



MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES



**FRACTAL GEOMETRY INSPIRED
SOLUTION GENERATION TO ENHANCE
EFFECTIVENESS OF METAHEURISTIC
ALGORITHMS**

MELİKE ÖZTÜRK

Ph.D. THESIS

Department of Engineering Management

Thesis Supervisor

Prof. Dr. Çiğdem ALABAŞ USLU

ISTANBUL, 2020



MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES



**FRACTAL GEOMETRY INSPIRED
SOLUTION GENERATION TO ENHANCE
EFFECTIVENESS OF METAHEURISTIC
ALGORITHMS**

MELİKE ÖZTÜRK

(724815003)

Ph.D. THESIS

Department of Engineering Management

Thesis Supervisor

Prof. Dr. Çiğdem ALABAŞ USLU

ISTANBUL, 2020

AKNOWLEDGEMENTS

I would like to thank and express my most sincere gratitude to my advisor Çiğdem ALABAŞ USLU for the immense knowledge she shared, for the continuous support she showed, for her patience, for her positive attitude which greatly inspired and motivated me. I feel very fortunate to have her as a mentor.

August, 2020

Melike Öztürk

TABLE OF CONTENTS

AKNOWLEDGEMENTS	i
TABLE OF CONTENTS	ii
ÖZET	v
ABSTRACT	vii
CLAIM FOR ORIGINALITY	ix
SYMBOLS	x
ABBREVIATIONS	xii
LIST OF FIGURES	xviii
LIST OF TABLES	xx
1. INTRODUCTION	1
1.2. Motivation of the thesis	2
1.3. Goals and Structure of the Thesis	2
2. LITERATURE REVIEW	4
2.1. Literature Review on Neighbor Generation Methods	4
2.2. Literature Review on Fractal Search and Stochastic Fractal Search	7
2.3. An Evaluation of the Literature Review	23
3. BASIC CONCEPTS IN SEARCH ALGORITHMS	24
3.1. Introduction to Algorithms	24
3.1.1. Search methods	25
3.1.2. Local search algorithm	29
3.1.3 Simulated annealing algorithm	31
3.2. Benchmark Problems	33
3.2.1. Travelling salesman problem	33

3.2.2. Quadratic assignment problem	34
3.3. Fractal Geometry.....	35
3.3.1. Recursive algorithms of fractal sets.....	36
3.3.2. Deterministic generation of cantor sets.....	37
3.3.3. Random generation of cantor sets.....	38
4. CANTOR-SET BASED METHOD	39
5. EXPERIMENTAL STUDY	52
5.1. Design Issues in CB Method.....	52
5.1.1. The number of stages that CB method proceeds	53
5.1.2. Dividing procedure of the strings in each stage.....	53
5.1.3. Reverse ordering of mid-substrings.....	55
5.1.4. Neighbor generation schemes: scheme-1 and scheme-2	55
5.2. Statistical Analysis of Design Issues in CB Mechanism.....	58
5.2.1. Stage-1 experiment report.....	59
5.2.2. Results of the stage-1 experiments	66
5.2.3. Stage-2 experiment report.....	67
5.2.4. Results of the stage-2 experiments	72
5.2.5. Stage-3 experiment report.....	77
5.3. Performance of CB Method on Additional Test Instances	83
5.3.1. Randomization of CB mechanisms.....	83
5.3.2. Parameter tuning for simulated annealing	84
5.3.3. Local search experiments.....	85
5.3.4. Simulated annealing experiments	89
6. CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS	93
6.1. Conclusions	93

6.2. Future Research Directions	97
REFERENCES	98
CURRICULUM VITAE	119



ÖZET

METASEZGİSEL ALGORİTMALARIN ETKİNLİĞİNİ ARTTIRMAK İÇİN ESİN KAYNAĞI FRAKTAL GEOMETRİ OLAN ÇÖZÜM OLUŞTURMA

Gerçek hayatta karşılaşılan eniyileme problemleri, en iyi çözümün bulunabilmesi için fazla karmaşıktır. Bu yüzden, birçok araştırmacı daha etkin çözüm yöntemleri üzerinde çalışır. Bunun sonucunda metasezgisel algoritmalar 90'lı yılların ortalarından beri dünya çapında popülerlik kazanmıştır. Bir sezgisel algoritmanın etkinliğini ve etkililiğini artırmak için algoritmanın arama kabiliyeti geliştirilmelidir. Bu yüzden, metasezgisellerin geliştirilmesinde yaygın olarak kullanılan yaklaşım, yeni bir algoritma tasarlamak ya da var olan bir algoritmayı iyileştirmektir. Bu yaklaşıma kıyasla daha az üzerine düşülen başka bir yaklaşım ise, yeni bir komşuluk üretme mekanizması tasarlamaktır. Bu çalışmada, permutasyon çözüm gösterimini kullanan tekli-çözüm tabanlı sezgiseller için kullanılan yeni bir komşuluk üretme mekanizması sunulmuştur. Önerilen mekanizma cantor kümesi tabanlı (CB) yöntem olarak adlandırılmıştır. Bu mekanizmanın esin kaynağı, fraktal bir yapı olan cantor kümesi oluşturmak için kullanılan yineleme algoritmasıdır. Mekanizma basitçe şöyle çalışır: bir çözüm gösterimi, yineleme algoritmasının kurallarına göre parçalara ayrılır, ortaya çıkan parçaların sırası rastgele değiştirilir ve parçalar yeniden birleştirilir. CB yöntemi, klasik değiş tokuş (*swap*) ve yer değiştirme (*insertion*) mekanizmalarıyla karşılaştırılmış; bu karşılaştırmanın yapılabilmesi için mekanizmalar yerel arama (*local search*) ve tavlama benzetimi (*simulated annealing*) algoritmaları içinde kullanılmıştır. En etkin ve etkili tasarımın bulunabilmesi için farklı uygulama şekilleri dikkate alınarak CB yönteminin türleri oluşturulmuştur. Bu türlerin üzerinde deney yapabilmek için gezgin satıcı problemine (TSP, *traveling salesman problem*) ve kareli atama problemine (QAP, *quadratic assignment problem*) ait test problemleri kullanılmıştır. TSP ve QAP problemleri, amaç fonksiyonu değerleri cinsinden birbirlerinden çok farklı çözüm uzayı yapılarına sahip oldukları için seçilmişlerdir. Özel olarak, TSP'nin çözüm uzayı çok girintili-çıkıntılı bir yüzeye sahiptir ve büyük değişikliklerle elde edilebilecek komşu çözümlerden olumlu etkilenir. QAP'nin çözüm uzayı düz bir yüzeye sahiptir ve küçük değişikliklerle yaratılan komşu çözümlerden olumlu etkilenir. TSP ve QAP test problemlerini kullanarak CB yönteminin duyarlılığını analiz etmek amaçlanmıştır. CB türlerini analiz etmeye ek olarak, CB türleri, değiş tokuş ve yer değiştirme mekanizmaları ile karşılaştırılmıştır.

Değiş tokuş ve yer değiştirme mekanizmaları, literatürde bulunan en temel mekanizmalar oldukları için seçilmişlerdir. Testler sonucunda CB'nin farklı türlerinin TSP ve QAP'ye etkin bir şekilde çözüm buldukları görülmüştür. Sonuç olarak, TSP ve QAP için tutarlı bir şekilde tatmin edici sonuçlar verebilecek genel bir mekanizmanın tasarlanabileceği kararına varılmıştır.

Anahtar kelimeler: Komşuluk oluşturma, yerel arama, tavlama benzetimi, cantor kümesi, gezgin satıcı problem, kareli atama problemi



ABSTRACT

FRACTAL GEOMETRY INSPIRED SOLUTION GENERATION TO ENHANCE EFFECTIVENESS OF METAHEURISTIC ALGORITHMS

Real-life optimization problems are too complex to solve optimally. Therefore, numerous researchers work on more efficient solution methods which led metaheuristic algorithms to gain world-wide popularity since mid-90s. To improve the efficiency and effectiveness of a heuristic algorithm, the searching capability of the algorithm needs to be improved. Hence, the usual approach in improving the science of metaheuristics is to design a novel algorithm or improve an existing algorithm. However, a relatively less explored approach is to design a new neighbor generation mechanism. In this study, a novel neighbor generation mechanism for single solution-based heuristics which use permutation solution representation is presented. The proposed mechanism is called cantor-set based (CB) method. The inspiration for the mechanism is the recursive algorithm that constructs a cantor set which is a fractal shape. Simply, a solution representation is divided based on the recursive algorithm and the resulting pieces are permuted and then combined again to generate neighbors. CB method is embedded into the classical local search (LS) and simulated annealing (SA) algorithms to test its advantages and to compare it with classical swap and insertion neighbor generation mechanisms. Different types of applications are considered to design CB method to find the most effective and efficient design of the method. A set of benchmark problems of the traveling salesman problem (TSP) and quadratic assignment problem (QAP) are used in the experiments to test these applications. TSP and QAP have very different solution spaces considering their objective functions which is why they are chosen as test problems. More specifically TSP has a steep landscape and benefits from drastic changes in neighbor generation while QAP has a flat landscape and requires delicate changes in neighbor generation. Therefore, the aim in using TSP and QAP as test problems is to analyze sensitivity of CB method. In addition to analyzing CB applications, CB method is compared to swap and insertion mechanisms. Swap and insertion are chosen because they are the most used and basic mechanisms that are encountered in the literature. The computational tests show that different applications of CB solve TSP and QAP very efficiently. In the end, it is concluded that it is possible to design a general mechanism that gives consistently good results for both TSP and QAP.

Keywords: Neighbor generation, local search, simulated annealing, cantor set, traveling salesman problem, quadratic assignment problem.



CLAIM FOR ORIGINALITY

Heuristic algorithms are essential to solve real-life optimization problems that have realistic objective functions and constraints. They are very popular amongst researchers that when a problem needs to be solved, a researcher first investigates the types of algorithms that are previously used to solve the same problem or similar problems. Consequently, designing new algorithms or improving existing algorithms are the usual approach in optimization research.

