, Kao et al. (2008) proposed a hybrid technique based on combining the K-means algorithm, Nelder-Mead simplex, and Particle Swarm Optimization (PSO) called K-NM-PSO. Yang et al. (2009) also introduced a hybrid method based on combining the PSO and K-harmonic means called PSOKHM to enhance the global search ability of their algorithm.

Inspired by the decision making processes of bee swarms., Zhang et al. (2010) introduced an ABC algorithm with a new criterion to generate scout bees for solving a clustering problem. In this method all food sources of employed bees in every iteration will be sorted based on their fitness and then the employed bee whose food source is worst will change to be a scout bee in order to enhance global search capability of the algorithm. Karaboga and Ozturk (2011) applied their ABC algorithm with supervised learning to enhance the classification accuracy.

However, these algorithms produce solutions of lower quality than the best known solutions. In addition, they frequently show poor efficiency on large problems, requiring extensive computational resources. To improve these results, we propose using the Best-so-far Artificial Bee Colony algorithm (Banharnsakun et al., 2011), which is based on the foraging behavior of bee swarms, extending the method to solve clustering problems in a distributed computing environment. We focus on problems with large search spaces, which are common in data mining today and which tend to take a long time to solve. In this work we add a multiple patrilines concept to the Best-so-far ABC approach. This concept allows the computational load to be split among multiple processing units.

7.3.2 Best-so-far ABC Based Multiple Patrilines for Clustering

To cluster data, the goal is to find a global optimization of the sum of squared Euclidean distance ($D(w,c)$). In other words, we try to find the set of cluster centers that minimizes $D(w,c)$ value. A complete Best-so-far ABC based Multiples Patrilines code is loaded on each process and the details can be described below:

Let P is the number of processors, N is the number of food sources, and N_p is the number of food sources in the local memory of each processor. Then, $N_p = N/P$.

First, randomly distributed initial N solutions consisting of a set of cluster center position values are generated. These parameter sets are treated as the food sources. The objective function determines how good a food source is. It can be represented by $F(x_i)$.

$$F(x_i)\,, x_i \in R^D\,, i \in \{1,2,3..,N\}, \tag{7.20}$$

where x_i is the position of a food source as a D-dimensional vector and $F(x_i)$ is the objective function.

The manager patriline decomposes the entire colony of bees into P patrilines. Each patriline consists of N_p food sources, N_p employed bees and N_p onlooker bees. In other words, there would be one employed bee for each food source. It then assigns these patrilines to each computing node. Note that while not coordinating the computation, the manager patriline also performs the useful work in the same way as the worker patrilines do.

Steps 1 to 5 are carried out in parallel at each processor.

1. Employed bees in each patriline independently update their food sources in local memory by using equation (7.21) below.

$$\tilde{x}_{ij} = x_{ij} + \emptyset_{ij}(x_{ij} - x_{kj}) \quad (7.21)$$

In the equation (7.21), $\tilde{x}_{ij}$ is a new feasible position of food source that is modified from its previous position value (x_{ij}) based on a comparison with the randomly selected position from its neighboring food source (x_{kj}). $\emptyset_{ij}$ is a random number between [-1,1] which is used to adjust the old food source to become a new food source in the next iteration. $k \in \{1,2,3..,N_p\} \wedge k \neq i$ and $j \in \{1,2,3..,D\}$ are randomly chosen indexes. The difference between x_{ij} and x_{kj} is a difference of position in a particular dimension.

If a new food source ($\tilde{x}_{ij}$) is better than an old food source (x_{ij}), the old food source is replaced by the new food source.

2. When the employed bees in each patriline return to their hive, they share information with the onlooker bees in their patriline about candidate food sources they found. The onlooker bees select these food sources based on probability. Food sources of higher fitness have a larger chance of being selected by onlooker bees than ones of lower fitness. The probability that a food source will be selected can be obtained from equation (7.22) below.

$$P_i = \frac{fit_i}{\sum_{n=1}^{N_p} fit_n} \quad (7.22)$$

Where fit_i is the fitness value of the food source *i,* which is related to the objective function value ($F(x_i)$) of the food source *i*.

3. Onlooker bees in patriline then select the food sources from their employ bees based on the probabilities calculated from step 2 and update those food sources based on our best-so-far method by using equation (7.23) below.

$$\tilde{x}_{id} = x_{id} + \Phi fit_b(x_{id} - x_{bd}) \quad (7.23)$$

where $\tilde{x}_{ij}$ = The new candidate food source for onlooker bee *i* dimension *d, d=1,2,3,...D*
x_{id} = The selected food source *i* in dimension *d*
Φ = A random number between -1 to 1
f_b = The fitness value of the best food source so far
x_{bd} = The best so far food source in dimension *d*

The old food source will be replaced by the new food source if the new food source has a better fitness value.

4. An employed bee in patriline whose food source is rejected as low quality within a certain period by employed and onlooker bees will change to a scout bee to search randomly for new food source. A new food source will be regenerated by using equation (7.24) below.

$$\tilde{x}_{ij} = x_{ij} + \emptyset_{ij}\left[\omega_{max} - \frac{iteration}{MCN}(\omega_{max} - \omega_{min})\right]x_{ij} \quad (7.24)$$

where $\tilde{x}_{ij}$ is a new feasible food source of a scout bee that is modified from the current food source of an abandoned food source (x_{ij}) and $\emptyset_{ij}$ is a random number between [-1,1]. The value of ω_{max} and ω_{min} represent the maximum and minimum percentage of the food source adjustment for the scout bee. The value of ω_{max} and ω_{min} are fixed to 1 and 0.2 respectively. These parameters were empirically chosen by the experimenter (Banharnsakun et al., 2011). With these selected values, the adjustment of scout bee's food source based on its current food source will linearly decrease from 100 percent to 20 percent in each experiment round.

5. Each patriline exchanges its local best food sources with its neighboring subgroups based on exchanging method proposed by Banharnsakun et al. (2010b). In this method the local best food source from one subgroup will replace the local worst food source on another subgroup.

A parameter called Maximum Cycle Number or MCN (Karaboga and Basturk, 2007) determines the number of iterations and is a termination criterion. Steps 1 to 5 will be repeated until the number of iteration is equal to the MCN.

6. In the last iteration, the manager patriline gathers the local best food sources from all worker patrilines and calculates the global best food source.

7.3.3 Clustering Experiments and Discussion

To demonstrate the effectiveness of our proposed technique, we divided our experiment into two parts. In the first part we compared the solution quality of our Best-so-far ABC with Multiple Patrilines algorithm with previous research on four standard data sets

taken from the UCI Machine Learning Repository (Blake and Merz, 1998). The four datasets are Iris, Contraceptive Method Choice (CMC), Wine, and Vowel. For the second part we evaluated the performance of a Best-so-far ABC with Multiple Patrilines from the perspectives of scalability and speedup on a manufacturing data set obtained from disk drive manufacturing process (Taetragool and Achalakul, 2011). The details of these datasets are described below.

The Iris dataset contains three different species of Iris flower: Iris species Setosa, Iris Versicolour, and Iris Virginica. For each specie, there exist 50 instances with four numeric features including sepal length, sepal width, petal length, and petal width. Thus K (number of clusters) is 3, D (number of dimensions) is 4, and N (number of instances to classify) is 150.

The Contraceptive Method Choice (CMC) dataset is a subset of the 1987 National Indonesia Contraceptive Prevalence Survey. The samples are married women who were either not pregnant or did not know if they were while being in interviewed. The purpose of the study was to predict the current contraceptive method choice (629 instances of no-use, 334 instances of long-term methods, and 510 instances of short-term) of a woman based on her demographic and socioeconomic characteristics. Ten attributes were sampled for each respondent. For this problem, K is 3, D is 10 and N is 1473.

The Wine dataset is the result of a chemical analysis of wines grown in the same region in Italy that derived from three different cultivars. There are three categories in the data: class 1 (59 instances), class 2 (71 instances), and class 3 (48 instances). These categories were characterized by 13 features: alcohol, malic acid, ash, alcalinity of ash, magnesium, total phenols, flavanoids, nonflavanoid phenols, proanthocyanins, color intensity, hue, OD280/OD315 of diluted wines, and praline. For this problem, K is 3, D is 13 and N is 178.

The Vowel dataset includes six overlapping classes of Indian Telugu vowel sounds: δ (72 instances), a (89 instances), i (172 instances), u (151 instances), e (207 instances), and o (180 instances). Each class has three features corresponding to the first, second, and third vowel frequencies. For this problem, K is 6, D is 3 and N is 871.

The Hard Disk Drive (HDD) data set is a large data set containing characteristics of materials, tools and processes obtained from the disk drive manufacturing process. There are two categories in the data: product instances passed (7285 instances), and product instances failed (715 instances). These categories were characterized by 30 features including attributes and machine parameters. For this problem, K is 2, D is 30 and N is 8000.

The characteristics of these datasets are summarized in Table 7.12.

Table 7.12 Characteristics of data sets considered.

Name of dataset	No. of clusters	No. of features	Size of data set (size of clusters in parentheses)
Iris	3	4	150 (50, 50,50)
CMC	3	10	1473 (629, 334, 510)
Wine	3	13	178 (59, 71, 48)
Vowel	6	3	871 (72, 89, 172, 151, 207, 180)
HDD	2	30	8000 (7285, 715)

We used the same software for all experiments. We implemented our algorithm in C++ with the MPI library and executed it on a three-machine cluster running Windows XP. The processor on each machine was the Intel Core 2 Quad CPU Q6600 with a speed of 2.40 GHz and 2 GB of RAM. Thus, the entire cluster provides a total of 12 computing units (four cores times three machines). The numbers of employed and onlooker bees were set to 36. The number of patrilines used (denoted as "#p" in Table 7.13 through Table 7.16) ranged from a single patriline to 12 patrilines (matching the number of available processors). Each experiment was repeated 10 times with different random seeds. The MCN was set to 2000 for all datasets except the HDD dataset, where it was set to 1000. The cycle of steps in the ABC algorithm will be repeated until the number of iterations reaches the MCN.

7.3.3.1 Solution Quality

In this section we compared the clustering solution quality of the Best-so-far ABC with Multiple Patrilines algorithm with the results reported in Niknam and Amiri (2010), Fathian and Amiri (2008), and Niknam et al. (2008). The solution quality is expressed by following two criteria:

1. The best, the average, and the worst intra-cluster distances, as defined in Eq. (2.1), plus the standard deviations of these values. Note that numerically smaller values of results indicate better solutions.

2. F-Measure, the measurement that uses the concepts of precision and recall from information retrieval (Dalli, 2003; Handl et al., 2003) to measure of a clustering's accuracy. The F-measure treats every cluster as a query and every class as the desired result set for a query. The F-measure of cluster j (generated by the algorithm) and class i (as given by the class labels of the used benchmark data set) are given by following:

$$F(i,j) = \frac{2 \cdot r(i,j) \cdot p(i,j)}{r(i,j) + p(i,j)} \tag{7.25}$$

where r and p denote recall and precision respectively.

Recall is defined as $r(i,j) = \frac{n_{ij}}{n_i}$ and precision is defined as $p(i,j) = \frac{n_{ij}}{n_j}$

where n_{ij} is the number of class i members in cluster j, while n_j and n_i are the size of cluster j and class i respectively.

The overall F-Measure for the entire data set of size n is given by following:

$$F = \sum_i \frac{n_i}{n} \max_j (F(i,j)) \tag{7.26}$$

F has an upper bound of 1.0, which occurs if all instances of class i and only instances of class i are assigned to cluster j. Obviously, a bigger value of the F-Measure indicates a higher quality clustering solution.

Results for the Iris data are shown in Table 7.13. Our Best-so-far ABC with Multiple Patrilines algorithm produces best, average, and worst values of 96.65 in all cases. This is as good as or better than other algorithms. The improvement percentages of the average F-measure results obtained from our method are up to 12 % in comparison to other approaches.

We show the runtimes from the literature for informational purposes. However, we cannot directly compare our runtime results because of possible hardware and other procedural differences.

As expected, increasing the number of patrilines used in our algorithm tends to reduce the runtime, without reducing accuracy. The best result on this data set was obtained from the Best-so-far ABC with 8 patrilines. The performance dropped when the number of patrilines increased to 12. The dominant issue is problem size. There is just not

enough computation to achieve a performance improvement from a larger number of patrilines on the Iris data, which is considered to be a small problem.

Table 7.13 Results obtained by the algorithms run on Iris.

Method	Intra-cluster distance values			S.D.	Runtime (s)	F-Measure
	Best	Average	Worst			
Best-so-far-ABC – 1p	**96.65**	**96.65**	**96.65**	**0**	8.54	**0.898**
Best-so-far-ABC – 2p	**96.65**	**96.65**	**96.65**	**0**	4.24	**0.898**
Best-so-far-ABC – 4p	**96.65**	**96.65**	**96.65**	**0**	3.05	**0.898**
Best-so-far-ABC – 8p	**96.65**	**96.65**	**96.65**	**0**	1.29	**0.898**
Best-so-far-ABC – 12p	**96.65**	**96.65**	**96.65**	**0**	2.38	**0.898**
PSO-ACO-K	**96.65**	**96.65**	**96.65**	**0**	-	0.788
HBMO	96.75	96.95	97.75	0.53	-	0.781
ACO	97.10	97.17	97.81	0.36	-	0.779
SA	97.10	97.13	97.26	2.01	-	0.776
TS	97. 36	97.86	98.57	0.53	-	0.777
GA	113.98	125.19	139.78	14.56	-	0.778

Results for the Contraceptive Method Choice (CMC) dataset are shown in Table 7.14. For this problem, the Best-so-far ABC with Multiple Patrilines consistently produces better solutions than the comparison algorithms. Solution quality improves slightly as the numbers of patrilines increases to 12, and run time progressively decreases with more patrilines, since there is sufficient computation in this larger problem to gain an impact from distributing the work to a larger number of processes. Compared to other approaches, the improvement percentages of the average F-measure results of our method are about 7 %.

Table 7.14 Results obtained by the algorithms run on CMC.

Method	Intra-cluster distance values			S.D.	Runtime (s)	F-Measure
	Best	Average	Worst			
Best-so-far-ABC – 1p	**5,693.74**	**5,693.86**	5,694.29	0.16	113.92	**0.401**
Best-so-far-ABC – 2p	**5,693.72**	**5,693.73**	**5,693.75**	0.01	57.77	**0.401**
Best-so-far-ABC – 4p	**5,693.72**	**5,693.73**	**5,693.73**	0.01	29.44	**0.401**
Best-so-far-ABC – 8p	**5,693.72**	**5,693.85**	**5,694.10**	0.16	16.52	**0.401**
Best-so-far-ABC – 12p	**5,693.72**	**5,693.76**	**5,693.99**	0.08	10.50	**0.401**
PSO-ACO-K	5,694.28	5,694.28	5,694.28	0	-	0.334
HBMO	5,699.26	5,713.98	5,725.35	12.69	-	0.330
ACO	5,701.92	5,819.13	5,912.43	45.63	-	0.328
SA	5,849.03	5,893.48	5,966.94	50.86	-	0.325
TS	5,885.06	5,993.59	5,999.80	40.84	-	0.327
GA	5,705.63	5,756.59	5,812.64	50.36	-	0.324

The results for clustering the Wine dataset are shown in Table 7.15. For this problem, the Best-so-far ABC algorithm with Multiple Patrilines once again provides better

solutions than any of the comparison algorithms, and runtimes decrease as more patrilines are added. Moreover, the improvement on the average F-measure for our method when compared with other methods is up to 20%.

Table 7.15 Results obtained by the algorithms run on Wine.