Heuristic algorithms can be categorized as problem-specific heuristics and metaheuristics. Metaheuristic algorithms can be categorized as population-based and single solution-based algorithms. Simply, single solution-based algorithms start from a single initial solution and that solution is manipulated according to a neighbor generation mechanism until a local optimum is reached. The neighbor generation mechanism is chosen by the researcher based on the needs of the problem at hand. There are multiple often used neighbor generation mechanisms in the literature. In their study, researchers use one or multiple mechanisms, separately or in parallel / in combination with one another.

The aim of the presented study is to design of a novel neighbor generation mechanism. A second aim is to design the mechanism in a way that could be used in a variety of problem types.

SYMBOLS

A	: A set of arcs
a'	: The number of neighbors generated in stage-1
b	: The number of neighbors generated in stage-2 while using specific dividing
b'	: The number of neighbors generated in stage-2 while using sequential dividing
c	: The number of neighbors generated in stage-3 while using specific dividing
c'	: The number of neighbors generated in stage-3 while using sequential dividing
C_i	: Iteration number i
C_{ij}	: Distance from i to j
d_{ij}	: The distance between locations i and j
D	: A distance matrix
F	: A flow matrix
f_{ij}	: The flow between facilities i and j
$f(x')$	: Objective function value of a neighbor of solution x
$f(x_{\text{best}})$	: Objective function value of solution x_{best}
$f(x)$	: Objective function value of solution x
G	: A graph
L	: A line with a 1 unit length
L_l	: Left side of the line L
L_m	: Middle of a line L
L_r	: Right side of the line L
n	: Problem size
$N(x)$	: Neighborhood of a solution x
p_{ij}	: j^{th} substring of string- i
p_0	: A complete string
r	: A counter
$\hat{S}$	: The subsets of the city set N
S	: Solution space
T	: Temperature parameter
V	: A set of vertices
x'	: A neighbor of solution x

- x_{best} : Best solution x
- x_{ij} : A binary decision variable
- x : A solution vector
- α : Cooling rate of geometric cooling schedule
- ϑ : Equilibrium state of cooling schedule in the algorithm
- Δf : The difference between the objective functions of x' and x



ABBREVIATIONS

4SS	: Four Step Search
ABC	: Artificial Bee Colony
ACO	: Ant Colony Optimization
AGA	: Adaptive Genetic Algorithm
AGC	: Automatic Generation Control
ALO	: Ant Lion Optimizer
AMALGAM-SO	: Multialgorithm Genetically Adaptive Method for Single Objective Optimization
AM-PSO	: Particle Swarm Optimization with an Analytical Method
AR-ICA	: Imperialist Competitive Algorithm with Attraction and Repulsion Concepts During the Search
ASACO	: Adaptive Penalty Function with the Basic Ant Colony Approach
AVR	: Automatic Voltage Regulator
BA	: Bat Algorithm
BBO	: Biogeography-Based Optimization
BFA	: Bacterial Foraging Optimization Algorithm
BM	: Block Matching
BSA	: Backtracking Search
CB	: Cantor-Set Based Method
CFS	: Chaotic Fractal Search
CFWT	: Chaotic Fractal Walk Trainer
CGBABC	: Chaotic Global Best Artificial Bee Colony
CoBiDE	: Covariance Matrix Adaptation in Differential Evolution
CoDE	: Composite Differential Evolution
CS	: Cuckoo Search
CSA	: Cuckoo Search Algorithm
CS-GA	: Hybridized Cuckoo Search and Genetic Algorithm
C-SOMGA	: Self-Organizing Migrating Genetic Algorithm
CSOS	: Chaotic Symbiotic Organisms Search
dDSF-EA	: Decremental Differential Stochastic Fractal Evolutionary Algorithm
DE	: Differential Evolution

DE-BM : Hybridized Differential Evolution and Block Matching
DS : Diamond Search
dynNP-jDE: Self-Adaptive Differential Evolution Algorithm Dealing with NP
E : Experiment
EA : Evolutionary Algorithm
EA-OPF : Evolutionary Algorithm with Optimal Power Flow
EHS : Enhanced Harmony Search
ELF : Exhaustive Load Flow
EOPF : Exhaustive Optimal Power Flow
EPSDE : Hybridized Mutation Strategies and Control Parameters with Differential Evolution
ES : Evolution Strategy
ES-PSO : Hybridized Eagle Strategy and Particle Swarm Optimization
FA : Firefly Algorithm
FCASO-SQP: Hybridized Fuzzy Adaptive Chaotic Ant Swarm Optimization and Sequential Quadratic Programming
FGO : Fuzzy Global Optimization
FPA : Flower Pollination Algorithm
FS : Fractal Search
FSA : Full Search Algorithm
FSP : Flow-shop Scheduling Problem
GA : Genetic Algorithm
GA-IWD : Hybridized Genetic Algorithm and Intelligent Water Drop
GAPF : Genetic Algorithm-Based Penalty Function
GA-PSO : Hybridized Genetic Algorithm and Particle Swarm Optimization
GA-SRS : Modified Genetic Algorithm
GSA : Gravitational Search Algorithm
GSA-SQP : Hybridized Gravitational Search Algorithm and Sequential Quadratic Programming
GWO : Grey Wolf Optimizer
HCLPSO : Heterogeneous Comprehensive Learning Particle Swarm Optimization
HPSO : Hybrid Particle Swarm Optimization
HS : Harmony Search

HSAPABC: Hybridized Harmony Search and Particle Artificial Bee Colony Algorithm

HS-BM : Hybridized Harmony Search and Block Matching

HSFS : Hybrid Stochastic Fractal Search

hSFS-PS : Hybridized Stochastic Fractal Search and Pattern Search

HS-IPSO-GA: Hybridized Harmony Search, Improved Particle Swarm Optimization, and Genetic Algorithm

HLSO : Hybridizing Sum-Local Search Optimizer

IA : Immune Based Algorithm

IAA : Improved Analytical Approach

IACS : Improved ACO

IAs : Penalty Guided Artificial Immune Algorithm

ICA-GA : Hybridized Imperialist Competitive Algorithm and Genetic Algorithm

ICS : Improved Cuckoo Search

ICSA : Improved Cuckoo Search Algorithm

IEEE : The Institute of Electrical and Electronics Engineers

IGAMU : Improved Genetic Algorithm with Multiplier Updating

ILS : Iterated Local Search

IMOHS : Improved Multi-Objective Harmony Search

I-NESA : Improved Nonequilibrium Simulated Annealing

INGHS : Improved Novel Global Harmony Search

IP : Interior Point Algorithm

IPSO : Improved Particle Swarm Optimization

ISFS : Improved Stochastic Fractal Search

ISFSA : Improved Stochastic Fractal Search Algorithm

ITS : Improved Tabu Search

IWD : Intelligent Water Drop Algorithm

IWO : Invasive Weed Optimization

JADE : Adaptive Differential Evolution with Optional External Archive

KHA : Krill Herd Algorithm

LS : Local Search

LSA : Lightning Search Algorithm

LSF : Loss Sensitivity Factor

LSFSA : A Loss Sensitivity Factor Simulated Annealing

L-SHADE : Linear Population Size Reduction Embedded Success History Based Adaptive Differential Evolution

LUS : Local Unimodal Sampling

LXPM-IPSO:Laplace Crossover Power Mutation Genetic Algorithm with Improved Particle Swarm Optimization

LXPM-IPSO-GS: Hybridized Laplace Crossover Power Mutation Genetic Algorithm, Improved Particle Swarm Optimization, and Golden Search

MA : Memetic Algorithm

MADS-LHS: Mesh Adaptive Direct Search and Latin Hypercube Sampling

MADS-PSO: Mesh Adaptive Direct Search and Particle Swarm Optimization

MBA : Mine Blast Algorithm

MFO : Moth-Flame Optimization

MLP : Multilayer Perceptron Neural Network

MODE : Multi-Objective Differential Evolution

MOEA/D : Multi-Objective Evolutionary Algorithm Based on Decomposition

MOEP : Multi Objective Evolutionary Programming

MOGWO : Multi-Objective Gray Wolf Optimizer

MOL : Many Optimizing Liaisons

MOPSO : Multi-Objective Particle Swarm Optimization

MOSFS : Multi-Objective Stochastic Fractal Search

MPEDE : Multi-Population Ensemble Differential Evolution

MPSO : Modified Particle Swarm Optimization

MRPS : Modified Random-to-Pattern Search

MSFS : Modified Stochastic Fractal Search

MSFS-BM: Hybridized Modified Stochastic Fractal Search and Block Matching

NAFSA : Novel Artificial Fish Swarm Algorithm

NC : Not concluded

NGHS : Novel Global Harmony Search

NMDE : Novel Modified Differential Evolution

NSGA-II : Nondominated Sorting Based Multi-Objective Evolutionary Algorithm

NSGA-III : Non-dominated Sorting Genetic Algorithm-III

NTSS : New Three Step Search

OBL : Opposition-Based Learning

ORCSA : One Rank Cuckoo Search
OXDE : Orthogonal Crossover (OX) Operators in Differential Evolution
PBIL : Population-Based Incremental Learning
PDE : Pareto Frontier Differential Evolution
PDF plus : 1 + Proportional Integral
PD-type : Proportional Derivative
PI : Proportional Integral
PI+DF : Proportional Integral Derivative Path First Order
PID : Proportional Integral Derivative
PI-PD : Proportional Integral-Proportional Derivative
PLA-BM : Hybridized Parameterized Learning Automata and Block Matching
PS : Pattern Search
PSO : Particle Swarm Optimization
PSO-BM : Hybridized Particle Swarm Optimization and Block Matching
PSO-GC : Particle Swarm Optimization with Gaussian Distribution and Chaotic Sequence
PSO-RDL : Hybridized Particle Swarm Optimization and Dynamic Linkage Discovery
PSSO : Particle-Based Simplified Swarm Optimization
Q : Questions
QAP : Quadratic Assignment Problem
QOSIMBO-Q: Quasi-oppositional Swine Influenza Model Based Optimization with Quarantine
QOTLBO : Quasi-oppositional Teaching-Learning Based Optimization
QP : Micro-Computer-Based Optimization
QTLBO : Quasi-Oppositional Teacher Learner-Based Optimization
RCCRO : Real Coded Chemical Reaction
REPSO : Rank Evolutionary Particle Swarm Optimization
RSFS : Robust Stochastic Fractal Search
SA : Simulated Annealing
SaDE : Self Adaptive Differential Evolution
SARGA : Self-Adaptive Real-Coded Genetic Algorithm
SDE : Shuffled Differential Evolution
SES : Simple and Efficient Three Step Search