Method	Intra-cluster distance values			S.D.	Runtime (s)	F-Measure
	Best	Average	Worst			
Best-so-far-ABC – 1p	**16,292.18**	**16,292.87**	**16,294.17**	0.71	90.02	**0.724**
Best-so-far-ABC – 2p	**16,292.18**	**16,292.82**	**16,294.17**	0.74	45.81	**0.724**
Best-so-far-ABC – 4p	**16,292.18**	**16,292.58**	**16,294.17**	0.61	24.30	**0.729**
Best-so-far-ABC – 8p	**16,292.18**	**16,292.28**	**16,292.69**	0.21	14.46	**0.729**
Best-so-far-ABC – 12p	**16,292.18**	**16,292.28**	**16,292.67**	0.20	10.72	**0.729**
PSO-ACO-K	16,295.31	16,295.31	16,295.31	0	-	0.521
HBMO	16,357.28	16,357.28	16,357.28	0	-	0.518
ACO	16,530.53	16,530.53	16,530.53	0	-	0.519
SA	16,473.48	17,521.09	18,083.25	753.08	-	0.515
TS	16,666.22	16,785.45	16,837.53	52.07	-	0.516
GA	16,530.53	16,530.53	16,530.53	0	-	0.515

Table 7.16 shows the results for clustering the Vowel dataset. For this large problem, the Best-so-far ABC with Multiple Patrilines produces better "best" solutions than any competing algorithm. The "average" and "worst" solutions are better than every algorithm except PSO-ACO-K. The highest quality solutions for our algorithm occur with four patrilines (in the "Best" and "Average" columns). Once again, run times decline as more patrilines are added, up to the maximum of 12.

Table 7.16 Results obtained by the algorithms run on Vowel.

Method	Intra-cluster distance values			S.D.	Runtime (s)	F-Measure
	Best	Average	Worst			
Best-so-far-ABC – 1p	**148,973.04**	148,997.11	149,067.50	32.14	112.92	0.650
Best-so-far-ABC – 2p	**148,971.76**	148,994.55	149,055.76	31.48	57.10	0.650
Best-so-far-ABC – 4p	**148,967.24**	**148,975.90**	149,040.43	22.72	28.87	**0.652**
Best-so-far-ABC – 8p	**148,967.24**	148,995.96	149,041.13	31.68	15.92	**0.652**
Best-so-far-ABC – 12p	**148,967.24**	148,985.79	149,070.94	31.60	10.43	**0.652**
PSO-ACO-K	148,976.00	148,976.00	**148,976.00**	**0**	-	**0.652**
HBMO	149,201.63	161,431.04	165,804.67	2,746.04	-	0.650
ACO	149,395.60	159,458.14	165,939.82	3,485.38	-	0.649
SA	149,370.47	161,566.28	165,986.42	2,847.08	-	0.645
TS	149,468.26	162,108.53	165,996.42	2,846.23	-	0.645
GA	149,513.73	159,153.49	165,991.65	3,105.54	-	0.647

The results from this experiment show that our proposed algorithm can produce more optimal solutions and thus is more effective than the other aforementioned methods on these standard data sets. Moreover, with a multiple patriline concept, the run times drop

as more patrilines are added to the computation, while the solution quality is maintained.

7.3.3.2 Performance and Scalability

The results from the previous section demonstrate that the Best-so-far ABC with Multiple Patrilines can produce solutions that are as good as or better than the current state-of-the-art clustering techniques. However, due to relatively small size of standard benchmark data sets presented in previous experiments, we cannot use them to analyze the performance of our Best-so-far ABC with Multiple Patrilines from the perspectives of scalability and speedup. In the context of parallel computing, scalability refers to the capability of a system to increase total throughput under an increased load when resources (processors) are added, while speedup refers to how much a parallel algorithm is faster than a corresponding sequential algorithm.

In order to demonstrate the effectiveness of the proposed algorithm in these perspectives, we applied the algorithm to the HDD data set obtained from a disk drive manufacturing process.

In this experiment we also used the F-Measure to evaluate the solution quality of clustering obtained from our proposed method. As for the performance analysis, speedup and efficiency described in previous Chapter (equation 6.6 and equation 6.7 respectively) are used as measurements.

We studied performance scalability of the Best-so-far ABC with Multiple Patrilines method by varying the number of attributes, number of data records, and number of processors. The number of attributes and the number of data records are properties that are associated with the problem size.

To observe the effect of the number of data records on performance scalability, we randomly selected between 200 and 8000 actual manufacturing data records. All data records initially had 30 attributes related to disk drive production. We varied the number of attributes used by randomly selecting subsets of 10 or 20 attributes, or using the full 30. Each combination of variables was repeated 10 times with random seeds. We then recorded the F-measure, the execution time in seconds (CT(sec)), speedup (S_p), and efficiency (E) in each case.

Variation in Number of Attributes

The effect of the number of attributes on the algorithm scalability is presented in this section. Table 7.17 and Figure 7.17 show speedup and efficiency results. Figure 7.18 illustrates the scalability plot of our Best-so-far ABC with Multiple Patrilines algorithm for the three problem sizes.

Table 7.17 Speedup and efficiency results of variation in number of attributes

Number of Processor(s)	disk drive manufacturing dataset with 800 data records			
	number of attributes = 10			
	F-Measure	CT (sec)	S_p	E
1	0.891	104.047	-	-
3	0.891	34.781	2.991	0.997
6	0.891	17.859	5.826	0.971
9	0.891	12.312	8.451	0.939
12	0.891	9.641	10.792	0.899
Number of Processor(s)	**disk drive manufacturing dataset with 800 data records**			
	number of attributes = 20			
	F-Measure	**CT (sec)**	S_p	E
1	0.890	202.015	-	-
3	0.892	67.609	2.988	0.996
6	0.892	34.265	5.896	0.983
9	0.892	23.297	8.671	0.963
12	0.892	17.984	11.233	0.936
Number of Processor(s)	**disk drive manufacturing dataset with 800 data records**			
	number of attributes = 30			
	F-Measure	**CT (sec)**	S_p	E
1	0.892	309.219	-	-
3	0.892	103.172	2.997	0.999
6	0.892	52.156	5.929	0.988
9	0.892	35.281	8.764	0.974
12	0.892	27.438	11.270	0.939

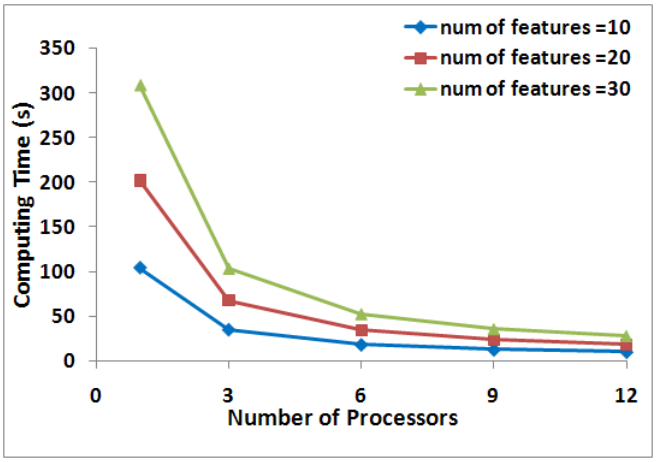


Figure 7.17 Varying the attributes size with 800 data records

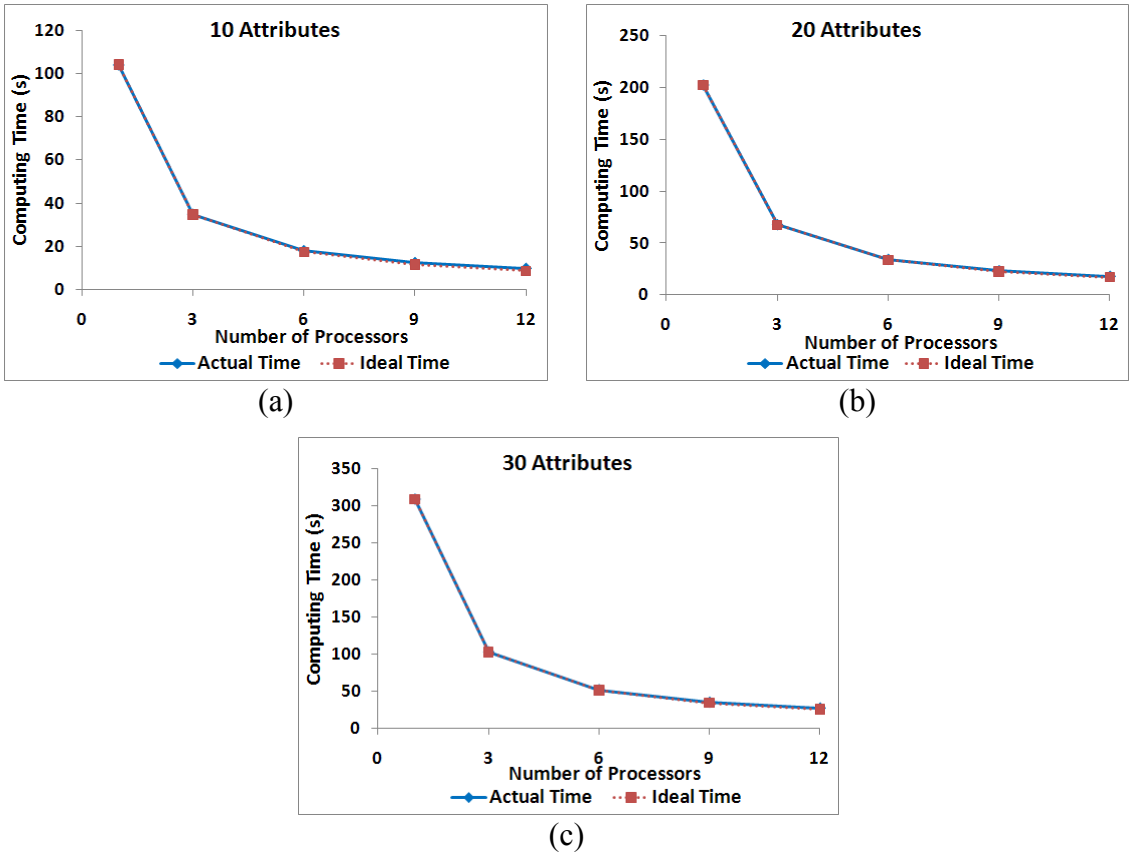


Figure 7.18 Comparing the ideal time to the actual time on various numbers of attributes

The results show that with a large problem size, the algorithm performance was very close to ideal, that is, the ratio of the time with a single processor to the number of processors. The efficiency was 93% or better. Performance dropped from the ideal for smaller problem sizes. Thus, the algorithm was more scalable as the problem size increased. For small problems, there was not sufficient computation to gain an impact from a large number of processors.

Variation in Number of Data Records

The effect of the number of data records on the algorithm scalability is discussed in this section.

For 200 data records, which is a small problem size, the efficiency was as much as 20% lower than ideal. The amount of computation required was not enough to overcome the synchronization overhead incurred when a large number of processors worked together.

However, in disk drive manufacturing process, several thousand data records are produced daily. Engineers and analysts depend on these data to analyze manufacturing yield. We thus expect that our parallel algorithm will scale well in the real world.

Table 7.18 and Figure 7.19 show speedup and efficiency results as the number of data records is varied. The actual and ideal performance of the algorithm with 30 attributes and a varying number of data records are also plotted in Figure 7.20.

Table 7.18 Speedup and efficiency results of variation in number of data records

Number of Processor(s)	**disk drive manufacturing dataset with 30 attributes**			
	number of records = 2000			
	F-Measure	**CT (sec)**	S_p	E
1	0.893	770.609	-	-
3	0.893	258.125	2.985	0.995
6	0.893	129.641	5.944	0.991
9	0.893	86.578	8.901	0.989
12	0.893	68.328	11.278	0.940
Number of Processor(s)	**disk drive manufacturing dataset with 30 attributes**			
	number of records = 4000			
	F-Measure	**CT (sec)**	S_p	E
1	0.892	1535.281	-	-
3	0.893	513.906	2.987	0.996
6	0.892	258.188	5.946	0.991
9	0.892	172.297	8.911	0.990
12	0.892	135.656	11.317	0.943
Number of Processor(s)	**disk drive manufacturing dataset with 30 attributes**			
	number of records = 8000			
	F-Measure	**CT (sec)**	S_p	E
1	0.883	3092.828	-	-
3	0.883	1031.953	2.997	0.999
6	0.883	516.597	5.987	0.998
9	0.883	344.697	8.973	0.997
12	0.883	267.844	11.547	0.962

Table 7.18 and Figure 7.19 show that the performance of our multiple patrilines algorithm improves as the number of data records grows larger. Figure 7.20 shows that execution times dropped almost linearly as more processors were added to the computation.

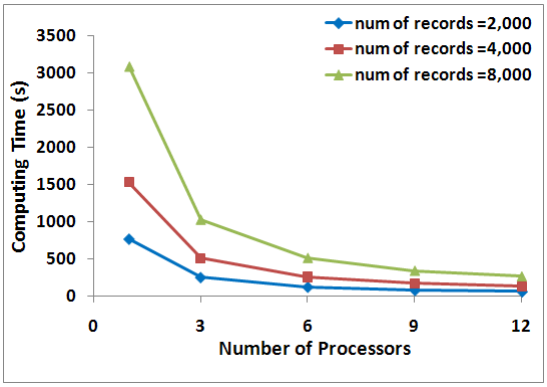
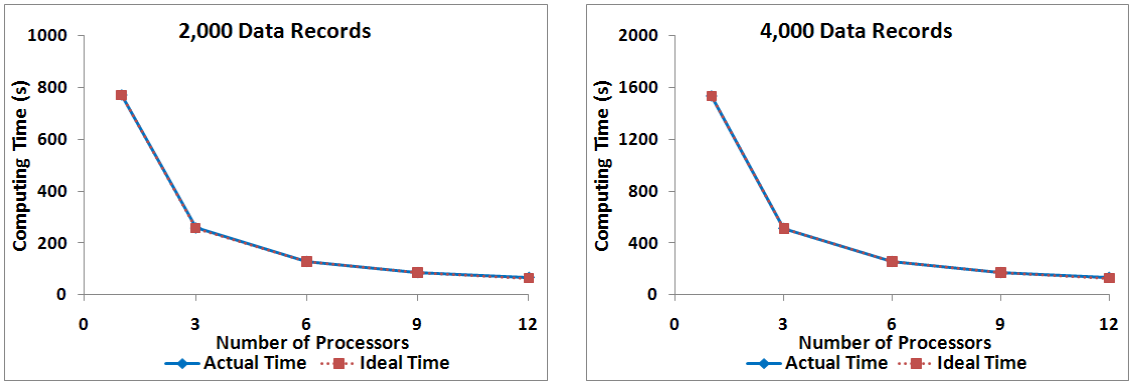
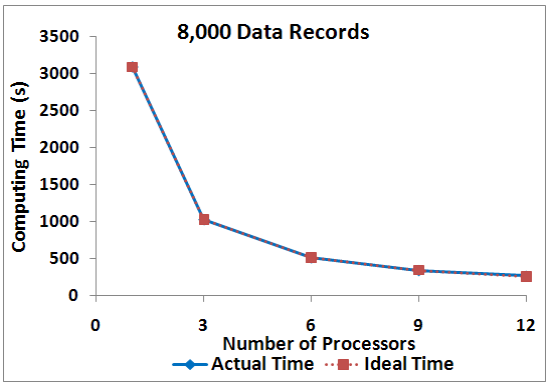


Figure 7.19 Varying the data record size with 30 attributes



(a) (b)



(c)

Figure 7.20 Comparing the ideal time to the actual time on various data record size

We can evaluate the solution quality of clustering based on the F-Measure. Table 7.17 and Table 7.18 show that our algorithm yields reasonably good results based on F-measure values in all cases. Furthermore, the solution quality varies very little across problem sizes and number of processing units, suggesting that Best-so-far ABC with Multiple Patrilines is a robust algorithm.

To summarize, our Best-so-far ABC with Multiple Patrilines algorithm can be used to save time and cost in clustering process. The time spent on the data clustering dropped from an hour to five minutes when we used 8000 sampling data records.

These results suggest that our distributed algorithm is well-suited to process huge volumes of input data within reasonable time while still keeping good solution quality.

CHAPTER 8 CONCLUSION AND FUTURE WORK

Optimization technique is used widely in many application domains. Searching for optimal solutions from all feasible solutions is very hard and often requires a considerably large computational time and resources. Heuristic methods are thus adopted.

Artificial Bee Colony (ABC) is one of many biological-inspired heuristic optimization algorithms mimicking the foraging behavior of honey bee swarm. The results from previous literature showed that ABC algorithm is well suited to solve various complex optimization problems. However, ABC is only good at exploration but relatively poor at exploitation. Its convergence speed is also an issue in some situations.

In this dissertation, the optimization framework based on the Artificial Bee Colony (ABC) algorithm has been introduced. This framework is used to describe how to map the parameters between the ABC algorithm and its applications. To improve the performance of the original ABC algorithm, the modified version of ABC algorithm called *Best-so-far ABC* has been proposed. In the original ABC algorithm, each onlooker bee selects a food source based on a probability that varies according to the fitness function explored by a single employed bee. In our best-so-far method, onlooker bees compare the information from all employed bees to select the best-so-far candidate food source. This will bias the solution handled by onlooker bees towards the optimal solution. Moreover, if the solution appears to stagnate in a local optimum, the scout bee can randomly generate a new position based on radius search method in order to maintain the diversity of new food sources. Furthermore, to enhance the computational throughput and global search capability in a large problem size, the implementation of the distributed Best-so-far ABC algorithm has been presented.

The performance of the best-so-far ABC method was then compared with the original ABC algorithm and the BSO algorithm using a set of benchmark functions. The computational complexity and the rate of convergence on both best-so-far ABC method and the original ABC method have been addressed.

The numerical results from the experiments provide evidence that the best-so-far ABC outperforms all the mentioned algorithms in both quality and convergence rate. We

further applied the best-so-far method to optimize the practical applications including image registration and job shop scheduling.

In the image registration application, we aimed to find the transformation variables that can maximize the mutual information values of the images. Several image pairs were used in the experiments. The results showed that our algorithm can arrive at the convergence state more quickly. The average improvement on the runtime for the best-so-far method compared with the original model on all experiments of image pairs is up to 70%. Lastly, the mutual information value produced by the best-so-far method is higher implying that the registration quality is also enhanced.

In the job shop scheduling application, we extend the Best-so-far ABC to find the job operation scheduling list that minimizes the makespan value. We empirically assessed the performance of our proposed method on 62 benchmark problems taken from the Operations Research Library (OR-Library). The results demonstrate that the proposed method is able to produce higher quality solutions than the current state-of-the-art heuristic-based algorithms. The average of relative percent error (RPE) with respect to the best known solution for the Best-so-far ABC on all 62 benchmark problems is only 0.09%.

However, in many real-world problems, there exist a large computational and resource demands and a single processor may not be computationally sufficient. To enhance the computational throughput and the global search capability on complex large-scale optimization problems, the parallel implementation of our proposed algorithm on distributed environments based on multiple patrilines concept has been introduced.

In order to demonstrate the performance of our distributed Best-so-far ABC algorithm, we apply this proposed algorithm to solve the clustering application, which is one of several real-world problems that require a large computational and resource demands.

The solution quality of our distributed Best-so-far ABC algorithm was assessed on several real datasets. The experiment was benchmarked against several state-of-the-art methods. The results obtained from our proposed method show that the distributed Best-so-far ABC can find the best known solution more effectively and efficiently than other aforementioned approaches. The improvement of solution quality of clustering based on the F-Measure is up to 20%.

To evaluate the performance and scalability of the proposed parallel algorithm, we further applied the distributed Best-so-far ABC algorithm to categorize the data set obtained from a disk drive manufacturing process, which is a relatively large problem.

The performance scalability was studied by varying several parameters such as number of attributes, number of data records, and number of processors. The results indicate that the distributed Best-so-far ABC algorithm achieves a near linear speedup with the efficiency more than 93%, while the solution quality is maintained. The optimization problem can thus be solved faster.

Thus, we can conclude that our Best-so-far ABC and its distributed version are effective in the perspective of solution quality and are efficient in the terms of the processing time required. The algorithm can serve as an alternative in several application domains in the future.

Finally, there are still some rooms for improvement for the works presented in this book. The future research directions are suggested here with the concluding remarks of this dissertation.

Multi-Objective Optimization

Multi-objective optimization problem is a problem of finding the trade-off optimal solution set, which represents the best possible compromises among the multiple conflicting objectives. Each objective corresponds to a different optimal solution. Thus, there is no solution from the set of trade-off optimal solutions can be said to be better than any other. It means that there is no single solution for this problem.
In this research, we only focus on a single objective optimization problem. Nevertheless, there exist several optimization problems considered as multiple objective problem. This should be considered to apply the Best-so-far ABC to optimize them.

Best-so-far Approach on other Swarm Intelligence Algorithms

The Best-so-far method is a general concept to improve the group decision making in the search process. So we can apply it to other swarm intelligence based algorithms such as Particle Swarm Optimization (PSO), and Artificial Immune System (AIS).

The Distributed Best-so-far ABC on CUDA Technology

CUDA (Compute Unified Device Architecture) is a parallel computing architecture in NVIDIA graphics processing units (GPUs). In recent years, GPUs have evolved to the point where many real-world applications are easily implemented on them and run significantly faster than on multi-core systems. Future computing architectures will be hybrid systems with parallel-core GPUs working in tandem with multi-core CPUs. It is thus interesting and challenging to implement the Best-so-far ABC on CUDA.

Adoption of Distributed Best-so-far ABC Best in other Real-World Applications

There exist many real-world problems that require a large computational and resource demands. Our distributed Best-so-far ABC should be considered in order to save time and cost. The examples of the interesting real-world problems are the job scheduling in Grid Computing and the performance tuning in the Cloud Computing.

REFERENCES

Adams, J., Balas, E., and Zawack, D., 1988, "The Shifting Bottleneck Procedure for Jobshop Scheduling", **Management Science**, Vol. 34, pp. 391-401.

Aghdam, M.H., Ghasem-Aghaee, N., and Basiri, M.E., 2009, "Text Feature Selection Using Ant Colony Optimization", **Expert Systems with Applications**, Vol. 36, pp. 6843-6853.

Akbari, R., Mohammadi, A., and Ziarati, K., 2010, "A Novel Bee Swarm Optimization Algorithm for Numerical Function Optimization", **Communications in Nonlinear Science and Number Simulation**, Vol. 15, pp. 3142-3155.

Alon, U., Barkai, N., Notterman, D.A., Gish, K., Ybarra, S., and Mack, D., 1999, "Broad Patterns of Gene Expression Revealed by Clustering Analysis of Tumor and Normal Colon Tissues Probed by Oligonucleotide Arrays", **Proc Natl Acad Sci USA**, Vol. 96, pp. 6745-6750.

Anaya, A.R., and Boticario, J.G., 2011, "Application of Machine Learning Techniques to Analyse Student Interactions and Improve the Collaboration Process", **Expert Systems with Applications**, Vol. 38, pp. 1171-1181.

Applegate, D., and Cook, W., 1991, "A Computational Study of Jobshop Problem", **ORSA Journal of Computing**, Vol. 3, pp. 149-156.

Asadzadeh, L., and Zamanifar, K., 2010, "An Agent-Based Parallel Approach for the Job Shop Scheduling Problem with Genetic Algorithms", **Mathematical and Computer Modeling**, Vol. 52, pp. 1957-1965.

Bagchi, T.P., Gupta, J.N.D., and Sriskandarajah, C., 2006, "A Review of TSP Based Approaches for Flowshop Scheduling", **European Journal of Operational Research**, pp. 816-854.

Bandyopadhyay, S., and Maulik, U., 2002, "An Evolutionary Technique Based on K Means Algorithm for Optimal Clustering in R^N", **Information Science**, Vol. 146, pp. 221-237.

Banharnsakun, A., Achalakul, T., and Sirinaovakul, B., 2010a, "ABC-GSX: A Hybrid Method for Solving the Traveling Salesman Problem", **In: Proceedings of the World Congress on Nature and Biologically Inspired Computing (NaBIC2010)**, pp. 7-12.

Banharnsakun, A., Achalakul, T., and Sirinaovakul, B., 2010b, "Artificial Bee Colony Algorithm on Distributed Environments", **In: Proceedings of the World Congress on Nature and Biologically Inspired Computing (NaBIC2010)**, pp. 13-18.

Banharnsakun, A., Achalakul, T., and Sirinaovakul, B., 2011, “The Best-so-far Selection in Artificial Bee Colony Algorithm”, **Applied Soft Computing,** Vol. 11, pp. 2888-2901.

Bassiou, N., and Kotropoulos, C., 2011, “Long Distance Bigram Models Applied to Word Clustering”, **Pattern Recognition**, Vol. 44, pp. 145-158.

Baykasoğlu, A., Özbakir, L., and Tapkan, P., 2007, "Artificial Bee Colony Algorithm and its Application to Generalized Assignment Problem", **In Swarm Intelligence: Focus on Ant and Particle Swarm Optimization**, Felix, T., Chan, S., and Tiwari, M.K., (Eds.), Itech Education and Publishing, Vienna, pp. 532-564.

Beasley, J. E., 1990, “Or-library: Distributing Test Problems by Electronic Mail”, **Journal of the Operational Research Society**, Vol. 14, pp. 1069-1072.

Bella, J.E., and McMullen, P.R., 2004, “Ant Colony Optimization Techniques for the Vehicle Routing Problem”, **Advanced Engineering Informatics**, Vol. 18, pp. 41-48.

Bhattacharya, A., and De, R.K., 2010, “Average Correlation Clustering Algorithm (ACCA) for Grouping of Co-Regulated Genes with Similar Pattern of Variation in their Expression Values”, **Journal of Biomedical Informatics**, Vol. 43, pp. 560-568.

Biesmeijer, J.C., and Seeley, T.D., 2005, “The Use of Waggle Dance Information by Honey Bees throughout their Foraging Careers”, **Behav. Ecol. Sociobiol**, Vol. 59, pp. 133-142.

Blake, C.L., and Merz, C.J., 1998, **UCI Repository of Machine Learning Databases** [Online], Available: http://archive.ics.uci.edu/ml/datasets.html [2011, January 11].

Carlier, J., and Pinson, E., 1989, “An Algorithms for Solving the Job-Shop Problem”, **Management Science**, Vol. 35, pp. 164-176.

Chan, C.-C.H., 2008, “Intelligent Spider for Information Retrieval to Support Mining-Based Price Prediction for Online Auctioning”, **Expert Systems with Applications**, Vol. 34, pp. 347-356.

Chen, H., Varshney, P., and Arora, M., 2003, “Mutual Information Based Image Registration for Remote Sensing Data”, **International Journal of Remote Sensing**, Vol. 24, pp. 3701-3706.

Chong, C.S., and Low, M.Y.H., 2006, “A Bee Colony Optimization Algorithm to Job Shop Scheduling”, **In: Proceedings of the 2006 Winter Simulation Conference**, pp. 1954-1961.

Chuang, L.Y., Chang, H.W., Tu, C.J., and Yang, C.H., 2008, “Improved Binary PSO for Feature Selection Using Expression Data”, **Computation Biology and Chemistry**, Vol. 32, pp. 29-38.

Ci, S., Guizani, M., and Sharif, H., 2007, "Adaptive Clustering in Wireless Sensor Networks by Mining Sensor Energy Data", **Computer Communications**, Vol. 30, pp. 2968-2975.

Cover, T.M., and Thomas J.A., 2006, **Elements of Information Theory**, John Wiley & Sons, New York.

Croes, G.A., 1958, ''A Method for Solving Traveling Salesman Problems'', **Operations Research**, Vol. 6, pp. 791-812.

Das, S., and Konar, A., 2009, "Automatic Image Pixel Clustering with an Improved Differential Evolution", **Applied Soft Computing**, Vol. 9, pp. 226-236.

Dhanapal, R., 2008, "An Intelligent Information Retrieval Agent", **Knowledge-Based Systems**, Vol. 21, pp. 466-470.

Dorigo, M., and Stützle, T., 2004, **Ant Colony Optimization**, MIT Press, Cambridge.

Fan, S., Chen, L., and Lee, W.-J., 2008, "Machine Learning Based Switching Model for Electricity Load Forecasting", **Energy Conversion and Management**, Vol. 49, pp. 1331-1344.

Fathian, M., and Amiri, B., 2008, "A Honey-Bee Mating Approach on Clustering", **The International Journal of Advanced Manufacturing Technology**, Vol. 38, pp. 809-821.

Fisher, H., and Thompson, G.L., 1963, "Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules", **In: Industrial Scheduling**, Muth, J.F., and Thompson, G.L., (Eds.), Prentice-Hall, Englewood Cliffs, pp. 225-251.

Fisher, M., 1981, "The Lagrangian Relaxation Method for Solving Integer Programming Problems", **Management Science**, Vol. 27, pp. 1-18.

Fogel, D.B., 2006, **Evolutionary Computation: Toward a New Philosophy of Machine Intelligence**, IEEE Press, New York.

Fournier, J.R.L., and Pierre, S., 2005, "Assigning Cells to Switches in Mobile Networks Using an Ant Colony Optimization Heuristic", **Computer Communications**, Vol. 28, pp. 65-73.

French, S., 1982, **Sequencing and Scheduling: an Introduction to the Mathematics of the Job Shop**, Wiley, New York, USA.

Garey, M.R., Johnson D.S., and Sethi, R., 1976, "The Complexity of Flowshop and Jobshop Scheduling", **Mathematics of Operations Research**, Vol. 1, pp. 117-129.

Ge, H., Du, W., and Qian, F., 2007, "A Hybrid Algorithm Based on Particle Swarm Optimization and Simulated Annealing for Job Shop Scheduling", **In: Proceedings of the Third International Conference on Natural Computation**, Vol. 3, pp. 715-719.

Ge, H.-W., Sun, L., Liang, Y.-C., and Qian, F., 2008, "An Effective PSO and AIS-Based Hybrid Intelligent Algorithm for Job-Shop Scheduling", **IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans**, Vol. 38, pp 358-368.

Girard, M.B., Mattila, H.R., and Seeley, T.D., 2011, "Recruitment-Dance Signals Draw Larger Audiences when Honey Bee Colonies have Multiple Patrilines", **Insectes Sociaux**, Vol. 58, pp. 77-86.

Glover, F., and Laguna, M., 1997, **Tabu Search**, Kluwer Academic Publishers, Massachusetts.

Golub, T.R., Slonim, D.K., Tamayo, P., Huard, C., Gaasenbeek, M., and Mesirov, J.P., 1999, "Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring", **Science**, Vol. 286, pp. 531-537.

Goncalves, J.F., Mendes, J.J.D.M., and Resende, M.G.C., 2005, "A Hybrid Genetic Algorithm for the Job Shop Scheduling Problem", **European Journal of Operational Research**, Vol. 167, pp. 77-95.

González, A., Valero-García M., and Cerio, L.De., 1995, "Executing Algorithms with Hypercube Topology on Torus Multicomputers", **IEEE Transactions on Parallel and Distributed Systems**, Vol. 6, pp. 803-814.

Goshtasby, A., 2005, **2-D and 3-D Image Registration for Medical, Remote Sensing, and Industrial Applications**, John Wiley & Sons, New Jersey.

Grama, A., Gupta, A., Karypis, G., and Kumar, V., 2003, **Introduction to Parallel Computing**, Addison-Wesley, New York.