SFS	: Stochastic Fractal Search
SFS-BM	: Hybridized Stochastic Fractal Search and Block Matching
SFS-LUS	: Hybridized Stochastic Fractal Search and Local Unimodal Sampling
SHADE	: Success History Based Adaptive Differential Evolution
SIMBO-Q	: Swine Influenza Model Based Optimization with Quarantine
SMS	: States of Matter Search
SOS	: Symbiotic Organisms Search
SPEA 2	: Strength Pareto Evolutionary Algorithm 2
SQP	: Sequential Quadratic Programming
SS	: Scatter Search
SSO	: Simplified Swarm Optimization
TLBO	: Teaching-Learning Based Optimization
TOPSIS	: Technique for Order Preference by Similarity to Ideal Solution
TPLA-BM	: Hybridized Team of Pursuit Learning Automata and Block Matching
TS	: Tabu Search
TSBDEA	: Taguchi-Sliding-Based Differential Evolution
TS-DE	: Hybridized Tabu Search and Differential Evolution
TSP	: Traveling Salesman Problem
TSS	: Three Step Search
UMOEAs	: Hybridized Genetic Algorithm, Differential Evolution and Covariance Matrix Adaptation Evolution Strategy
VMI	: Vendor Managed Inventory
VNS	: Variable Neighborhood Search
WCA	: Water Cycle Algorithm
WCO	: World Cup Optimization
XKL	: Analytic Algorithms and Local Strategy

LIST OF FIGURES

Figure 3.1. Complexity classes	25
Figure 3.2. Search methods for optimization	26
Figure 3.3. Pseudo code of LS heuristic	30
Figure 3.4. Objective function values of feasible solutions in a search space	31
Figure 3.5. Pseudo code of SA heuristic	33
Figure 3.6. Snowflake	36
Figure 3.7. Fractal set examples	36
Figure 3.8. The recursive algorithm that generates a deterministic cantor set	37
Figure 3.9. Middle-thirds cantor set	37
Figure 4.1. A representative figure for how substrings in each stage of CB would look like	39
Figure 4.2. Pseudo code of CB method	41
Figure 4.3. A flow chart of CB mechanism steps	42
Figure 5.1. A visual representation of neighbor generation using scheme-1	57
Figure 5.2. A visual representation of neighbor generation using scheme-2	57
Figure 5.3. Structure of stage-1 experiments	59
Figure 5.4. Pairwise bar chart comparisons of run-time solving TSP by stage-1 experiments	62
Figure 5.5. Pairwise bar chart comparisons of run-time solving QAP by stage-1 experiments	62
Figure 5.6. Positive and negative differences of performance data to compare ordering issue for QAP	64
Figure 5.7. Positive and negative differences of performance data to compare neighbor generation scheme for QAP	65

Figure 5.8. Positive and negative differences of performance data to compare the size of p_{i2} for QAP	66
Figure 5.9. Structure of stage-2 experiments	67
Figure 5.10. Pairwise bar chart comparisons of run-time solving TSP by stage-2 experiments	73
Figure 5.11. Pairwise bar chart comparisons of run-time solving QAP by stage-2 experiments	73
Figure 5.12. Structure of stage-3 experiments	77
Figure 5.13. Pairwise comparisons of chosen mechanisms for TSP	82
Figure 5.14. Pairwise comparisons of chosen mechanisms for QAP	83

LIST OF TABLES

Table 2.1 A summary of the studies on neighbor generation mechanisms in the literature	6
Table 2.2. Literature review of Fractal Search and Stochastic Fractal Search	21
Table 4.1. Whole neighborhood of $x=[7, 6, 5, 4, 3, 2, 1]$ in stage-1 using scheme-1	43
Table 4.2. Whole neighborhood of $x=[7, 6, 5, 4, 3, 2, 1]$ in stage-2 using scheme-1	46
Table 4.3. Neighborhood of $x = [15,14,13,12,11,10,9,8,7,6,5,4,3,2,1]$ in stage-3 using scheme-1	46
Table 4.4. Whole neighborhood of $x=[7, 6, 5, 4, 3, 2, 1]$ in stage-1 using scheme-2	48
Table 4.5. Whole neighborhood of $x=[7, 6, 5, 4, 3, 2, 1]$ in stage-2 using scheme-2	49
Table 4.6. Whole neighborhood of $x=[15,14,13,12,11,10,9,8,7, 6, 5, 4, 3, 2, 1]$ in Stage-3 using scheme-2	50
Table 5.1. Performance of stage-1 experiments in terms of solution quality	60
Table 5.2. Number of neighbors generated by stage-1 experiments	61
Table 5.3. Run time of stage-1 experiments (in seconds)	61
Table 5.4. Wilcoxon signed ranks test statistics to compare ordering issue for QAP	64
Table 5.5. Wilcoxon signed ranks test statistics to compare neighbor generation scheme for QAP	65
Table 5.6. Wilcoxon signed ranks test statistics to compare the size of p_{i2} for QAP	66
Table 5.7. Conclusion summary for stage-1 experiments	67
Table 5.8. Performance of stage-2 experiments that use specific dividing in terms of solution quality	68
Table 5.9. Performance of stage-2 experiments that use sequential dividing in terms of solution quality	69
Table 5.10. Number of neighbors generated by stage-2 experiments that use specific dividing	70
Table 5.11. Number of neighbors generated by stage-2 experiments that use sequential dividing	70
Table 5.12. Run time of stage-2 experiments that use specific dividing (in seconds)	71
Table 5.13. Run time of stage-2 experiments that use sequential dividing (in seconds)	71

Table 5.14. Conclusion summary of stage-2 experiments	72
Table 5.15. Performance comparison in terms of solution quality after incorporating the fifth design issue	75
Table 5.16. Run-time (in seconds) comparison after incorporating the fifth design issue	75
Table 5.17. Descriptive Statistics	76
Table 5.18. Wilcoxon Signed Ranks test statistics	76
Table 5.19. Performance of stage-3 experiments in terms of solution quality	79
Table 5.20. Number of neighbors generated by of stage-3 experiment	80
Table 5.21. Run time of stage-3 experiments (in seconds)	81
Table 5.22. Performance of chosen variants while solving TSP embedded into LS in terms of solution quality	86
Table 5.23. Performance of chosen variants while solving QAP embedded into LS in terms of solution quality	86
Table 5.24. Average time required (in seconds) for TSP embedded into LS	87
Table 5.25. Average time required (in seconds) for QAP embedded into LS	88
Table 5.26. Descriptive statistics of the mechanisms in terms of percentage deviation from the best-known solution for TSP embedded into LS	89
Table 5.27. Descriptive statistics of the mechanisms in terms of percentage deviation from the best-known solution for QAP embedded into LS	89
Table 5.28. Performance of chosen variants while solving TSP embedded into SA in terms of solution quality	90
Table 5.29. Performance of chosen variants while solving QAP embedded into SA in terms of solution quality	90
Table 5.30. Average time required (in seconds) for TSP embedded into SA	91
Table 5.31. Average time required (in seconds) for QAP embedded into SA	91
Table 5.32. Descriptive statistics of the mechanisms in terms of percentage deviation from the best-known solution for TSP embedded into SA	92
Table 5.33. Descriptive statistics of the mechanisms in terms of percentage deviation from the best-known solution for QAP embedded into SA	92

1. INTRODUCTION

Optimization problems constitute an important part of the industry and daily life. However, most of the real-life problems are too complex to solve using exact solution methods. Even if approximation solution methods are used, there are problems that cannot be solved at all, or there are problems that cannot be solved satisfactorily.

Exact solution methods find an exact solution to a problem by following a set of optimization rules while approximate solution methods find an approximate solution. In analyzing computational complexity of problems, the concern is to determine whether a problem can be solved in polynomial time or not. After modelling a problem, the crucial decision is to choose between using an exact algorithm or approximate algorithm to solve the problem based on the analysis that if a problem can or cannot be solved in a reasonable amount of time using an exact algorithm.

Heuristic algorithms for optimization problems were introduced in the middle of the 20th century and gained world-wide interest. Even though these methods cannot guarantee optimality, they are effective in finding approximate solutions within relatively short computer time. Heuristic algorithms can be categorized as problem specific heuristics and metaheuristics. Problem specific heuristics are designed for a specific problem or problem type while metaheuristics are designed generally to solve a variety of problems. Metaheuristics can be categorized as single solution-based heuristics and population-based heuristics. Single solution-based heuristics start from a single solution and manipulate that solution to find a satisfying solution while population-based heuristics start from a population of solutions and manipulate them to find a satisfying solution. The effectiveness of the search is highly affected by the solution representation because the solution representation affects the manipulation of a solution which in turn affects the effectiveness of the search.

There are countless studies in the literature on heuristic algorithms. Usually studies concern the design of a novel search algorithm, a novel application of an existing algorithm or application of an existing algorithm to a specified problem. As a contribution to the literature, in this study, a novel neighbor generation mechanism called Cantor-Set Based (CB) mechanism is designed. Neighbor generation mechanisms are common to search heuristics as they are responsible for manipulating the solution to find a better

solution which means they make the search proceed.

1.2. Motivation of the Thesis

CB mechanism is inspired by the recursive algorithm that generates a fractal set called cantor set. The objective of CB is to generate self-similar neighbors that retain the good chunks of the solution and that are different enough by benefitting from the fractal nature of the mechanism. It must be noted that CB is not developed for a specific problem or a specific problem type. It is designed as a general mechanism which can be used for any combinatorial problem where a solution point can be represented using permutation solution representation.

There are decision points in the recursive algorithm to generate neighbors as the fractal set. Specifically, the solution representation is treated as a complete string. The first step is to divide the solution representation into three substrings. This first step results in two decision points. What is the length of these substrings? How will these substrings be manipulated to generate neighbors? The second step is to divide the previously created substrings according to the rules of the recursive algorithm. Another decision point is to decide how many times the dividing will be done? The specifics of these questions are discussed in Section 5. However, the reason to mention them here is to argue that, there are various ways to implement CB mechanism, because these decision points create various design possibilities to generate neighbors.

Thus, design possibilities that can be applied are tested, and the findings are reported in the presented study. In the computational tests, CB is embedded into Local Search (LS), which is a deterministic heuristic, and Simulated Annealing (SA), which is a stochastic heuristic, separately. The results are obtained by solving two combinatorial optimization problems, namely Traveling Salesman Problem (TSP) and Quadratic Assignment Problem (QAP) which exhibit very different characteristics in their solution spaces, the first having a steep landscape and the latter having a flat landscape.

1.3. Goals and Structure of the Thesis

The presented study describes a novel neighbor generation mechanism called CB mechanism. The contribution to the literature is a neighbor generation method which is a topic often overshadowed by the studies on search algorithms, therefore, deserving more research. Design possibilities are considered to make CB a general mechanism which is

applicable to a variety of problems. In the moment, the problems are solved by tailoring existing neighbor generation methods to the needs of the problem. This complication is aimed to be solved in the presented research.

In the 2nd section literature reviews on neighbor generation mechanisms and usage of fractals in optimization are presented. In the 3rd section, basic concepts that are used and mentioned in the study are explained. The section is prepared to not disturb the continuity with explanations. In the 4th section, the theory behind CB mechanism is described. In the 5th section experimental study is reported and finally in the 6th section, conclusions are drawn based on the findings.



2. LITERATURE REVIEW

The literature review consists of three parts. In the first part a review for the studies that present or analyze neighbor generation methods for single solution-based algorithms, regardless of problem type or search algorithm is presented. In the second part, a review for fractal concepts in optimization is presented. In the final part, an evaluation of the literature is given and also, some conclusions on the differences and value of the presented study are shared.

2.1. Literature Review on Neighbor Generation Methods

In this subsection, comparative studies on existing neighbor generation mechanisms and novel neighbor generation mechanisms are examined. In Table 2.1, a summary of the studies is shown. Also, a brief mention of neighbor generation mechanisms that are designed specifically for TSP and QAP is shared at the end of the section.

Tian et al. (1999) compared neighbor generation mechanisms on TSP, QAP and Flow Shop Scheduling Problem (FSP), which are suited to permutation solution representation, to investigate performance of SA according to neighboring mechanisms. The compared mechanisms are consecutive swap (swapping two elements on consecutive positions), general swap (swapping two elements on any random positions), general insertion (inserting a random element to a random position), block insertion (changing the positions of a sub-sequence of elements randomly), block reversion (reversing the positions of a sub-sequence of elements randomly), block reversion and/or block insertion. It is concluded that different mechanisms are suitable for different problem types. Tian et al. (1999) stated that the best methods are block reversion and/or block insertion for TSP, general swap for QAP and block insertion for FSP.

Ben-Daya and Al-Fawzan (1998) also analyzed the effects of neighbor generation on the performance of Tabu Search (TS) solving FSP. They figured out that if TS algorithm alternates between insertion, block insertion, or swap mechanisms randomly, efficiency and effectiveness are balanced, and TS performs well. Also, Nearchou (2000) tested the performance of the same neighboring mechanisms in Tian et al. (1999) and embedded these mechanisms into a hybridization of SA, LS, and Genetic Algorithms (GA) to solve FSP. At the end of the experimentation, random swap mechanism was selected as the most suitable mechanism to solve FSP using a single solution based heuristic algorithm.

Deroussi et al. (2008) presented a new solution representation based on vehicles rather than machines for the problem of simultaneous scheduling of machines and vehicles in flexible manufacturing systems. The aim of the study is to analyze the efficiency of Iterated Local Search (ILS), SA and ILS/SA hybridization that use swap and insertion mechanisms to generate neighbors. In addition, a method called neighborhood reduction is incorporated into the mechanisms in which the size of the neighborhood is decreased by some pre-defined rules. In the experiments, swap, reduced swap, insertion, reduced insertion, and reduced swap and insertion are used and analyzed separately. It is concluded that using both reduced swap and insertion gives the best performance.

Zhang et al. (2010) used SA algorithm to solve high school timetabling problem. In the study, a neighborhood structure is designed in which the neighbors are generated by using the swap mechanism between time slots, instead of assignments which is the usual option. Two sets of benchmark instances are used to perform computational tests, and the conclusion is that their heuristic performs better than the compared methods.

Beham et al. (2015) aimed to compare some single solution-based heuristics (LS, SA, TS, Variable Neighborhood Search (VNS)) and population-based heuristics (GA, Memetic Algorithm (MA)) to solve QAP. In the experiments, swap is used in the single solution-based algorithms, because Beham et al. (2015) stated that the most used neighbor generation mechanism in QAP studies is swap. It is concluded that there is not a single best algorithm for solving QAP and different instances respond well to different heuristics.

Table 2.1. A summary of the studies on neighbor generation mechanisms in the literature

Author(s), Year	Separately	In alteration	Mechanism	Heuristic	Problem
Tian et al. (1999)	x		consecutive swap, general swap, general insertion, block insertion, block reversion, block reversion and/or block insertion	SA	TSP, QAP, FSP
Ben-Daya and Al-Fawzan (1998)		x	insertion, block insertion, or swap	TS	FSP
Nearchou (2000)	x		consecutive swap, general swap, general insertion, block insertion, block reversion, block reversion and/or block insertion	hybrid LS+SA+GA	FSP
Deroussi et al. (2008)	x		Swap, insertion	ILS, SA, and hybrid ILS+SA	Scheduling and transportation problems
Zhang et al. (2010)	x		Swap	SA	High school timetabling problem
Beham et al. (2015)	x		Swap	LS, SA, TS, VNS, GA, MA	QAP

There are various neighbor generation mechanisms that are designed specifically for TSP such as r-opt (Lin and Kernighan, 1973), ejection chains (Glover, 1991), stem-and-cycle (Glover, 1996). These mechanisms cannot be generalized to other combinatorial problems. Rego et al. (2011) compared most successful LS algorithms which employ six variations of Lin-Kernighan heuristic and two variations of stem-and-cycle algorithm and three generalizations derived from these variations for symmetric and asymmetric TSP. They examined which mechanism is successful in what way. They also showed how to view all the implementations in an ejection chain framework in order to show the similarities and differences for solving large and difficult TSP instances. It is concluded that the stem-and-cycle based approaches outperform the basic Lin-Kernighan based variations in terms of solution quality, while needing longer run-times.

Most of the studies in the literature use swap mechanism for the QAP problem (see for examples; Angel and Zissimopoulos (2001), Mills et al. (2003), Misevičius (2000), Stützle (2006), Taillard (1991)). This means that there is not a neighbor generation mechanism specifically designed to solve QAP in the literature. On the other hand, the interest of the present study lies with generalized neighbor generation mechanisms not problem specific ones.

2.2. Literature Review on Fractal Search and Stochastic Fractal Search

In this section, a literature review on the usage of fractal concepts in optimization is presented. The earliest mention of fractals in optimization is found in Sorkin (1991). In the study, the framework of SA algorithm is analyzed. Sorkin (1991) stated that the efficiency of the algorithm depends on the energy landscapes which is the way solution space is called in the study, therefore, energy landscapes must be improved in order to improve the algorithm. For this reason, fractal properties are added to the algorithm design in a geometric cooling schedule which is modeled as a random walk on a graph (A *random walk* is a stochastic process that creates a path with a succession of random steps on a mathematical space). The comparisons are made by using random search (which is SA that use random neighbor generation) and decent algorithms (which is SA that generates a neighborhood and always choose a better solution than the solution at hand). The experiments are made by using three placement problems with different problem sizes (a real-world example with 87 objects, one with a random graph with 225 objects, and a randomly (hierarchically) constructed problem with 225 objects). The conclusion of the study is that, for some fractals, the presented method can give better results, however the run times increase exponentially with problem size.

After that, fractal geometry is used in heuristic algorithms by Salimi (2015). The presented algorithms are called Fractal Search (FS) and Stochastic Fractal Search (SFS) algorithms. In the study, the designed algorithms are described by using the discharge pattern of a dielectric breakdown. Diffusion limited aggregation method which is a physically generated model based on clusters describing a bacterial colony concept is used to grow random fractals. Levy flight and Gaussian walk are fractal methods that is used in the algorithms to generate random neighbors which are called particles in FS. A *Levy flight* is a type of random walk. It has step lengths with a probability distribution

that has infinite variance. When defined as a walk, the steps are taken to random directions with same lengths. Levy flight is also used when random walk takes place on a discrete space. A *Gaussian walk* is a random walk that takes place on a continuous space and the walk is associated with normally distributed random variable in every direction.

FS is a population based heuristic algorithm that uses random fractals to improve effectiveness. It is explained that, although FS performs well, it has a lot of parameters and information exchange does not occur among the particles which creates disadvantages that result in much more randomness in the algorithm than desired. For this reason, FS is thought to be similar to random search. In addition, the exploration of the search space is uncontrollable. So, a diffusion process and updating process is added to FS and the resulting algorithm is called SFS. SFS is a population based heuristic algorithm that uses only Gaussian walk to generate random fractals in the *Diffusion Process*. Then in the *Updating Process*, the algorithm updates the existing particles (best ones) instead of creating new ones.