Gui, J., Wang, S.-L., and Lei, Y.-K., 2010, "Multi-step Dimensionality Reduction and Semi-supervised Graph-based Tumor Classification Using Gene Expression Data", **Artificial Intelligence in Medicine**, Vol. 50, pp. 181-191.

Hansen, P., Mladenović, N., and Pérez, J.A.M., 2008, "Variable Neighbourhood Search: Methods and Applications", **4OR: A Quarterly Journal of Operations Research**, Vol. 6, pp. 319-360.

Heinonen, J., and Pettersson, F., 2007, "Hybrid Ant Colony Optimization and Visibility Studies Applied to a Job-Shop Scheduling Problem", **Applied Mathematics and Computation**, Vol. 187, pp. 989-998.

Iizuka, N., Oka, M., Yamada-Okabe, H., Nishida, M., Maeda, Y., and Mori, N., 2003, "Oligonucleotide Microarray for Prediction of Early Intrahepatic Recurrence of Hepatocellular Carcinoma after Curative Resection", **Lancet**, Vol. 361, pp. 923-929.

Jacquet, W., Nyssen, E., Bottenberg, P., Truyen, B., and de Groen, P., 2009, "2D Image Registration Using Focused Mutual Information for Application in Dentistry", **Computers in Biology and Medicine**, Vol. 39, pp. 545-553.

Jain, A.S., and Meeran, S., 1999, "Deterministic Job-Shop Scheduling: Past, Present and Future", **European Journal of Operational Research**, Vol. 113, pp. 390-434.

Jie, X., CaiYun, L., and Zhong, C., 2008, "A New Parallel Ant Colony Optimization Algorithm Based on Message Passing Interface", **Pacific-Asia Workshop on Computational Intelligence and Industrial Application**, pp. 178-182.

Kang, F., Li, J., and Xu, Q., 2009, "Structural Inverse Analysis by Hybrid Simplex Artificial Bee Colony Algorithms", **Computers and Structures**, Vol. 87, pp. 861-870.

Kao, Y.-T., Zahara, E., and Kao, I.-W., 2008, "A Hybridized Approach to Data Clustering", **Expert Systems with Applications**, Vol. 34, pp. 1754-1762.

Karaboga, D., 2005, **An Idea Based on Honey Bee Swarm for Numerical Optimization**, Technical Report-TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, Turkey.

Karaboga, D., **Artificial Bee Colony (ABC) Algorithm Homepage** [Online], Available: http://mf.erciyes.edu.tr/abc [2009, November 25].

Karaboga, D., and Akay, B., 2009a, "A Comparative Study of Artificial Bee Colony Algorithm", **Applied Mathematics and Computation**, Vol. 214, pp. 108-132.

Karaboga, D., and Akay, B., 2009b, "A Survey: Algorithms Simulating Bee Swarm Intelligence", **Artificial Intelligence Review**, Vol. 31, pp. 61-85.

Karaboga, D., Akay, B., and Ozturk, C., 2007, "Artificial Bee Colony (ABC) Optimization Algorithm for Training Feed-Forward Neural Networks", **LNCS: Modeling Decisions for Artificial Intelligence**, Springer-Verlag, Vol. 4617, pp. 318-329.

Karaboga, D., and Basturk, B., 2007, "A Powerful and Efficient Algorithm for Numerical Function Optimization: Artificial Bee Colony (ABC) Algorithm", **Journal of Global Optimization**, Vol. 39, pp. 459-471.

Karaboga, D., and Basturk, B., 2008, "On the Performance of Artificial Bee Colony (ABC) Algorithm", **Applied Soft Computing**, Vol. 8, pp. 687-697.

Karaboga, D., and Ozturk, C., 2011, "A Novel Clustering Approach: Artificial Bee Colony (ABC) Algorithm", **Applied Soft Computing**, Vol. 11, pp. 652-657.

Karaboga, N., 2009, "A New Design Method Based on Artificial Bee Colony Algorithm for Digital IIR Filters", **Journal of the Franklin Institute**, Vol. 346, pp. 328-348.

Kellegoz, T., and Toklu, B., 2008, "Comparing Efficiencies of Genetic Crossover Operations for One Machine Total Weighted Tardiness Problem", **Applied Mathematics and Computation**, Vol. 199, pp. 590-598.

Kennedy, J., and Eberhart, R.C., 1995, "Particle Swarm Optimization", **In: Proceedings of the 1995 IEEE International Conference on Neural Networks**, Vol. 4, pp. 1942-1948.

Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P., 1983, "Optimization by Simulated Annealing", **Science**, Vol. 220, pp. 671-680.

Lawrence, S., 1984, **Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques (Supplement)**, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh.

Li, B., and Wada, K., 2005, "Parallelizing Particle Swarm Optimization", **IEEE Pacific Rim Conference on Communications, Computers and Signal Processing**, pp. 288-291.

Li, B., Zheng, C.-H., Huang, D.-S., Zhang, L., and Han, K., 2010, "Gene Expression Data Classification Using Locally Linear Discriminant Embedding", **Computer in Biology and Medicine**, Vol. 40, pp. 802-810.

Lin, T.-L., Horng, S.-J., Kao, T.-W., Chen, Y.-H., Run, R.-S., Chen, R.-J., Lai, J.-L., and Kuo, I.-H., 2010, "An Efficient Job-Shop Scheduling Algorithm Based on Particle Swarm Optimization", **Expert Systems with Applications**, Vol. 37, pp. 2629-2636.

Liu, F., Qi, Y., Xia, Z., and Hao, H., 2009, "A Discrete Differential Evolution Algorithm for the Job Shop Scheduling Problem", **In: Proceedings of the First ACM/SIGEVO Summit on Genetic and Evolutionary Computation**, pp. 879-882.

Lourenc, H.R., Martin O., and Stützle, T., 2002, "Iterated Local Search", **In Handbook of Metaheuristics, International Series in Operations Research & Management Science**, Glover, F., and Kochenberger, G., (Eds.), Kluwer Academic Publishers, Massachusetts, Vol. 57, pp. 321-353.

Macintyre, G., Bailey, J., Gustafsson, D., Haviv, I., and Kowalczyk, A., 2010, "Using Gene Ontology Annotations in Exploratory Microarray Clustering to Understand Cancer Etiology", **Pattern Recognition Letters**, Vol. 31, pp. 2138-2146.

Maes, F., Collignon, A., Vandermeulen, D., Marchal, G., and Suetens, P., 1997, "Multimodality Image Registration by Maximization of Mutual Information", **IEEE Transactions on Medical Imaging,** Vol. 16, pp. 187-198.

Mount, D.M., Netanyahu, N.S., and LeMoigne, J., 1999, "Efficient Algorithms for Robust Point Pattern Matching and Applications to Image Registration", **Pattern Recognition**, Vol. 32, pp. 17-38.

Narasimhan, H., 2009, "Parallel Artificial Bee Colony (PABC) Algorithm", **World Congress on Nature & Biologically Inspired Computing**, pp. 306-311.

Niknam, T., and Amiri, B., 2010, "An Efficient Hybrid Approach Based on PSO, ACO and K-Means for Cluster Analysis", **Applied Soft Computing**, Vol. 10, pp. 183-197.

Niknam, T., Olamaie, J., and Amiri, B., 2008, "A Hybrid Evolutionary Algorithm Based on ACO and SA for Cluster Analysis", **Journal of Applied Science**, Vol. 8, pp. 2695-2702.

Nonsiri, S., and Supratid, S., 2008, "Modifying Ant Colony Optimization", **In: Proceedings of IEEE Conference on Soft Computing in Industrial Applications**, pp. 95-100.

Nutt, C.L., Mani, D.R., Betensky, R.A., Tamayo, P., Cairncross, J.G., and Ladd, C., 2003, "Gene Expression-based Classification of Malignant Gliomas Correlates better with Survival than Histological Classification", **Cancer Res**, Vol. 63, pp. 1602-1607.

Pan, Q.-K., Tasgetiren, M.F., Suganthan, P.N., and Chua, T.J., 2011, "A Discrete Artificial Bee Colony Algorithm for the Lot-streaming Flow Shop Scheduling Problem", **Information Sciences**, Vol. 181, pp. 2455-2468.

Perez, R.E., and Behdinan, K., 2007, "Particle Swarm Approach for Structural Design Optimization", **Computers and Structures**, Vol. 85, pp. 1579-1588.

Pham, D.T., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S., and Zaidi, M., 2006, "The Bees Algorithm, a Novel Tool for Complex Optimisation Problems", **In Proceedings of 2nd International Conference on Intelligent Production Machines and Systems (IPROMS 2006)**, Oxford: Elsevier, pp. 454-459.

Pochet, N., Smet, F.D., Suykens, J.A.K., and Moor, B.L.R.D., 2004, "Systematic Benchmarking of Microarray Data Classification: Assessing the Role of Non-linearity and Dimensionality Reduction", **Bioinformatics**, Vol. 20, pp. 3185-3195.

Quinn, M.J, 2004, **Parallel Programming in C with MPI and OpenMP**, Mc Graw Hill, New York, USA.

Rao, R. S., Narasimham, S.V.L., and Ramalingaraju, M., 2008, "Optimization of Distribution Network Configuration for Loss Reduction Using Artificial Bee Colony Algorithm", **International Journal of Electrical Power and Energy Systems Engineering**, Vol. 1, pp. 708-714.

Ratanasanya, S., Mount, D.M., Netanyahu, N.S., and Achalakul, T., 2008, "Enhancment in Robust Feature Matching", **In: Proceedings of ECTI-CON**, pp. 505-508.

Reinelt, G., 1991, "TSPLIB - A Traveling Salesman Problem Library", **ORSA Journal on Computing**, Vol. 3, pp. 376-384.

Sabat, S.L., Udgata, S.K., and Abraham, A., 2010, "Artificial Bee Colony Algorithm for Small Signal Model Parameter Extraction of MESFET", **Engineering Applications of Artificial Intelligence**, Vol. 23, pp. 689-694.

Selim, S.Z., and Alsultan, K., 1991, "A Simulated Annealing Algorithm for the Clustering Problem", **Pattern Recognition**, Vol. 24, pp. 1003-1008.

Sengoku, H., and Yoshihara, I., 1998, "A Fast TSP Solver Using GA on JAVA", **In Proceedings of 3rd International Symposium on Artificial Life and Robotics**, Vol. 1, pp. 283-288.

Shelokar, P.S., Jayaraman, V.K., and Kulkarni, B.D., 2004, "An Ant Colony Approach for Clustering", **Analytica Chimica Acta**, Vol. 509, pp. 187-195.

Shi, X.H., Liang, Y.C., Lee, H.P., Lu, C., and Wang, Q.X., 2007, "Particle Swarm Optimization-based Algorithms for TSP and Generalized TSP", **Information Processing Letters**, Vol. 103, pp. 169-176.

Silvaa, C.A., Sousaa, J.M.C., and Runkler, T.A., 2008, "Rescheduling and Optimization of Logistic Processes Using GA and ACO", **Engineering Applications of Artificial Intelligence**, Vol. 21, pp. 343-352.

Singh, A., 2009, "An Artificial Bee Colony Algorithm for the Leaf-constrained Minimum Spanning Tree Problem", **Applied Soft Computing**, Vol. 9, pp. 625-631.

Singh, D., Febbo, P.G., Ross, K., Jackson, D.G., Manola, J., Ladd, C., Tamayo, P., Renshaw, A.A., D'Amico, A.V., Richie, J.P., Lander, E.S., Loda, M., Kantoff, P.W., Golub, T.R., and Sellers, W.R., 2002, "Gene Expression Correlates of Clinical Prostate Cancer Behavior", **Cancer Cell**, Vol. 1, pp. 203-209.

Statnikov, A., Aliferis, C.F., and Tsamardinos, I., 2005, **Gene Expression Model Selector** [Online], Discovery System Laboratory, Department of Biomedical Informatics, Vanderbilt University, USA, Available: http://www.gems-system.org/ [2011, August 12].

Sundareswaran, K., and Sreedevi, V.T., 2008, "Development of Novel Optimization Procedure Based on Honey Bee Foraging Behavior", **In: Proceedings of IEEE International Conference on Systems, Man and Cybernetics**, pp. 1220-1225.

Sung, C.S., and Jin, H.W., 2000, "A Tabu-Search-Based Heuristic for Clustering", **Pattern Recognition**, Vol. 33, pp. 849-858.

Taetragool, U., and Achalakul, T., 2011, "Method for Failure Pattern Analysis in Disk Drive Manufacturing", **International Journal of Computer Integrated Manufacturing**, Vol. 24, pp. 834-846.

Tan, K.S., and Isa, N.A.M., 2011, "Color Image Segmentation Using Histogram Thresholding - Fuzzy C-means Hybrid Approach", **Pattern Recognition**, Vol. 44, pp. 1-15.

Tasgetiren, M.F., Sevkli, M., Liang, Y.-C., and Yenisey, M.M., 2006, "A Particle Swarm Optimization and Differential Evolution Algorithms for Job Shop Scheduling Problem", **International Journal of Operations Research**, Vol. 3, pp. 120-135.

Tsai, J.-T., Liu, T.-K., Ho, W.-H., and Chou, J.-H., 2008, "An Improved Genetic Algorithm for Job-shop Scheduling Problems Using Taguchi-based Crossover", **The International Journal of Advanced Manufacturing Technology**, Vol. 38, pp. 987-994.

Viola, P., and Wells, W.M., 1997, "Alignment by Maximization of Mutual Information", **International Journal of Computer Vision**, Vol. 24, pp.137-154.

Watanabe, M., Ida, K., and Gen, M., 2005, "A Genetic Algorithm with Modified Crossover Operator and Search Area Adaptation for the Job-Shop Scheduling Problem", **Computers & Industrial Engineering**, Vol. 48, pp. 743-752.

Wong, L.-P., Hean Low, M.Y., and Chong, C.S., 2008, "A Bee Colony Optimization Algorithm for Traveling Salesman Problem", **In: Proceedings of Second Asia International Conference on Modelling & Simulation**, pp. 818-823.

Xu, C., and Duan, H., 2010, "Artificial Bee Colony (ABC) Optimized Edge Potential Function (EPF) Approach to Target Recognition for Low-altitude Aircraft", **Pattern Recognition Letters**, Vol. 31, pp. 1759-1772.

Yagmahan, B., and Yenisey, M.M., 2008, "Ant Colony Optimization for Multi-Objective Flow Shop Scheduling Problem", **Computers & Industrial Engineering**, Vol. 54, pp. 411-420.

Yamada, T., and Nakano, R., 1992, "A Genetic Algorithm Applicable to Large Scale Job Shop Problems", **In Proceeding of the Second International Workshop on Parallel Problem Solving from Nature**, Manner, R., and Manderick, B., (Eds.), Vol. 2, Elsevier, Amsterdam, pp. 281-290.

Yang, F., Sun, T., and Zhang, C., 2009, "An Efficient Hybrid Data Clustering Method Based on K-Harmonic Means and Particle Swarm Optimization", **Expert Systems with Applications**, Vol. 36, pp. 9847-9852.

Yang, J.-H., Sun, L., Lee, H.P., Qian, Y., and Liang, Y.-C., 2008, "Clonal Selection Based Memetic Algorithm for Job Shop Scheduling Problems", **Journal of Bionic Engineering**, Vol. 5, pp. 111-119.

Yang, X.S., 2005, "Engineering Optimizations via Nature-inspired Virtual Bee Algorithms", **In: Lecture Notes in Computer Science**, Springer (GmbH), pp. 317-323.

Yang, X.S., 2008, **Introduction to Computational Mathematics**, World Scientific Publishing, New Jersey.

Yang, Y., and Kamel, M.S., 2006, "An Aggregated Clustering Approach Using Multi-Ant Colonies Algorithms", **Pattern Recognition**, Vol. 39, pp. 1278-1289.