The continuation of this literature review consists of studies that use FS or/and SFS. The studies can be observed in Table 2.2 in summary. In the studies, FS or/and SFS is used either in their pure state, in hybridization with other algorithms, in comparison with other algorithms or by applying some improvement. The resulting algorithms are used to solve different problems in different areas such as databases, production, image processing, allocation, power systems operations, medicine, design and construction, supply chain management, electronics, neural networks, chaotic systems.

Sohal et al. (2015) designed a new framework for solving query optimization in distributed databases based on SFS in its pure form. Query optimization in databases is an NP-Hard problem which aims to plan queries in a way that optimizes the execution cost.

Awad et al. (2016) presented a hybridization between Differential Evolution (DE) and update processes of the SFS called Decremental Differential Stochastic Fractal Evolutionary Algorithm (*dDSF-EA*) for solving global numerical optimization. In the computational study, thirty benchmark functions of real parameter single objective competition are used. The comparisons are made between dynNP-jDE (Brest and Maučec, 2008, self-adaptive DE algorithm dealing with NP), OXDE (Wang and Zhang,

2012, orthogonal crossover (OX) operators in DE), JADE (Zhang and Sanderson, 2009, adaptive DE with optional external archive), SaDE (Qin et al., 2008, self-adaptive DE), SHADE (Tanabe and Fukunaga, 2013, success history based adaptive DE), L-SHADE (Tanabe and Fukunaga, 2014, linear population size reduction embedded SHADE), EPSDE (Mallipeddi et al., 2011, an ensemble of mutation strategies and control parameters with the DE), CoDE (Wang and Zhang, 2011, a composite DE), CoBiDE (Wang et al., 2014, covariance matrix adaptation in DE), UMOEAs (Elsayed et al., a hybridized GA, DE and covariance matrix adaptation evolution strategy), and AMALGAM-SO (Vrugt et al., 2008, multialgorithm genetically adaptive method for single objective optimization), SFS and *d*DSF-EA. Awad et al. (2016) concluded that *d*DSF-EA is more efficient in solving the problem than SFS, L-SHADE component algorithms and other compared algorithms.

Khalilpourazari et al. (2016) used and compared SQP (Spellucci, 1998, a sequential quadratic programming), SFS, SA, and WCA (Eskandar et al., 2012, water cycle algorithm) to solve a model that aim to minimize the total inventory cost, in which shortages are allowed and partially backordered with fixed and linear costs subject to realistic real-world constraints. Ninety numerical examples are solved to evaluate the performance of the algorithms. Khalilpourazari et al. (2016) concluded that WCA and SFS shows the best performance in terms of objective function and run-time.

Li (2016) studied the problem of target matching via edge features. In this problem, the contribution is to use a novel potential-based shape matching model with a new similarity measurement criterion. Also, comparisons are made with existing similarity measurement criterion. To optimize the geometric transformations SFS algorithm is utilized.

Mellal and Zio (2016) modified SFS to be penalty guided to solve reliability optimization problems (namely reliability allocation, redundancy allocation, and reliability-redundancy allocation problems). The computational study has been executed by using ten case studies and comparisons are made with GA (Holland, 1992), PSO (Kennedy and Eberhart, 1995, particle swarm optimization), ABC (Karaboga, 2005, artificial bee colony), BBO (Simon, 2008, biogeography-based optimization), CS (Yang and Deb, 2009, cuckoo search), SA (Kirkpatrick et al., 1983), HS (Geem et al., 2001, harmony search), CS-GA (Kanagaraj et al., 2013, a hybridized CS and GA), GA-SRS (Ardakan

and Hamadani, 2014, a modified GA), ES-PSO (Liu and Qin, 2014, a combination of eagle strategy with PSO), ICS (Valian et al., 2013, an improved CS), MPSO (Liu and Qin, 2014, a modified PSO), IPSO (Wu et al., 2011, an improved PSO), NGHS (Ouyang et al., 2015, a novel global HS), INGHS (Ouyang et al., 2015, an improved novel global HS), IA (Hsieh and You, 2011, an immune based algorithm), IAs (Chen, 2006, a penalty guided artificial immune algorithm), LXPM-IPSO-GS (Zhang et al., 2013, a hybridized Laplace crossover power mutation GA, improved PSO, and golden search), HNN-SC (Hsieh et al., 1998, name assigned by the editors of the study), XKL (Xu et al., 1990, analytic algorithms and local strategy), AR-ICA (Afonso et al., 2013, an imperialist competitive algorithm with attraction and repulsion concepts during the search), SSO (Yeh, 2009, simplified swarm optimization), PSSO (Huang 2015, a particle-based SSO), GA-PSO (Sheikhalishahi et al., 2013, a hybridized GA and PSO), DE (Liu and Qin, 2015, a DE algorithm combined with Lévy flight), TS-DE (Liu and Qin, 2014, a hybridized TS and DE), PSO-GC (dos Santos Coelho, 2009, a PSO based on Gaussian distribution and chaotic sequence), NMDE (Zou et al., 2011, a novel modified DE), FGO (Ravi et al., 2000, a fuzzy global optimization), I-NESA (Ravi et al., 1997, an improved nonequilibrium SA), C-SOMGA (Deep, 2009, a self-organizing migrating GA), MRPS (Murty and Reddy, 1999, a modified random-to-pattern search), LXPM-IPSO (Zhang et al., 2013, a Laplace crossover power mutation GA with improved PSO), HS-IPSO-GA (Zhang et al., 2013, a hybrid approach based on HS, improved PSO, and GA), AGGA (Agarwal and Gupta, 2006, a GA), ASACO (Agarwal and Sharma, 2010, an adaptive penalty function with the basic ant colony approach), NAFSA (He et al., 2015, a novel artificial fish swarm algorithm). Mellal and Zio (2016) concluded that the proposed method performs better than the compared methods.

Mosbah and El-Hawary (2016) used SFS algorithm to execute tracking state estimation in order to solve a nonlinear power system problem under sudden load changes. To show the efficiency of the algorithm four test cases in various sizes are used and comparisons are made by using GA and PSO algorithms. Mosbah and El-Hawary (2016) concluded that the proposed method gives a precise state estimator while other estimators do not. However, the method suffers from long run-times.

Rahman and Tokhi (2016) modified SFS by adding five different chaotic maps (namely, Chebyshev map, Circle map, Gauss/Mouse map, Iterative map, Logistic map) aiming for

making the search more focused and less random. In the computational tests, eight standard continuous benchmark functions are used. Rahman and Tokhi (2016) concluded that SFS modified by adding Gauss/Mouse map gives the best performance especially when embedded into Diffusion Process of SFS. Also, the overall run-time is faster when chaotic maps are embedded into SFS compared to the run-time observed when pure SFS is utilized.

Chuan et al. (2016) studied protein structure prediction problem that is used in areas like research and development of drugs. In the study, the problem is modelled as a numerical optimization model and SFS in its pure form is used to calculate minimum free energy value of protein structure.

Tejani et al. (2016) studied optimizing the design of construction, namely minimizing the weight subjected to physical constraints. SFS, GWO (Mirjalili 2014, grey wolf optimizer), JADE is used for solving the problem and comparisons are made between the algorithms as well as with TLBO (Toğan 2012, teaching-learning based optimization), EHS (Maheri and Narimani 2014, enhanced harmony search) and GA results from the literature. Tejani et al. (2016) concluded that the best and second-best results are given by SFS and JADE respectively.

Tyagi et al. (2016) solved a multi-objective optimal dispatch problem of solar-wind-thermal system by using an improved version of stochastic fractal search (ISFSA) algorithm. In the computational tests multiple multi-objective complex test systems are used. Also, comparisons are made between the presented algorithm and DE, QTLBO (Mandal and Roy 2013, quasi-oppositional teacher learner-based optimization), BSA (Civicioglu 2013, backtracking search), RCCRO (Bhattacharjee et. al 2014, real coded chemical reaction), SFS. The comparisons are made by utilizing TOPSIS (technique for order preference by similarity to ideal solution) ranking system and in conclusion, ISFSA is found to give the best results.

Ashraf et al. (2017) studied a non-linear integer programming problem which is vendor managed inventory (VMI) systems problem. Three variations of VMI problem, namely single supplier - single retailer, single supplier - multi-retailer, and multi supplier - multi retailer are solved and the aim is to minimize the cost. In the paper a hybrid of SFS and BBO is used. Also, comparisons are made using results of GA and a heuristic algorithm

developed by Cárdenas-Barrón et al. (2012) from the literature. Ashraf et al. (2017) concluded that the presented method outperforms the compared methods.

El-Fergany and Hasanien (2017) studied the optimal relay coordination problem of meshed power networks which is bus networks in various sizes (namely eight, nine, fifteen bus networks). SFS is used in its pure form and the comparisons are made with FPA (Yang 2012, flower pollination algorithm), GSA (Rashedi et al., 2009, gravitational search algorithm), SQP (Radosavljević, and Jevtić 2016), GSA-SQP (Radosavljević, and Jevtić 2016, a hybridized GSA and SQP) and DE algorithms from the literature. El-Fergany and Hasanien (2017) concluded that SFS outperforms the compared algorithms on performance and shows lower run-time while needing to tune less parameters.

Khanam and Parmar (2017a) solved a continuous problem which is order reduction of large-scale single input single output linear time invariant systems using SFS algorithm. In the problem, order reduction is necessary to avoid the problem becoming too complex. In the computational tests, two numerical examples are solved, and the performance of the algorithm is evaluated by evaluating the step and frequency responses. The comparisons are made with the techniques taken from the literature and SFS is found to give lower integral square error value than the compared methods.

The information given here concerns some of the following papers. In control systems, various conditions require different responses from a system. For example, AGC (automatic generation control) systems, AVR (automatic voltage regulator) systems of power systems are close loop systems that need to be controlled and responses need to be used as corrective feedback. Some of the controllers are PI (proportional integral) controller, PID (proportional integral derivative) controller, PI-PD (proportional integral-proportional derivative) controller, PD-type (proportional derivative) fuzzy logic controller, PDF plus (1 + PI) controller. These controllers are studied in some of the following papers.

Khanam and Parmar (2017b) studied the speed control of DC motors. In the problem, tuning the parameters of PID controller is a necessary step which is done by using SFS algorithm. In the literature, similar usages of solving the parameter tuning problem of PID with heuristics like PSO and IWO (Mehrabian and Lucas, 2006, invasive weed optimization) exists. The comparisons are made between SFS and PSO, IWO and results

showed that SFS gives comparable results with PSO and IWO and in addition SFS do not give over estimation in any parameters.

Lin and Wang (2017) studied the parameter identification of fractional order chaotic systems. To determine the parameters, a hybrid SFS (HSFS) is developed by integrating various LS principles into SFS. Specifically, OBL (Tizhoosh, 2005, opposition-based learning), DE and a re-initialization mechanism are embedded into SFS. In the computational study, typical fractional-order chaotic systems are simulated, and comparisons are made between HSFS and PSO, DE, SFS. Lin and Wang (2017) concluded that HSFS is an effective algorithm.

Mosavi et al. (2017) used SFS for training radial basis function of neural networks. Benchmark sonar iris and lenses datasets are used in the computational study and comparisons are made between SFS and PSO, ACO, ES (evolution strategy), GA, PBIL (Baluja, 1994, population-based incremental learning). Mosavi et al. (2017) concluded that SFS gives better results than the compared algorithms in terms of classification accuracy and run-time due to its simple structure.

Mosbah and El-Hawary (2017) worked on determining the optimal set of parameters for multilayer perceptron neural network (MLP) using SFS, meaning MLP is optimized by SFS. The problem is the dynamic state estimation under communication failure problem and in the computational study, IEEE (The Institute of Electrical and Electronics Engineers) 14-bus systems and IEEE 118-bus systems combined with the New York Independent System Operator under differing scenarios have been used. Comparisons are made between MLP optimized by SFS, nonoptimized MLP, MLP optimized by GA, MLP optimized by a hybrid SFS-PSO. Mosbah and El-Hawary (2017) concluded that MLP optimized by SFS is superior to the compared algorithms.

Padhy and Panda (2017) used a hybridization of SFS with Pattern Search (PS) called hSFS-PS to optimize a controller for thermal, hydro and gas power unit-based power systems named PI-PD controller. The comparisons are made between SFS and other approaches in the literature that have been proposed for the problem such as optimal control, DE and TLBO. Padhy and Panda (2017) concluded that when hSFS-PS optimized PI-PD controller a better system response is observed that when PI-PD controller is optimized by the compared algorithms.

Rahman et al. (2017a) improved SFS by embedding two chaos maps (namely, Chebyshev map and Gauss/Mouse map) into the algorithm. The resulting algorithm is called Chaotic Fractal Search (CFS). Both benchmark test functions and optimal tuning of PID controller and PD-type fuzzy logic controllers are used in the computational study. By analyzing the results of the computational study, Rahman et al. (2017a) observed that CFS that use Gauss/Mouse map in Diffusion Process outperforms SFS. Also, another observation is that embedding the chaos maps into first updating process lowers the run-time.

Rahman et al. (2017b) studied CFS from (Rahman and Tokhi, 2016) and used vibration control to suppress the unwanted vibration from a flexible beam structure to evaluate the performance of the algorithm to solve the problem. At the end, it is concluded that CFS can present a stable and adequate model.

Sivalingam et al. (2017) presented a hybridization of SFS and LUS (Pedersen 2010, local unimodal sampling) algorithms called SFS-LUS for multistage PID controller consisting of PDF plus for AGC of power systems. To compare the proposed algorithm, DE and TLBO results for the same problem are used. Sivalingam et al. (2017) concluded that SFS performs better than DE and TLBO and SFS-LUS hybridization performs better than SFS, DE and TLBO.

Alomoush and Oweis (2018) studied a non-linear problem that is environmental economic dispatch problem in power systems operations under real life constraints and solved the problem using SFS algorithm. In the computational tests, common test systems are used, and three scenarios (first scenario consists of a 6-unit system and a 15-unit system, second scenario consists of a 6-unit system and a 10-unit system, finally the third scenario is IEEE 30-bus power system) are formed. The comparisons are made between various algorithms and SFS. However, it is worth mentioning that in the study, different forms of the algorithms that are taken from various studies in the literature are used and, in the comparisons, not all algorithms are compared for the same scenario. More specifically, for the first scenario, SFS is compared with PSO, GA presented in Ciornei and Kyriakides (2013) and Kuo (2008). For the second scenario, SFS is compared with GSA (presented in Güvenç et al., 2012), GA (presented in Güvenç, 2010) and all the following algorithms are taken from Balamurugan and Subramanian, 2007; λ -iteration approach, recursive approach, PSO, DE, a simplified recursive approach, GSA and

finally, all the following algorithms are taken from Basu, 2011; MODE (multi-objective DE), PDE (Pareto frontier DE), NSGA-II (nondominated sorting based multi objective evolutionary algorithm), and SPEA 2 (strength Pareto evolutionary algorithm 2). For the third scenario GSA (Duman et al., 2012), GA (Lai et al., 1997), gradient method (Lee et al., 1985), ABC (Herbadji et al., 2013b), two BBOs (Bhattacharya and Chattopadhyay, 2011 and Herbadji et al., 2013a), three DEs (Sayah and Zehar, 2008, Vaisakh and Srinivas, 2011a, and Vaisakh and Srinivas, 2011b), two PSOs (Niknam et al., 2012, Sivanagaraju et al., 2014) are used. Alomoush and Oweis (2018) concluded that SFS gives a better performance in comparison to considered algorithms in a low run-time.

Çelik (2018) used SFS to tune the parameters of PID structure in AVR systems. In the computational tests, ABC, MOL (Pedersen and Chipperfield, 2010, many optimizing liaisons), LUS, WCO (Razmjooey et al., 2016, world cup optimization), BBO, GSA is used for comparative purposes. Çelik (2018) stated that when simulation results analyzed if PID controller parameters are tuned by SFS, a better dynamic response is observed.

Dubey et al. (2018) prepared an overview of nature inspired algorithms that are used for wind integrated multi-objective optimal power dispatch problem and compared their performances using multi criteria decision making concepts. Some standard cases are selected and the performance of the selected algorithms which are SFS, FPA, MBA (Sadollah et al., 2012, mine blast algorithm), BSA, SOS (Cheng and Prayogo, 2014, symbiotic organisms search), ALO (Mirjalili, 2015a, ant lion optimizer), MFO (Mirjalili, 2015b, moth-flame optimization) and LSA (Shareef et al., 2015, lightning search algorithm) are compared. Dubey et al. (2018) concluded that FPA BSA and ALO give the best results for the tested problems, but the other five algorithms are also promising.

Khalilpourazari and Khalilpourazary (2018) studied the optimization of grinding process of machining processes. The problem is modelled as a non-linear model that has three objective functions under some operational constraints. The main objective in the study is to optimize the grinding process and minimize the total process cost. They presented a modified version of SFS to solve the problem in that Taguchi method tunes the parameters of SFS which is called RSFS (robust SFS). In the computational study, an experimental example is solved, and comparisons are made with other methods from the literature which are QP (Wen et al., 1992, micro-computer-based optimization), GA, ACO (Dorigo

and Di Caro, 1999, ant colony optimization), SS (Glover, 1977, scatter search), DE, EP-PSO (Lin and Li, 2008, enhanced pareto PSO), EA (Slowik and Slowik, 2008, evolutionary algorithm with remembered Pareto set), PSO, TSBDEA (Lee et al., 2011, the Taguchi-sliding-based DE), HPSO (Zhang et al., 2014, a hybrid PSO), MPEDE (Lynn and Suganthan, 2015, multi-population ensemble DE), HCLPSO (Mallipeddi et al., 2016, heterogeneous comprehensive learning PSO). Khalilpourazari and Khalilpourazary (2018) concluded that RSFS gives a better performance than the compared methods.

Khishe et al. (2018) used a version of SFS that integrates chaotic maps into the algorithm called CFWT (chaotic fractal walk trainer) to train multi-layer perceptron neural network in order to design the evolutionary classifier. In the computational tests, benchmark and sonar datasets are used. Also, comparisons are made between CFWT and PSO, GA, ACO, ES. Khishe et al. (2018) concluded that SFS provides better results for run-time, classification accuracy, and exploration/exploitation. When chaotic maps are embedded into the algorithm CFWT gives even better results than SFS.

Nguyen and Vo (2018) solved the allocation of distributed generators problem in radial distribution systems that has various objectives and constraints using SFS algorithm. IEEE 33-bus, 69-bus, and 118-bus problems are solved for four different scenarios and for each scenario three different cases. In the computational study, and comparisons are made between SFS and KHA (Gandomi and Alavi, 2012, krill herd algorithm), LSFSA (Injeti and Kumar, 2013, a loss sensitivity factor SA), TLBO, QOTLBO (Sultana and Roy, 2014, quasi-oppositional teaching-learning based optimization), SIMBO-Q (Pattnaik et al., 2013, swine influenza model based optimization with quarantine), QOSIMBO-Q (Sharma et al., 2016, quasi-oppositional swine influenza model based optimization with quarantine), GA, AGA (Ganguly and Samajpati, 2017, an adaptive GA), MINLP (Kaur et al., 2014, a mixed integer non-linear programming), and IAA (Hung and Mithulanathan, 2011, an improved analytical approach), PSA, GA-PSO (Moradi and Abedini, 2012), IWD (Shah-Hosseini, 2009, intelligent water drop algorithm), GA-IWD (Moradi and Abedini, 2016, a hybridized GA and IWD), IMOHS (Nekooei et al., 2013, an improved multi-objective HS), SOS, CSOS (Saha and Mukherjee, 2016, a chaotic SOS), ICA-GA (Moradi et al., 2014, a hybridized imperialist competitive algorithm and GA), ORCSA (Khoa et al., 2017, one rank CS). It is worth mentioning that all these algorithms are not compared for every single scenario and case,

but they are used separately and partially in the comparisons. The algorithms are used in the literature to solve the presented problem. Nguyen and Vo (2018) concluded that SFS performs better than many other algorithms particularly for the large-scale systems and it has reliable convergence.