Yao, B., Yang, C., Hu, J., Yin, G., and Yu, B., 2010, "An Improved Artificial Bee Colony Algorithm for Job Shop Problem", **Applied Mechanics and Materials**, Vol. 26-28, pp. 657-660.

Yin, P.Y., 2006, "Particle Swarm Optimization for Point Pattern Matching", **Journal of Visual Communication and Image Representation**, Vol. 17, pp. 143-162.

Yuan, T., and Kuo, W., 2008, "Spatial Defect Pattern Recognition on Semiconductor Wafers Using Model-Based Clustering and Bayesian Inference", **European Journal of Operational Research**, Vol. 190, pp. 228-240.

Yun, C.-H., Chuang, K.-T., and Chen, M.-S., 2006, "Adherence Clustering: An Efficient Method for Mining Market-Basket Clusters", **Information Systems**, Vol. 31, pp. 170-186.

Yussof, S., Razali, R.A., See, O.H., Ghapar, A.A., and Din, M.M., 2009, "A Coarse-Grained Parallel Genetic Algorithm with Migration for Shortest Path Routing Problem", **In: Proceedings of IEEE International Conference on High Performance Computing and Communications**, pp. 615-621.

Zhang, B., Hsu, M., and Dayal, U., 1999, **K-harmonic means - A Data Clustering Algorithm**, Technical Report HPL-1999-124, Hewlett-Packard Laboratories.

Zhang, B., Hsu, M., and Dayal, U. 2000. "K-Harmonic Means", **In: Proceedings of International Workshop on Temporal, Spatial and Spatio-Temporal Data Mining**, TSDM2000, Lyon, France, September 12.

Zhang, C., Ouyang, D., and Ning, J., 2010, "An Artificial Bee Colony Approach for Clustering", **Expert Systems with Applications**, Vol. 37, pp. 4761-4767.

Zhong, W.-L., Zhang, J., and Chen, W.-N., 2007, "A Novel Discrete Particle Swarm Optimization to Solve Traveling Salesman Problem", **In: Proceedings of IEEE Congress on Evolutionary Computation (CEC)**, pp. 3283-3287.

APPENDIX A

EXPERIMENTAL RESULTS FROM
ABC WITH THE TRAVELING SALESMAN PROBLEM (TSP)

Experiment Setting and Results

To verify the validity of our ABC-GSX algorithm on TSP, we benchmarked our results against those reported in Shi et al. (2007), Zhong et al. (2007), Nonsiri and Supratid (2008), and Wong et al. (2008), and instances from the TSP library (Reinelt, 1991). All experiments in this paper were performed on a PC with a 2.83-GHz Intel Core 2 Quad CPU and 4 GB of memory.

A fair comparison in terms of computational efficiency would be difficult. However, we tried to compare our proposed method to other aforementioned methods based on the same parameter settings (e.g. the number of populations and the number of iterations). We thus conducted two separated experiments.

In the first experiment, we compared our proposed method to PSO (Shi et al., 2007) and C3DPSO (Zhong et al., 2007). The number of employed and onlooker bees were 15 and the MCN was $N * 10$ where N was the number of cities, matching the parameters used in the experiment on C3DPSO reported in Zhong et al. (2007).

In the second experiment, we compared our proposed method to ACO-PSO (Nonsiri and Supratid, 2008) and BCO (Wong et al., 2008). The number of employed and onlooker bees were 50. The MCN was 2000. Note that the number of populations used by ACO-PSO and BCO experiments reported in Nonsiri and Supratid (2008) and Wong et al. (2008) were $N * 2$ and N respectively, for the number of iterations shown in Table A.1.

Each problem instance in each experiment ran for 100 times.

Table A.1 Number of iterations used in ABC-GSX, ACO-PSO and BCO algorithms

Problem	Number of iteration usage		
	ABC-GSX	ACO-PSO	BCO
EIL51	2000	n/a	50000
BERLIN52	2000	2000	n/a
EIL76	2000	n/a	50000
KROA100	2000	3500	50000
KROB100	2000	n/a	50000
CH150	2000	4000	n/a
KROB200	2000	n/a	50000
LIN318	2000	n/a	50000

"n/a" stands for "not available".

The numerical results from the first and second experiments are shown in Table A.2 and Table A.3 respectively. The problem size ranged from 51 to 100 cities in the first experiment and from 51 to 318 cities in the second experiment. The “Optimal” figures represent the best known optimal tour cost in each problem instance. The “Best” and the “Err” figures show, respectively, the best result and the relative error obtained from each algorithm.

The relative error is calculated using

Error = [(Average - Optimal) / Optimal] x 100 %

Note that lower "Best" and "Error" values indicate better solutions.

Table A.2 Results of the first TSP experiment

Problem	Optimal	ABC-GSX		PSO [17]		C3DPSO [9]	
		Best	*Err (%)*	*Best*	*Err (%)*	*Best*	*Err (%)*
EIL51	426	426	0.9413	427	2.5751	427	1.7930
BERLIN52	7542	7542	0.0301	7542	3.8458	7542	0.7530
ST70	675	675	0.7126	675	3.3422	n/a	n/a
EIL76	538	538	2.4145	546	4.1673	540	2.5500
PR76	108159	108159	0.4614	108280	3.8176	n/a	n/a
KROA100	21282	21282	0.1988	n/a	n/a	21296	1.9140

“n/a” stands for “not available”.

Table A.3 Results of the second TSP experiment

Problem	Optimal	ABC-GSX		ACO-PSO [8]		BCO [10]	
		Best	*Err (%)*	*Best*	*Err (%)*	*Best*	*Err (%)*
EIL51	426	426	0.1596	n/a	n/a	428	0.8500
BERLIN52	7542	7542	0	7544	0.0300	n/a	n/a
EIL76	538	538	0.7305	n/a	n/a	538	2.0100
KROA100	21282	21282	0	22238	4.5000	21762	3.4300
KROB100	22141	22141	0.0303	n/a	n/a	22636	3.1000
CH150	6528	6533	1.1631	6689	2.4700	n/a	n/a
KROB200	29437	29438	1.6275	n/a	n/a	30349	6.3600
LIN318	41345	42215	4.3478	n/a	n/a	44685	7.5500

“n/a” stands for “not available”.

It can be drawn that using ABC-GSX on TSP has produced, on average, nearly optimal results in each problem instance. Our proposed approach outperformed all other aforementioned approaches. Average and best-case solution qualities of ABC-GSX were always as good as or better than of those obtained with other approaches. Maximum relative error never exceeded 2% except for the LIN318 problem instance and average relative error was less than 1%. Average improvements on the average results of ABC-GSX method compared to PSO, C3DPSO, ACO-PSO, and BCO across

all experiments were 2.54%, 0.84%, 1.87%, and 2.61% respectively. More importantly, ABC-GSX managed to find globally optimal solutions on most problem instances. ABC-GSX also converged substantially faster with a much smaller number of iterations needed when we focus on the number of iteration usage setting in Table 1. Average improvement on the number of iteration usage for ABC-GSX compared to ACO-PSO and BCO method were 16% and 96% respectively.

APPENDIX B

EXPERIMENTAL RESULTS FROM
ABC WITH DIMENSION REDUCTION IN BIOINFORMATICS DATA

Experiment Setting and Results

In order to investigate the performance of ABC-kNN's search capability on feature selection problems, we validated our proposed technique in comparison to other state-of-the-art algorithms used in bioinformatics, which include a hybrid improved binary particle swarm optimization with k-nearest neighbor method (IBPSO-kNN) (Chuang et al., 2008), a least squares support vector machine (LS-SVM) (Pochet et al., 2004), a principal component analysis with Fisher discriminant analysis (PCA-FDA) (Pochet et al., 2004), a multi-step dimensionality reduction learning with local and global consistency (MSDR-LGC) (Gui et al., 2010), and a locally linear discriminant embedding with k-nearest neighbor method (LLDE-kNN) (Li et al., 2010). The performance of the algorithms is evaluated based on 16 well-known datasets taken from Alon et al. (1999), Golub et al. (1999), Singh et al. (2002), Iizuka et al. (2003), Nutt et al. (2003), and Statnikov et al. (2005).

In our experiments, the parameter settings for ABC-kNN algorithm are as follows: The number of employed and onlooker bees are set to 12. The maximum number of iterations is 100, which is the same stopping criteria as used each in aforementioned literature (Chuang et al., 2008). The predetermined number of iterations for avoiding the local optima solutions is set to 10.

Our algorithm was coded in C++ and run on a PC with a 2.83-GHz Intel Core 2 Quad CPU and 4 GB of memory.

The 16 benchmark datasets used in the experiments include Colon_Cancer, Acute_Leukemia, Hepatocellular_Carcinoma, High-grade_Glioma, Prostate_Cancer, 9_Tumors, 11_Tumors, 14_Tumors, Brain_Tumor1, Brain_Tumor2, Leukemia1, Leukemai2, Lung_Cancer, SRBCT, Prostate_Tumor, and DLBCL. The problem size ranges from 2 to 26 distinct diagnostic categories, 50 to 308 samples (patients), and 2,000 to 15,009 variables (genes). The characteristics of these datasets can be summarized and shown in Table B.1. All these problems can be received as data mining problems. The researchers are trying to find specific sets of feature values that reliably predict the occurrence or severity of a disease.

Table B.1 The characteristics of data sets considered.

Dataset Number	Data Name	Number of		
		Samples	Categories	Genes
1	Colon_Cancer	62	2	2,000
2	Acute_Leukemia	72	2	7,129
3	Hepatocellular_Carcinoma	60	2	7,129
4	High-grade_Glioma	50	2	12,625
5	Prostate_Cancer	136	2	12,600
6	9_Tumors	60	9	5,726
7	11_Tumors	174	11	12,533
8	14_Tumors	308	26	15,009
9	Brain_Tumor1	90	5	5,920
10	Brain_Tumor2	50	4	10,367
11	Leukemia1	72	3	5,327
12	Leukemia2	72	3	11,225
13	Lung_Cencer	203	5	12,600
14	SRBCT	83	4	2,308
15	Prostate_Tumor	102	2	10,509
16	DLBCL	77	2	5,469

In order to achieve a fair comparison, we compare our proposed method to other aforementioned methods based on the same group of datasets reported in previous literatures (Chuang et al., 2008; Gui et al., 2010; Li et al., 2010). The quality of the results in each method is measured based on the value of classification accuracy. As a result, it is necessary to divide the experiments into two sets. For the first set, we compare our results against the results from LS-SVM, PCA-FDA, MSDR-LGC, LLDE-kNN using the first five datasets listing in Table B.1. The second set of experiments is run based on the rest of the datasets in Table B.1. The results of IBPSO-kNN and our proposed method are compared separately.

The best and the average of the classification accuracy over 50 runs are used to measure the performance of ABC-kNN. The boldface represents problems in which ABC-kNN provides a better solution quality than other approaches.

The results from the first set of experiments are provided in Table B.2.

From Table B.2, it can be seen that the ABC-kNN method generates better results than other methods in terms of the best and the average of classification accuracy. The best accuracy achieved by ABC-kNN is at least to 95% for all datasets. In the Acute_Leukemia dataset, 100 % classification accuracy was achieved even before the algorithm reached the maximum number of iterations as shown in Figure B.1.

Table B.2 Comparison results of the ABC-kNN and LS-SVM, PCA-FDA, MSDR-LGC, LLDE-kNN.

Data Name	LS-SVM		PCA-FDA		ABC-kNN	
	% Accuracy		% Accuracy		% Accuracy	
	Best	Average	Best	Average	Best	Average
Colon_Cancer	82.03	n/a	80.30	n/a	**98.39**	**96.35±0.01**
Acute_Leukemia	93.56	n/a	94.40	n/a	**100.00**	**100.00±0.00**
Hepatocellular_Carcinoma	68.61	n/a	69.49	n/a	**95.00**	**89.50±0.03**
High-grade_Glioma	69.95	n/a	68.72	n/a	**98.00**	**94.76±0.02**
Data Name	**MSDR-LGC**		**LLDE-kNN**		**ABC-kNN**	
	% Accuracy		% Accuracy		% Accuracy	
	Best	Average	Best	Average	Best	Average
Colon_Cancer	90.91	84.15±1.52	n/a	n/a	**98.39**	**96.35±0.01**
Acute_Leukemia	97.90	95.52±0.76	97.63	95.59±2.86	**100.00**	**100.00±0.00**
Hepatocellular_Carcinoma	75.26	69.97±2.06	n/a	n/a	**95.00**	**89.50±0.03**
High-grade_Glioma	77.90	71.38±1.90	80.72	75.17±7.76	**98.00**	**94.76±0.02**
Prostate_Cancer	n/a	n/a	94.92	92.65±3.18	**98.53**	**96.19±0.01**

n/a = No result reported

The improvement percentages of the ABC-kNN's best classification accuracy results over MSDR-LGC across Colon_Cancer, Acute_Leukemia, Hepatocellular_Carcinoma, and High-grade_Glioma datasets are 7.5%, 2.1%, 19.7%, and 20.1% respectively. Comparing with LLDE-kNN, the improvement percentages for Acute_Leukemia, High-grade_Glioma, and Prostate_Cancer datasets are 2.4%, 17.3%, and 3.6% respectively.

In addition, the results from Table B.2 indicate that ABC-kNN is more robust than the MSDR-LGC and the LLDE-kNN algorithms as shown by the lower standard deviations of accuracy across runs. The standard deviations obtained from ABC-kNN are less than or equal to 0.03 across all data sets.

The number of iterations versus genes selected and the percentage of classification accuracy for ABC-kNN on these datasets are illustrated in Figure B.2 and Figure B.3. As the number of iterations is increased, the number of genes selected is declines and the classification accuracy is improved. These results indicate that the classification accuracy can be maintained while the number of selected genes is reduced, i.e. not all features are needed in order to gain the total classification accuracy.

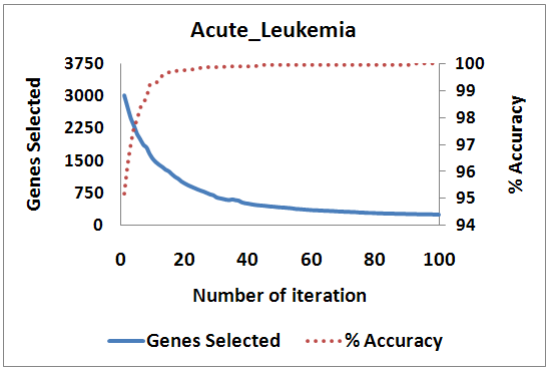
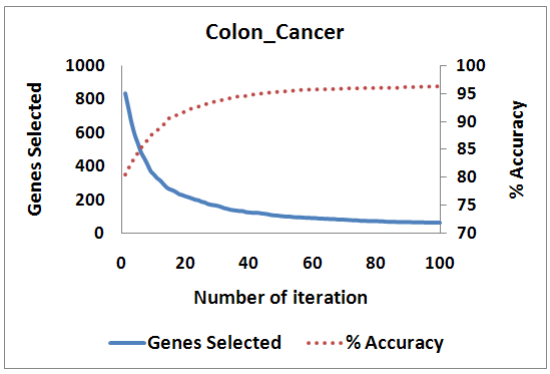


Figure B.1 The number of iterations vs. genes selected and classification accuracy on Acute_Leukemia dataset



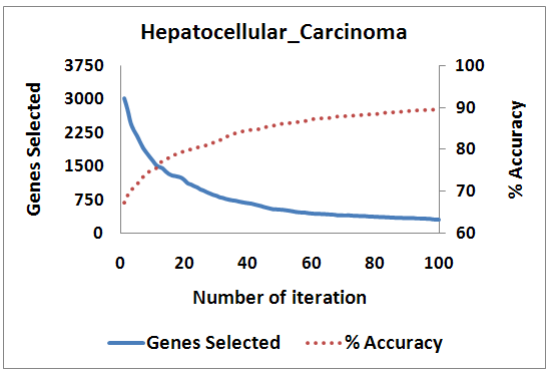
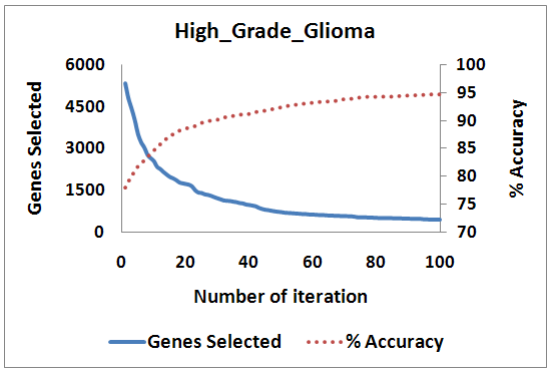


Figure B.2 The number of iterations vs. genes selected and classification accuracy on Colon_Cancer and Hepatocellular_Carcinoma datasets



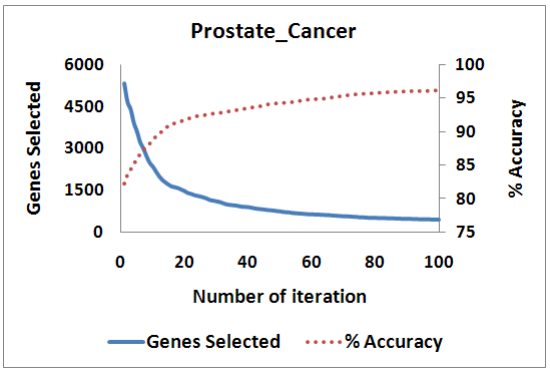


Figure B.3 The number of iterations vs. genes selected and classification accuracy on High_Grade_Glioma and Prostate_Cancer datasets

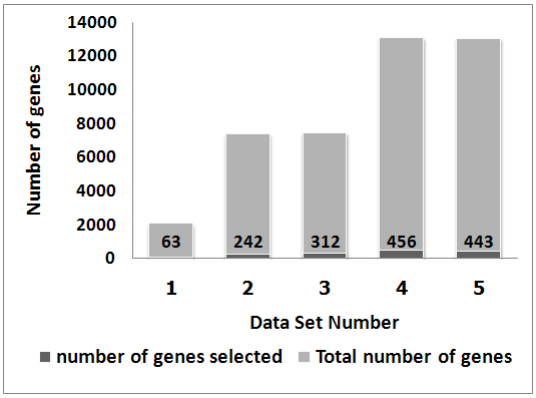


Figure B.4 Number of genes selected for each of the 5 datasets obtained from ABC-kNN

Figure B.4 illustates the number of genes selected obtained from ABC-kNN on these five datasets. The percentage of genes selected for Colon_Cancer, Acute_Leukemia, Hepatocellular_Carcinoma, High-grade_Glioma, and Prostate_Cancer datasets are reduced to 3.15%, 3.39%, 4.38%, 3.61%, and 3.59% of the total available respectively.

In the second set of experiments, we compared ABC-kNN with IBPSO-kNN. Eleven benchmark datasets are used (9_Tumors, 11_Tumors, 14_Tumors, Brain_Tumor1, Brain_Tumor2, Leukemia1, Leukemai2, Lung_Cancer, SRBCT, Prostate_Tumor, and DLBCL). The results are shown in Table B.3.

Table B.3 Comparison results of the ABC-kNN and IBPSO-kNN.

Data Name	IBPSO-kNN			ABC-kNN		
	Genes Selected	% Genes selected	% Accuracy	Genes Selected	% Genes selected	% Accuracy
9_Tumors	1280	22.35	78.33	**276**	**4.82**	**81.67**
11_Tumors	2948	23.52	93.10	**744**	**5.94**	**95.40**
14_Tumors	2777	18.50	66.56	**1105**	**7.36**	**68.18**
Brain_Tumor1	754	12.74	94.44	**294**	**4.97**	94.44
Brain_Tumor2	1197	11.55	94.00	**335**	**3.23**	**96.00**
Leukemia1	1034	19.41	100.00	**135**	**2.53**	100.00
Leukemia2	1292	11.51	100.00	**266**	**2.37**	100.00
Lung_Cencer	1897	15.06	96.55	**551**	**4.37**	**97.54**
SRBCT	431	18.67	100.00	**54**	**2.34**	100.00
Prostate_Tumor	1294	12.31	92.16	**210**	**2.00**	**98.04**
DLBCL	1042	19.05	100.00	**150**	**2.74**	100.00
Average		16.79	92.29		**3.88**	**93.75**

The results show that, ABC-kNN performs better than IBPSO-kNN in 6 out of 11 datasets, and equally well in another five datasets. The results also show that our method provide 100% classification accuracy for Leukemia1, Leukemia2, SRBCT, and

DLBCL datasets. Especially, the results in Figure B.5 and Figure B.6 depict that our proposed method obtains 100% classification accuracy on these datasets before reaching the maximum number of iterations.

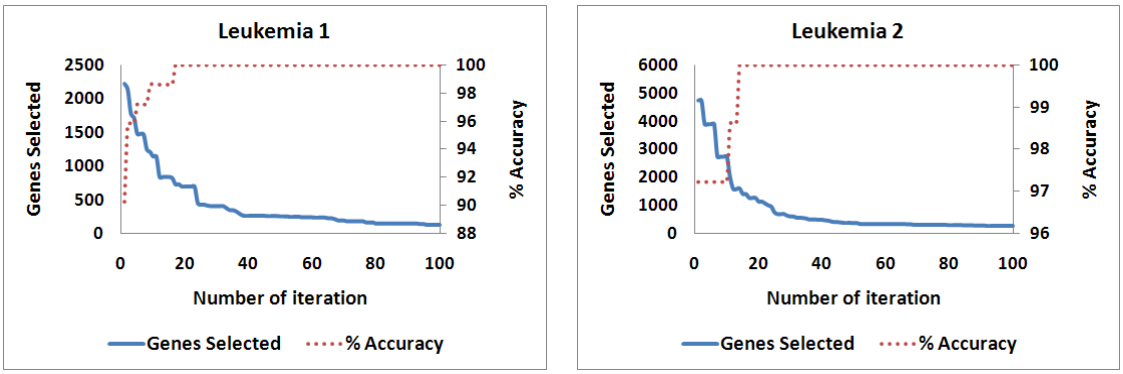


Figure B.5 The number of iterations vs. genes selected and classification accuracy on Leukemia1and Leukemia2 datasets

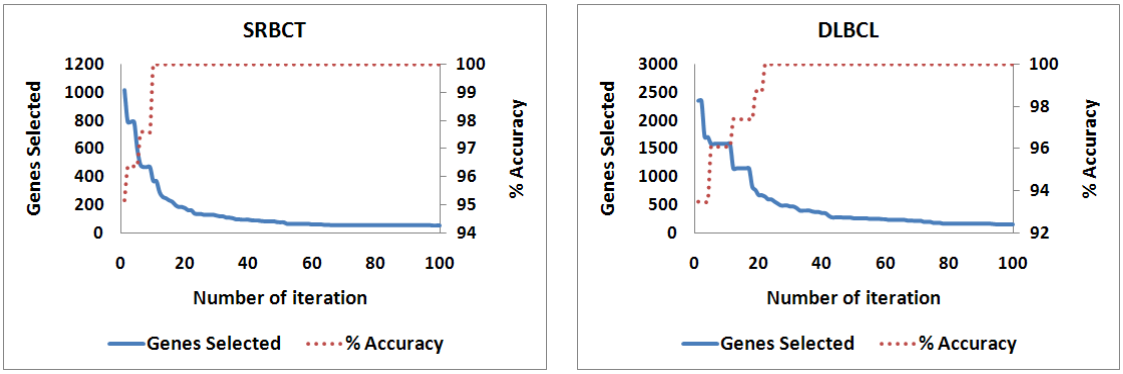


Figure B.6 The number of iterations vs. genes selected and classification accuracy on SRBCT and DLBCL datasets

According to Table B.3, ABC-kNN improves on the classification accuracy of ABC-kNN method in comparison to IBPSO-kNN method across 9_Tumors, 11_Tumors, 14_Tumors, Brain_Tumor2, Lung_Cancer, and Prostate_Tumor datasets are approximately 3.3%, 2.3%, 1.6%, 2.0%, 1.0%, and 5.9% respectively. The classification accuracy for Brain_Tumor1 dataset is 94.44% which is equal to the best value from IBPSO-kNN approach. The average classification accuracy for ABC-kNN is 93.75% whereas it is 92.29% for IBPSO-kNN.

The number of iterations versus genes selected and classification accuracy for ABC-kNN on these datasets are shown in Figure B.7 to Figure B.9. The plateau regions on these graphs also demonstrate that our ABC-kNN method helps avoid suboptimal solutions, which may occur sometimes before reaching the maximum number of iterations.

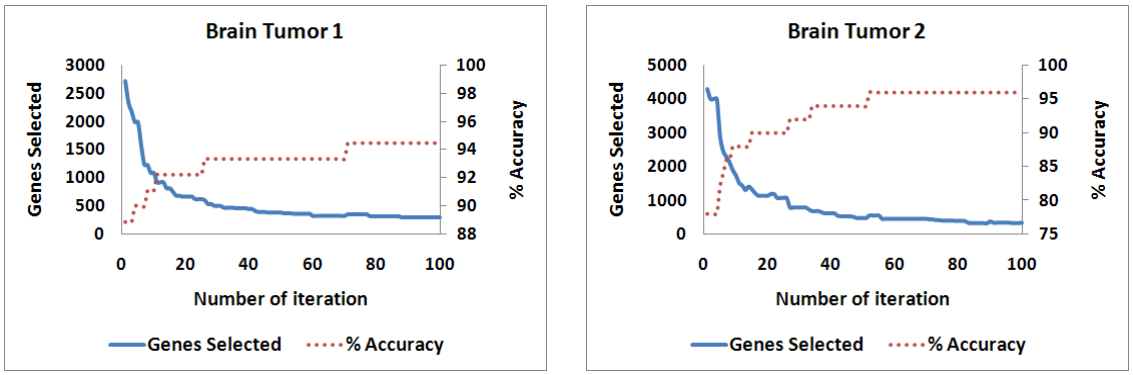


Figure B.7 The number of iterations vs. genes selected and classification accuracy on Brain_Tumor1 and Brain_Tumor2 datasets

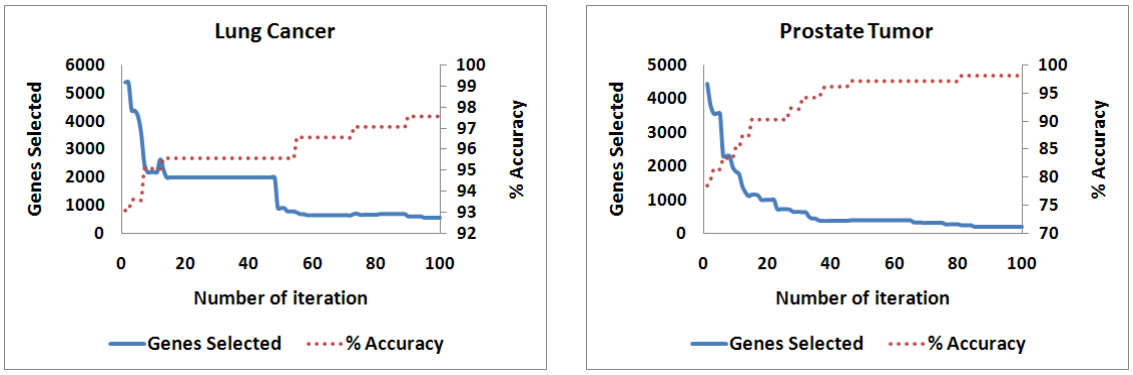


Figure B.8 The number of iterations vs. genes selected and classification accuracy on Lung_Cancer and Prostate_Tumor datasets

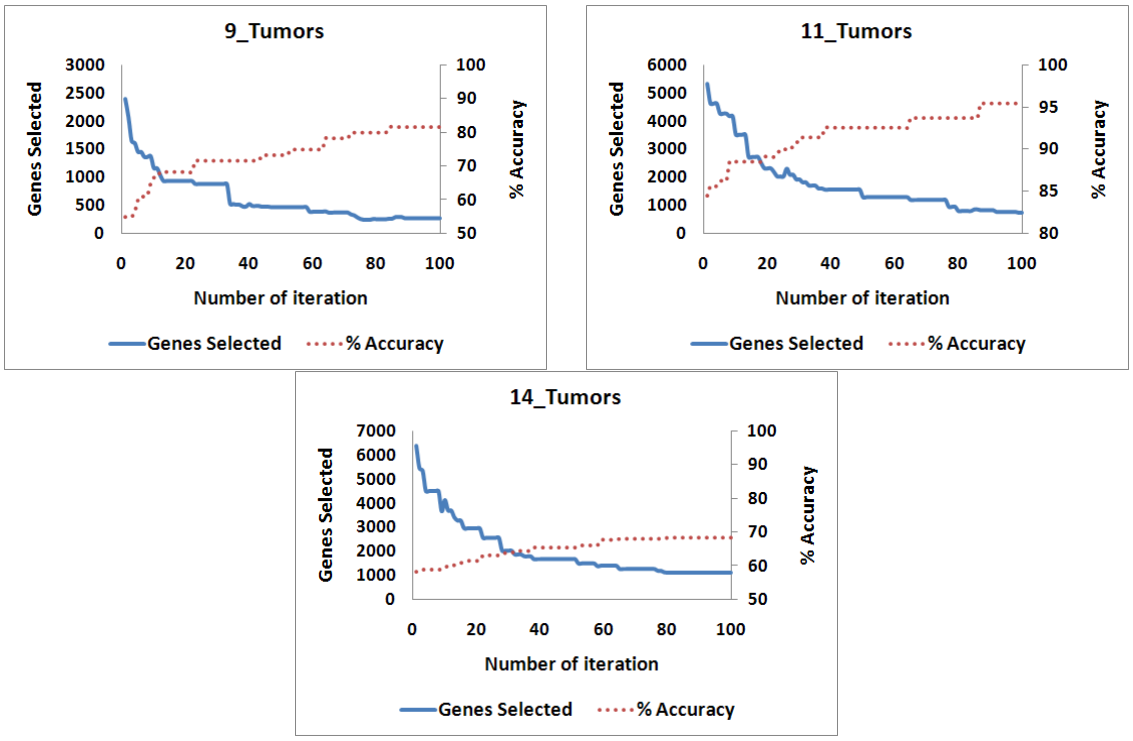


Figure B.9 The number of iterations vs. genes selected and classification accuracy on 9_Tumors, 11_Tumors, and 14_Tumors datasets

Figure B.10 illustrates that the percentage of genes selected in our approach is lower than that of the IBPSO-kNN approach in all problems. The average percentage for ABC-kNN is 3.88% whereas that from IBPSO-kNN is 16.79%. The highest percentage for ABC-kNN is 7.36% for 14_Tumors dataset, and the lowest percentage is 2.00% for Prostate_Tumor. Once again, it is clear that not all features are needed in order to achieve the total classification accuracy.