Nguyen et al. (2018) studied the problem of reducing the power loss in distribution systems and used a chaotic SFS (CSFS) to solve the problem. In CSFS, ten chaotic maps (namely, Chebyshev map, Circle map, Gauss/mouse map, Iterative map, Logistic map, Piecewise map, Sine map, Singer map, Sinusoidal map, Tent map) are embedded into SFS separately to improve the randomness the original SFS presents. In computational testes, IEEE 33-bus, 69-bus, and 118-bus radial distribution systems are used. The comparisons are made between CSFS and SFS, PSO, EA (Fogel et al., 1966, Rechenberg, 1973, Holland, 1975, Goldberg, 1989, evolutionary algorithm. EA consists of a collective set of principles that was developed over time, therefore, it is impossible to attribute the beginning of EA to a single research.), EA-OPF (Mahmoud et al., 2015, EA with optimal power flow), EOPF (Mahmoud et al., 2015, exhaustive optimal power flow), AM-PSO (Kansal et al., 2016, a PSO that use an analytical method), ELF (Hung et al., 2011, exhaustive load flow), IA, LSF (Hung et al., 2011, loss sensitivity factor), BSA, REPSO (Jamian et al., 2015, a rank evolutionary PSO), GA, GA-PSO (Moradi et al., 2012), IWD, GA-IWD (Moradi et al., 2016), SA, KHA, BFA (Passino, 2002, bacterial foraging optimization algorithm), TLBO, QOTLBO, HSAPABC (Muthukumar and Jayalalitha, 2016, a hybridized HS and particle artificial bee colony algorithm) that are used in the literature to solve the presented problem. It is concluded that CSFS that use Tent map gives the best performance.

Zhou et al. (2018) studied the prediction of three-dimensional structures of proteins based on the known amino acid sequences. The problem is solved by an improved version of SFS that uses Levy flight and internal feedback information to avoid falling into local minimum easily by measuring the state of the convergence performance. Zhou et al. (2018) concluded that ISFS is an accurate and efficient algorithm in predicting protein structure.

Betka et al. (2019) aimed to make block matching (BM) algorithm faster by hybridizing BM with SFS (SFS-BM) to solve movement prediction problem. In the study a modified

version of SFS (that uses a fixed pattern to generate the initial solution instead of generating the initial solution randomly, the initial solution are evaluated by a fitness function, search space is managed by an adaptive window size strategy, also a fitness approximation method that reduces run-time while sacrificing some accuracy) is hybridized with BM (MSFS-BM). The comparisons are made between SFS-BM, MSFS-BM and FSA (Choudhury and Saikia, 2014, full search algorithm), PSO-BM (Yuan and Shen, 2008, a hybridization of PSO and BM), DE-BM (Cuevas et al., 2013, a hybridization of DE and BM), HS-BM (Díaz-Cortés et al., 2017, a hybridization of HS and BM), PLA-BM (Damerchilu et al., 2016, a hybridization of parameterized learning automata and BM), TPLA-BM (Damerchilu et al., 2016, a hybridization of team of pursuit learning automata and BM), TSS (Koga, 1981, three step search), NTSS (Li and Liou, 1994, new three step search), SES (Lu and Liou, 1997, simple and efficient three step search), 4SS (Po and Ma, 1996, a novel four step search), DS (Zhu and Ma 2000, a novel diamond search). Betka et al. (2019) concluded that MSFS-BM outperforms the compared algorithms while FSA comes as second.

Çelik and Gör (2019) aimed to improve the process control systems by efficiently taking over the workload. The usual method to deal with the problem is using PID controller. In the study, a new method is introduced called PI+DF (proportional integral derivative path first order) controller. The necessary parameters are tuned by using SFS in this control method. The comparisons are made between SFS-PI+DF and PSO-PI+DF and a control scheme called SOS-PI (Çelik and Öztürk 2018, SOS tuning proportional integral controller) to test the efficacy of SFS. Çelik and Gör (2019) concluded that SFS-PI+DF give better results than PSO-PI+DF.

Lin and Wang (2019) solved the multi-area economic dispatch problem of power system operations by using an improved version of SFS called ISFS. In ISFS, initial solution and neighbors are generated by an opposition-based learning method. Also, DE is incorporated into the diffusion process of SFS. Lastly, a novel repair-based penalty approach is used to find feasible solutions. Comparisons are made between ISFS and HSLSO (Ghasemi et al., 2016, hybridizing sum-local search optimizer), CGBABC (Secui, 2015, chaotic global best ABC) and SFS. Lin and Wang (2019) concluded ISFS performs better than the compared algorithms in solving the problem.

Mahdad (2019) applied FS to solve optimal reactive power dispatch problem that is used to optimize the power quality the customer receives in cases of load growth and faults. The specific condition the study considers is overloads resulting from increase in air-conditioning usage in the summertime. The comparisons are made between FS and IP (interior point algorithm), SA, GSA. Mahdad (2019) concluded that FS give efficient results which are robust, particularly in terms of performance and run-time.

Nguyen et al. (2019) used a modified SFS called MSFS to solve reactive power dispatch optimization. The modification is to insert three methods to update solutions more efficiently. In the first method the diffusion process, in the second and third method updating process is affected by the inserted methods. The computational study is done by using single objective IEEE bus system problems in sizes 30 and 118 and comparisons are made between MSFS and GSA, TLBO, QOTLBO, ALO, SSO, MFO with the addition of various modified and hybridized versions of PSO, GA, DE. Nguyen et al. (2019) concluded that MSFS responds better to Gaussian walk rather than Levy flight. In comparison MSFS gives better results, is more robust and has lower run-time.

Pham et al. (2019a) studied the economic load dispatch problem and used two variations of SFS that make small modifications to the original SFS to solve the problem. The variations differ in that one uses Levy flight and the other uses Gaussian random walk only in the diffusion process. In the computational tests three test systems in sizes 6, 10 and 20 are used. Comparisons are made with results from the literature with BBO, ALO and various modified versions of FA (Yang, 2008, firefly algorithm), PSO, DE, GA. Pham et al. (2019a) concluded that SFS solves the presented problem effectively.

Pham et al. (2019b) solved economic load dispatch problem using a modified stochastic fractal search algorithm (MSFS) that makes small changes in solution updated procedure. The computational tests are executed by using test problems in various sizes: 3, 6, 10, 20. Comparisons with MFSF and SFS, BBO, MOEP (Jeyakumar et al., 2007 a multi objective evolutionary programming algorithm), CSA (Gandomi et al., 2013 a cuckoo search algorithm), ICSA (Tran et al., 2016 an improved cuckoo search algorithm) NSGA-II (Deb et al., 2000), FA, ITS (Lin et al., 2002, an improved TS), MPSO (Park et al., 2005), FCASO-SQP (Cai et al., 2012, a hybridization of fuzzy adaptive chaotic ant swarm optimization and SQP), PSO-RDL (Chen et al., 2007, a hybridization of PSO with

dynamic linkage discovery), SDE (Reddy and Vaisakh, 2013, shuffled DE). Pham et al. (2019b) concluded that MSFS performs better than SFS.

Tran et al. (2019) used a modified SFS called ISFSOA to solve optimal reactive power dispatch problem with three separate objectives. The modification concerns the diffusion process of SFS. In the computational study, an IEEE 30-bus system with three case studies are used and comparisons are made between ISFSOA and SFS, some hybridizations of PSO that are taken from the literature, TS, ALO, QOTLBOM, TLBO, SGA, HS. Tran et al. (2019) concluded that ISFSOA performs better than SFS and finds comparable or better solutions in comparison to other algorithms.

Alomoush (2020) solved the highly non-linear combined heat and power generation economic dispatch problem using SFS. Three different cases are used in the computational study and comparisons are made using previously reported results in the literature between Lagrangian relaxation, branch-and-bound algorithm, IACS (Haghrach et al., 2016, improved ACO), GAPF (Subbaraj et al., 2009, GA-based penalty function), PSO, EA, FA, IGAMU (Subbaraj et al., 2009, improved GA with multiplier updating), HS, SARGA (Subbaraj et al., 2009, self-adaptive real-coded GA), ABC, DE, MADS-PSO (Nguyen et al., 2016, mesh adaptive direct search and PSO), MADS-LHS (Nguyen et al., 2016, mesh adaptive direct search and Latin hypercube sampling), IWO, GSA, CSA, CSOS and SFS. Alomoush (2020) concluded that SFS is a promising algorithm for global optimization and it can outperform some well-known algorithms.

Çelik (2020) worked on a novel cost function for AGC performance of interconnected multi-area power system and used ISFS (which adds chaotic maps, namely logistic, piecewise linear, iterative, sine maps, to the algorithm) to solve six benchmark problem. The comparisons are made between SMS (Cuevas et al., 2014, states of matter search), BA (Yang, 2008, bat algorithm), FPA, CS, FA, ALO and four ISFS variants that embed four chaos maps separately. It is concluded that the piecewise linear map to be the best.

Khalilpourazari et al. (2020) solved complex multi-objective optimization problems which are nine CEC 2009 (IEEE Congress on Evolutionary Computation) problems and an engineering design problem called welded beam design problem, using a novel multi-objective stochastic fractal search (MOSFS) algorithm. The comparisons are made between MOSFS, MOPSO (Coello and Lechuga 2002, multi-objective PSO), MOEA/D

(Zhang and Li 2007, multi-objective EA based on decomposition), MOGWO (Mirjalili et al. 2016, multi-objective GWO) and NSGA-III (Deb and Jain 2014, non-dominated sorting GA-III). Khalilpourazari et al. (2020) concluded that MOSFS gives the most promising results compared to other algorithms.