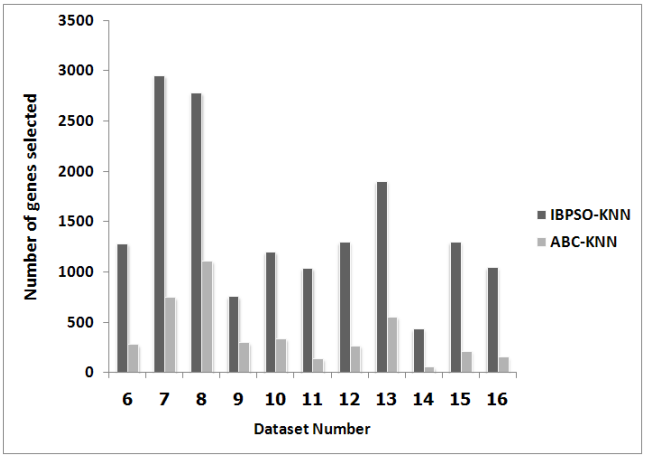


Figure B.10 Number of genes (features) selected for each of the 11 datasets obtained from ABC-kNN versus IBPSO-kNN

In summary, the experiments show that ABC-kNN can select an optimal subset as the representative of the original dataset with high accuracy even in the case of multi-classification problem like 9_Tumors and 11_Tumors datasets. The classification accuracy is also more than 90% except for 9_Tumors, 14_Tumors, and Hepatocellular_Carcinoma datasets. These three datasets either have a complex and nonlinear structure or a large number of classification categories, which may cause the degradation in the algorithm's performance. However, the search mechanism based on heuristic method of ABC-kNN can still explore the complex and nonlinear structure of data more effectively than other aforementioned approaches. The results show that even for these problems, ABC-kNN can produce up to 17% higher classification accuracy.

The results in the experiments provide evidence that our ABC-kNN method can produce better solutions than other state-of-the-art feature selection algorithms such as IBPSO-kNN, LS-SVM, PCA-FDA, MSDR-LGC, and LLDE-kNN in terms of both the number of features selected and the classification accuracy.

APPENDIX C

EXPERIMENTAL RESULTS FROM
THE EFFECT OF PROPOSED METHODS AND SENSITIVITY OF THE PARAMETERS SETTING IN BEST-SO-FAR ABC

The Effect of Proposed Methods in Best-so-far ABC

Experiment Setting and Results

In order to investigate the performance on proposed methods including the Best-so-far ABC method (BSF), the Adjustable Search Radius (ASR), and Objective-value-based Comparison Method (OBC), numerical benchmark functions as shown in Table 4.1 were used in this experiment. The objective of the search process is to find the solutions that can produce the minimum output value from these benchmark functions. The number of employed and onlooker bees were set to 50. The value of limit and the maximum numbers of iterations were set to 50 and 1000 respectively. Each of the experiments was repeated 20 times with different random seeds. All the experiments in this paper were run on the same hardware (Intel Core 2 Quad with 2.4 GHz CPU and 4 GB memory). The mean and standard deviation of the output values of benchmark functions were recorded and shown in Table C.1 and the iterations to convergence of the mean best values presented in Table C.1 have been illustrated in Figure C.1 to C.3.

Table C.1 Mean of Best Function Values from ABC with different proposed method.

Function	D	ABC		ABC with BSF		ABC with ASR		ABC with OBC	
		Mean	S.D.	Mean	S.D.	Mean	S.D.	Mean	S.D.
Sphere	30	3.32E-9	2.45E-9	2.33E-11	6.49E-11	4.04E-11	1.66E-10	2.76E-9	1.83E-9
Griewank	30	2.63E-6	1.10E-5	5.17E-19	6.00E-19	9.90E-9	2.03E-8	1.79E-8	3.77E-8
Rastrigin	30	0.04977	0.22247	3.55E-17	2.58E-17	1.39E-6	6.17E-7	0.00014	0.00053
Rosenbrock	30	4.43710	3.20420	0.01698	0.01801	1.69539	1.36531	2.71635	1.93521
Ackley	30	8.28E-6	2.95E-6	0	0	6.89E-6	1.76E-6	6.11E-6	2.06E-6
Schaffer	2	1.44E-5	3.92E-5	5.26E-17	3.91E-17	4.76E-17	3.26E-17	1.19E-6	1.76E-6

D, dimension of problems

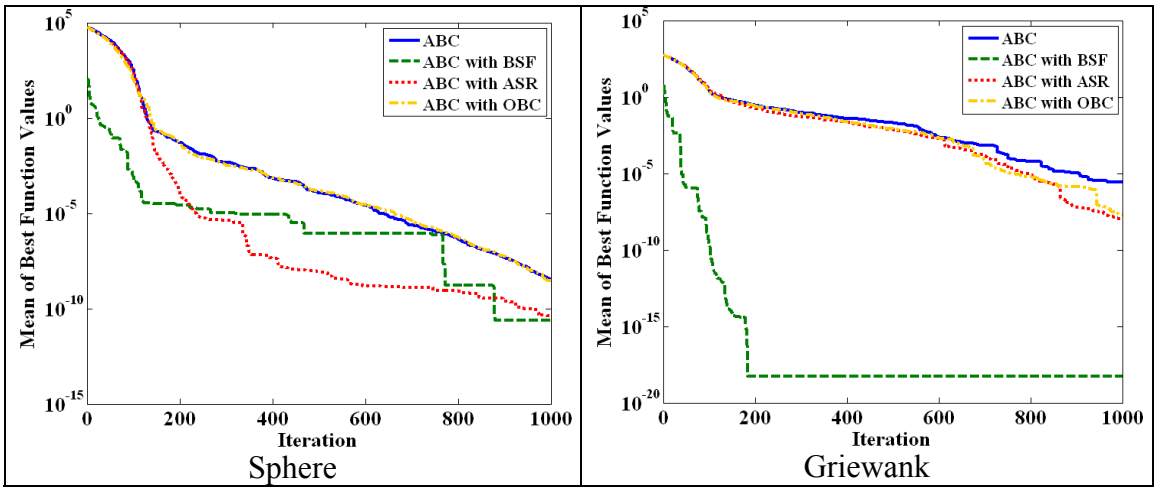


Figure C.1 The effect of proposed methods on Sphere and Griewank

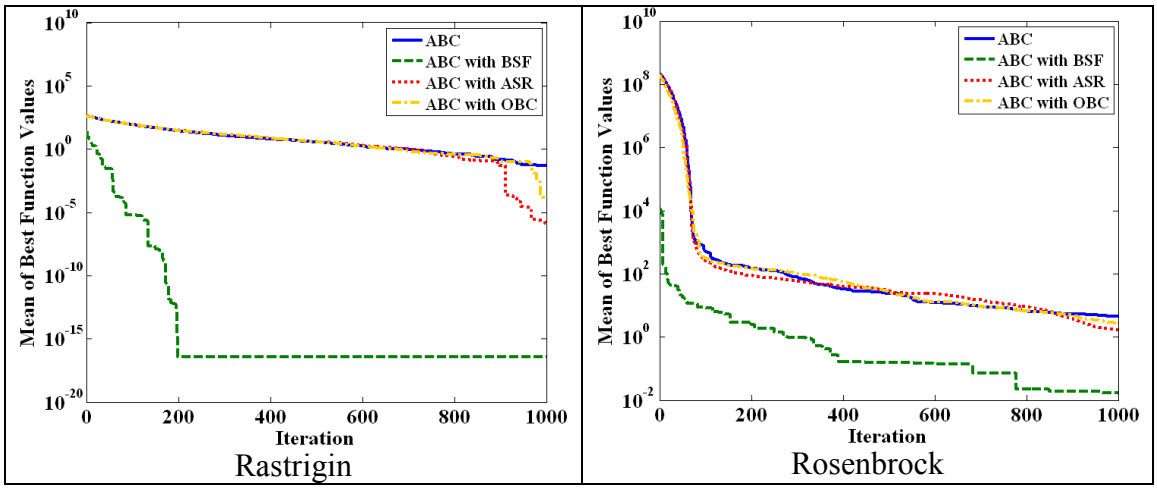


Figure C.2 The effect of proposed methods on Rastrigin and Rosenbrock

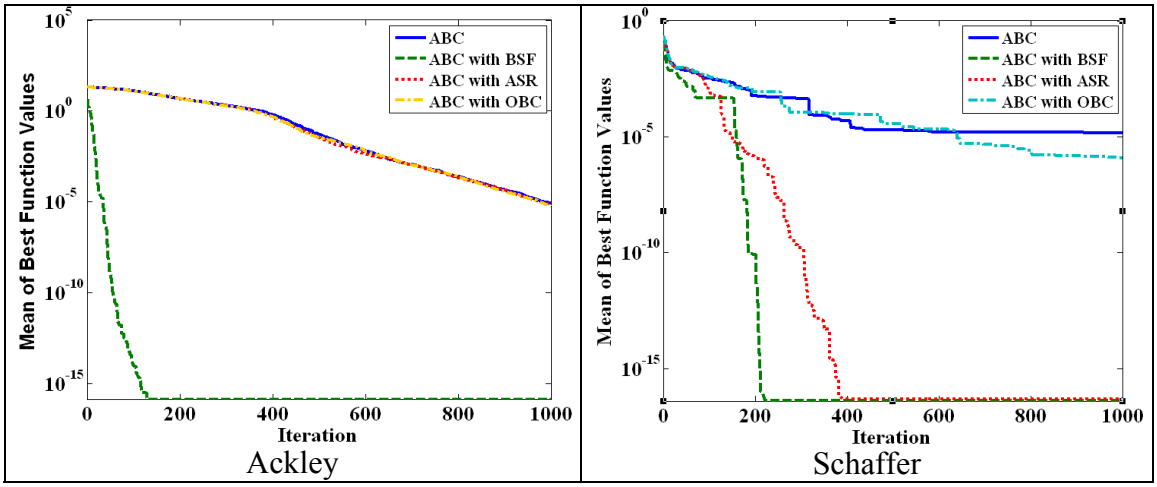


Figure C.3 The effect of proposed methods on Ackley and Schaffer

Sensitivity of the Parameters Setting in Best-so-far ABC

Experiment Setting and Results

To analyze the sensitivity of the parameters setting in Best-so-far ABC, two parameters including "colony size" and "limit" values were observed. This experiment aims to investigate the behavior of the Best-so-far ABC with different population sizes and limit values. The iterations to convergence of the mean best values presented in Table 4.3 have been illustrated in Figure C.4 to C.8.

As seen from Figure C.4 to C.8, the production of scout improves the search ability of our Best-so-far ABC. It has useful effect on the performance of the algorithm for all benchmark functions except the Rosenbrock function. Its benefit becomes much clearer for smaller colony sizes.

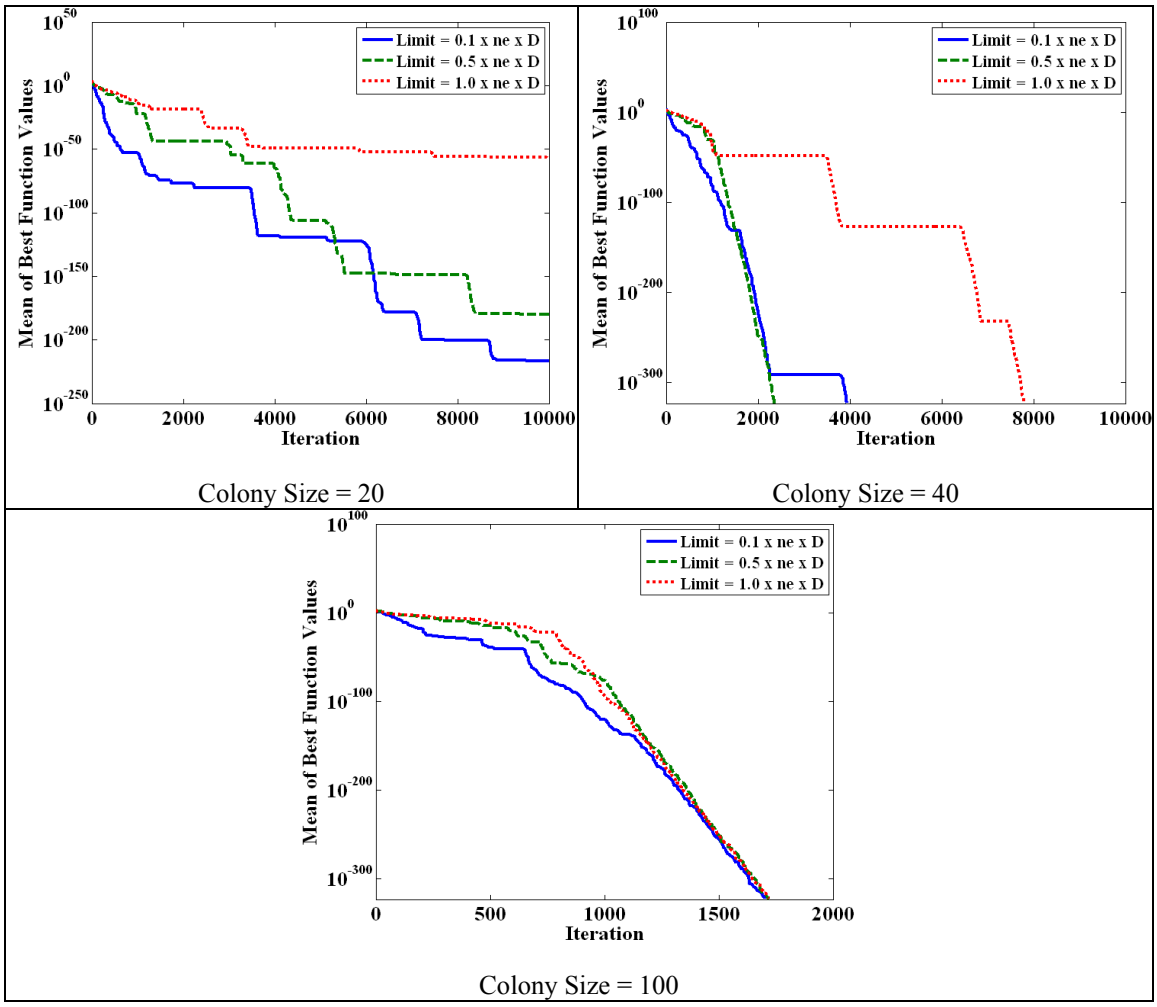


Figure C.4 Iterations to convergence under different "limit" values on Sphere

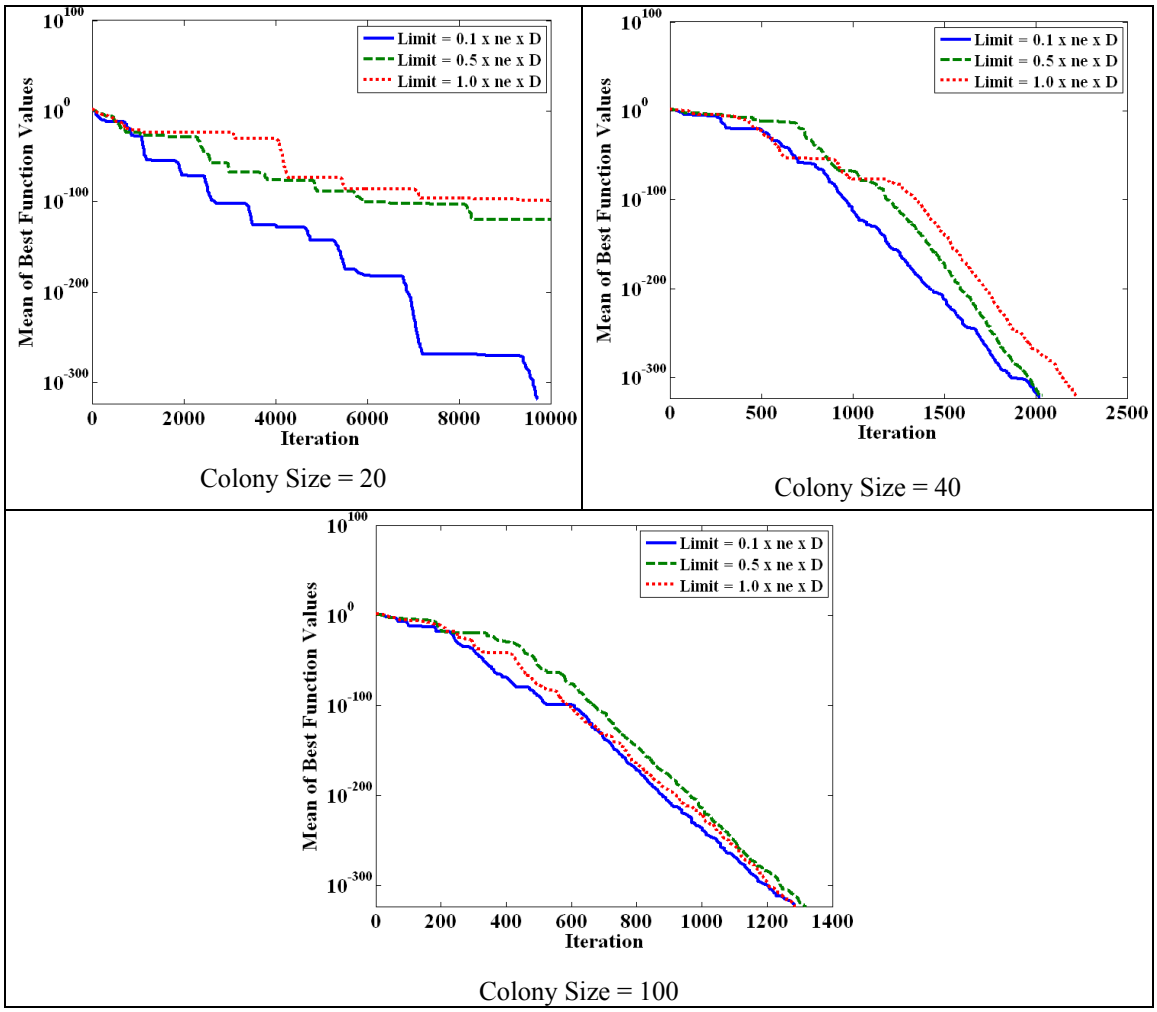


Figure C.5 Iterations to convergence under different "limit" values on Rastrigin

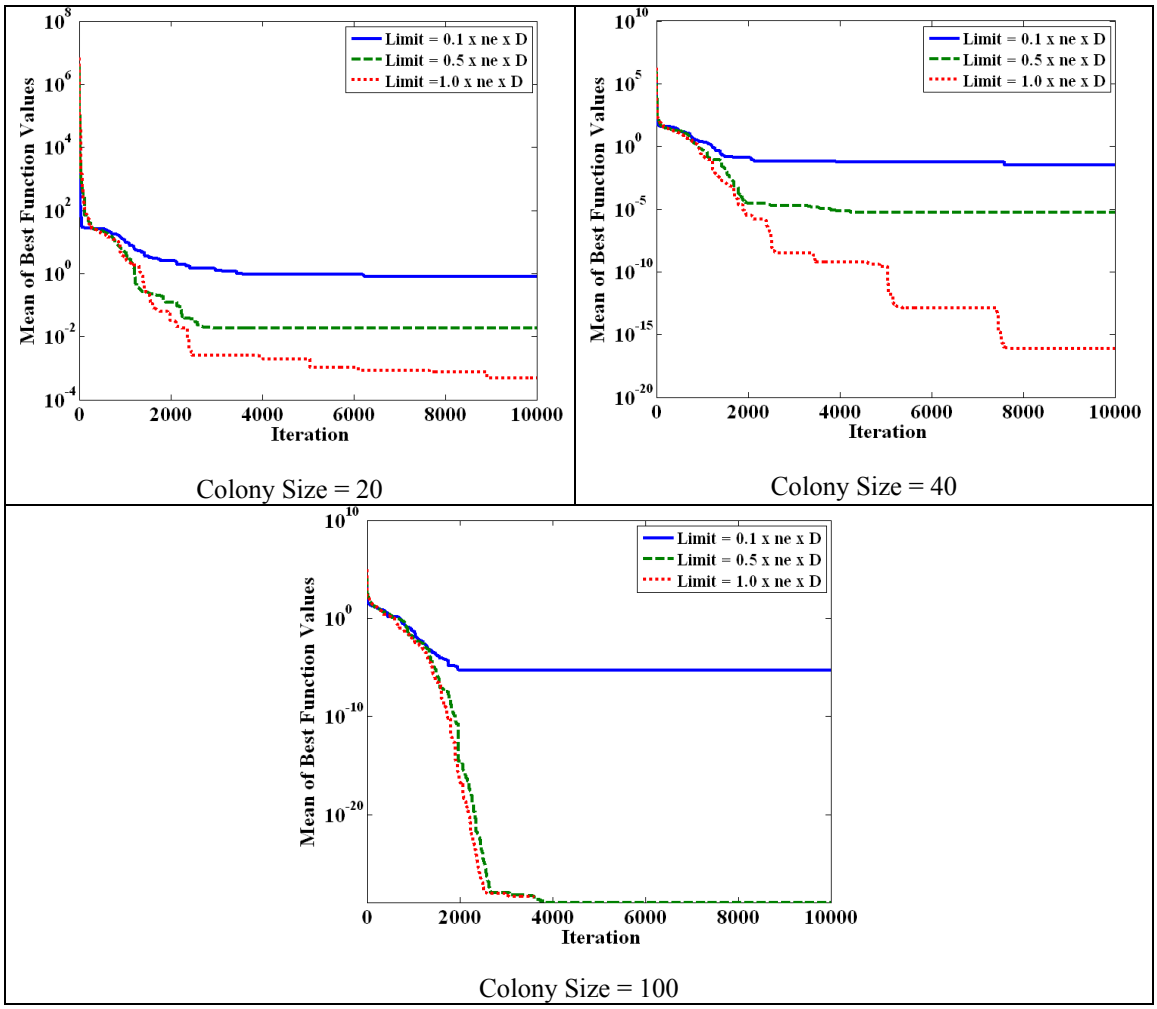


Figure C.6 Iterations to convergence under different "limit" values on Rosenbrock

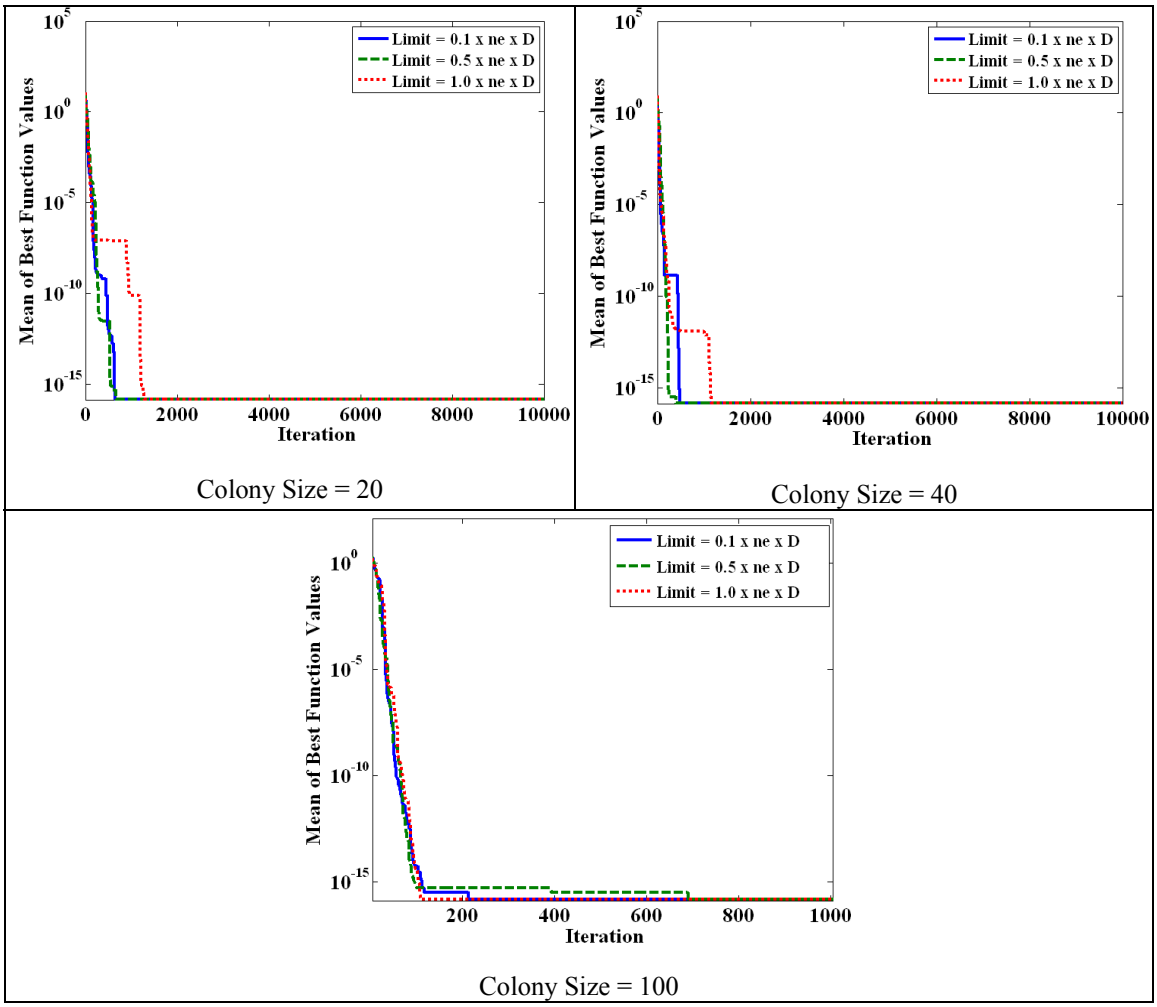


Figure C.7 Iterations to convergence under different "limit" values on Ackley

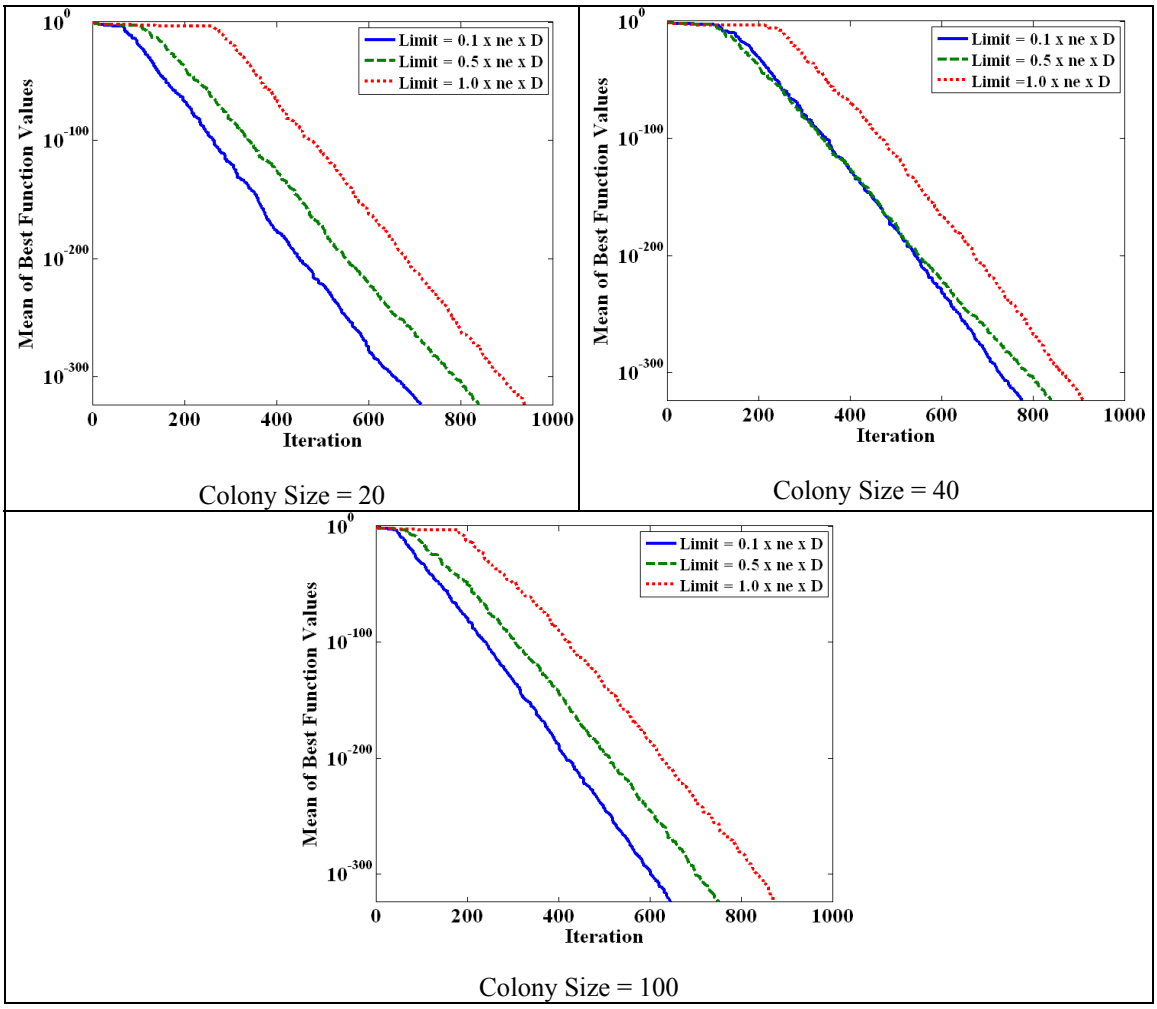


Figure C.8 Iterations to convergence under different "limit" values on Schaffer

CURRICULUM VITAE

NAME	Mr. Anan Banharnsakun
DATE OF BIRTH	25 November 1978
EDUCATIONAL RECORD	
HIGH SCHOOL	High School Graduation Wat Songtham School, 1996
BACHELOR'S DEGREE	Bachelor of Engineering (Computer Engineering) King Mongkut's University of Technology Thonburi, 2000
MASTER'S DEGREE	Master of Engineering (Computer Engineering) King Mongkut's University of Technology Thonburi, 2006
DOCTORAL DEGREE	Doctor of Philosophy (Electrical and Computer Engineering) King Mongkut's University of Technology Thonburi, 2011
EMPLOYMENT RECORD	System Programmer Kasikorn Bank, 2001-2004
	Advanced Technical Specialist IBM Solutions Delivery Co,Ltd., 2004-2009
	Teaching Assistant Computer Engineering Department, Faculty of Engineering, KMUTT, 2009-2010
SCHOLARSHIPS	King Mongkut's Diamond Scholarship The Royal Golden Jubilee PhD Scholarship Grant No. PHD/0038/2552
PUBLICATION	Banharnsakun, A., Achalakul, T., and Sirinaovakul, B., 2011, "The Best-so-far Selection in Artificial Bee Colony algorithm", **Applied Soft Computing,** Vol. 11, pp. 2888-2901.
	Banharnsakun, A., Sirinaovakul, B., and Achalakul, T., 2011, "Job Shop Scheduling with the Best-so-far ABC", **Engineering Applications of Artificial Intelligence,** doi:10.1016/j.engappai.2011.08.003.
	Banharnsakun, A., Sirinaovakul, B., and Achalakul, T., 2011, "The Best-so-far ABC with Multiple Patrilines for Clustering Problems", **Neurocomputing**. (Submitted)

Banharnsakun, A., Achalakul, T., and Sirinaovakul, B., 2010, "ABC-GSX: A Hybrid Method for Solving the Traveling Salesman Problem", **In: Proceedings of the World Congress on Nature and Biologically Inspired Computing (NaBIC2010)**, pp. 7-12.

Banharnsakun, A., Achalakul, T., and Sirinaovakul, B., 2010, "Artificial Bee Colony Algorithm on Distributed Environments", **In: Proceedings of the World Congress on Nature and Biologically Inspired Computing (NaBIC2010)**, pp. 13-18.

Prasartvit, T., Banharnsakun, A., Kaewkamnerdpong, B., and Achalakul, T., 2011, "Reducing Bioinformatics Data Dimension with ABC-kNN", **Neurocomputing**. (Submitted)