Table 2.2. Literature review of Fractal Search and Stochastic Fractal Search

Author(s), Year	Pure	Hybrid	Comparison	Improvement	Heuristic	Problem	Application Area
Sohal et al. (2015)	x				SFS	Query optimization	Distributed databases
Awad et al. (2016)		x	x		DE+SFS (dDSF-EA)	Benchmark functions	IEEE CEC2014
Khalilpourazari et al. (2016)	x		x		SFS,	Multi product economic production quantity optimization	Production
Li (2016)	x				SFS	Numerical optimization	Image retrieval
Mellal and Zio, (2016)			x	x	PSFS	Benchmark problems	Reliability and redundancy allocations
Mosbah and El-Hawary, 2016	x		x		FS	Tracking state estimation	Power systems operations
Rahman and Tokhi, 2016				x	CFS	Benchmark functions	-
Chuan et al., 2016	x				SFS	Protein structure prediction	Disease treatment and drug research and development
Tejani et al., 2016			x		SFS	Design optimization	Steel and Construction
Tyagi et al., 2016			x	x	ISFSA	Multi-objective optimal dispatch	Power systems operations
Ashraf et al., 2017		x	x		SFS+BBO(HBBO)	Distribution and control of goods	Supply chain management
El-Fergany and Hasanien, 2017	x		x		SFS	Optimal relay coordination	Meshed power networks
Khanam and Parmar, 2017a	x				SFS	Order reduction of large scale single input single output linear time invariant systems optimization	Control systems
Khanam and Parmar, 2017b	x		x		SFS	Parameter tuning of PID controller	Electronics
Lin and Wang, 2017		x			HSFS	Parameter identification	Control & synchronization of fractional order chaotic systems
Mosavi et al., 2017	x				SFS	Training radial basis function	Neural networks
Mosbah and El-Hawary, 2017	x		x		SFS	Multilayer perceptron neural network parameters optimization (weights and thresholds)	Neural networks
Padhy and Panda, 2017		x			SFS+PS(hSFS-PS)	Automatic generation control optimization	Power systems operations
Rahman et al. 2017				x	CFS	Benchmark functions, optimal tuning of conventional PID and PD-type fuzzy logic controllers	Twin rotor system control design

Table 2.2. (cont'd) Literature review of Fractal Search and Stochastic Fractal Search

Author(s), Year	Pure	Hybrid	Comparison	Improvement	Heuristic	Problem	Application Area
Rahman et al., 2017b				x	CFS	Suppressing the unwanted vibration from a flexible beam structure	Active vibration control
Sivalingam et al., 2017		x	x		SFS+LUS(SFS-LUS)	Automatic generation control	Power systems operations
Alomoush and Oweis, 2018	x		x		SFS	Environmental economic dispatch problem	Power systems operations
Çelik, 2018	x				SFS	Parameter tuning of PID controller for an automatic voltage regulator system	Power systems operations
Dubey et al., 2018	x		x			Wind integrated multi-objective power dispatch optimization	Power systems operations
Khalilpourazari and Khalilpourazary 2018			x	x	RSFS	Optimization of the grinding process	Machining processes
Khishhe et al., 2018				x	CFWT	Benchmark dataset, high-dimensional practical sonar dataset, filed programmable field array	Sonar data set classification
Nguyen and Vo, 2018	x				SFS	Optimal allocation of distributed generators	Radial distribution systems
Nguyen et al., 2018			x	x	CSFS	Reducing the power loss	Distribution systems
Zhou et al., 2018				x	ISFSA	Protein structure prediction	Biological information research, disease treatment, and drug development
Betka et al., 2019		x		x	BM+SFS (SFS-BM) and MSFS-BM	Motion estimation	Computer vision
Çelik and Gör, 2019	x		x		SFS	Improving the performance of PID controller	Electronics
Lin and Wang, 2019				x	ISFSA	Multi-area economic dispatch problem	Power systems operations
Mahdad, 2019	x				FS	Optimal reconfiguration, optimal location, and sizing	Reactive power planning
Nguyen et al., 2019			x	x	MSFS	Optimal dispatch of reactive power, power loss optimization, voltage deviation optimization, and L-index optimization	Power systems operations
Pham et al., 2019a			x	x	SFS	Economic load dispatch problem	Power systems operations
Pham et al., 2019b			x	x	MSFS	Economic load dispatch problem	Power systems operations
Tran et al., 2019				x	ISFSA	Optimal reactive power dispatch problem	Power systems operations
Alomoush, 2020	x		x		SFS	Economic dispatch problem	Power systems operations
Çelik, 2020			x	x	ISFS	AGC performance of multi-area power system	Power systems operations
Khalilpourazari et al., 2020			x	x	MOSFS	Benchmark problems, welded beam design problem	Engineering design

2.3. An Evaluation of the Literature Review

Literature reviews on neighbor generation mechanisms and FS, SFS are given in Sections 2.1 and 2.2, separately. In Section 2.1, when the resulting literature is observed, common neighbor generation mechanisms (not problem specific ones) are swap, insertion, reversion and different variations and combinations of these neighbor generation mechanisms. Although there are studies on comparing mentioned neighbor generation mechanisms for various problems, there are not many studies on designing and improving neighbor generation mechanisms and it is important to note that efficiency and effectiveness, obtained from embedding different existing neighbor generation mechanisms into an algorithm for solving different problems, change drastically. The presented study is one such study, since a neighbor generation mechanism is designed and tested for two benchmark problems which are TSP and QAP. It is important to remind the reader that TSP and QAP are selected because of their different characteristics in their solution spaces and the focus is to design a neighbor generation mechanism that can solve both problems efficiently enough. In Section 2.2, heuristic algorithms that use fractal principles called FS and SFS are explained and their usages in the literature are shared. FS and SFS are comparatively new and gained popularity amongst researchers over the past few years. However, even though fractal principles are used in the heuristics, they are population-based heuristics that are used for solving continuous and mostly non-linear problems. The presented research proposes a neighbor generation method which use fractal principles that is used in single solution-based heuristics and it is used to solve combinatorial problems. Hence, the motivation and the objective in design of CB mechanism and usage of CB mechanism are different from FS and SFS algorithms. Moreover, studies on designing neighbor generation mechanisms are very limited in the literature. Usually, existing mechanisms are tailored to be suitable to solve the discussed problem. Therefore, the presented study greatly differs from the literature as the motivation in designing CB mechanism is to find a design that is more generally usable in solving problems with different characteristics.

3. BASIC CONCEPTS IN SEARCH ALGORITHMS

In this section, the basic concepts that are used in the study are explained in detail. In the presented study, a neighbor generation mechanism, called CB mechanism, that can be embedded into single solution-based heuristics is designed. Then, a comparative analysis between different variations of CB and two neighboring mechanisms from the literature, namely swap and insertion, is done to find an efficient design for CB.

In the computational tests, to examine the performance of CB, two single solution-based heuristics which are LS and SA algorithms are employed. They are chosen because they are very basic, unsophisticated algorithms that can be applied to a wide variety of computationally hard problems. LS is a deterministic algorithm and SA is a stochastic algorithm which will help to examine both deterministic and stochastic performances of the CB mechanism.

Two combinatorial optimization problems, namely TSP and QAP are solved to investigate the performance of CB mechanism. TSP and QAP are chosen because of their differing characteristics, such that TSP has a steep landscape in its solution space and QAP has a flat landscape in its solution space.

In section 3.1. basic knowledge on algorithms is shared and brief explanations of LS and SA algorithms are given. In section 3.2. TSP and QAP are explained and their mathematical models are given. In section 3.3. basic information on fractal geometry and cantor sets are shared.

3.1. Introduction to Algorithms

Algorithms are special procedures to solve problems that can also be implemented to a computer. Hence, they do not find precise answers but rather they give a set of rules to follow in order to find an answer. A legitimate algorithm does not work most of the time, but it works for all feasible inputs all the time.

Computation theory studies whether a given problem is computationally solvable or not while using possibly appropriate computation methods. *Complexity theory* studies the resources which are “time” (steps needed to solve the problem) and “memory” (space needed for the algorithm) required to compute an algorithm. Complexity of a problem depends on the complexity of the best algorithm that can solve that problem.

Computational Complexity aims to calculate the difficulty to solve the problem and as a result gives insight on how to solve problems. Complexity of problems are categorized as P (the problem is solvable in polynomial time) and NP (not determined if the problem is solvable in polynomial time) (Edelkamp and Schroedl, 2011). Relationship between the complexity classes can be seen in Figure 3.1. Notice that NP -complete, which are the problems that are proven to be not solvable in polynomial time are separated from the problems that are solvable in polynomial time.

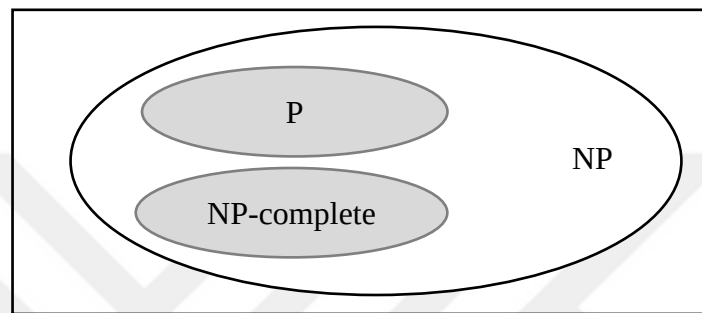


Figure 3.1 Complexity classes

No-free lunch theorem suggests that there is not a “best-suited” metaheuristic algorithm that can solve all optimization problems. An algorithm may show satisfactory results on a set of problems or even on some problem instances. However, it can produce poor results for other problems, or different instances of the same problem. *Analysis of algorithms* usually means an investigation of an algorithm’s efficiency with respect to run-time and memory. Unlike simplicity and generality, efficiency can be measured in quantitative terms.

3.1.1. Search methods

Optimization problems can be formulated as mathematical models, combinatorial optimization models, constraint satisfaction models and non-analytical models based on their characteristics. Search methods that are used to solve optimization problems are given in a hierarchical way in Figure 3.2. Notice that, in the figure, only the branch that concerns the explanations of this section is advanced further.

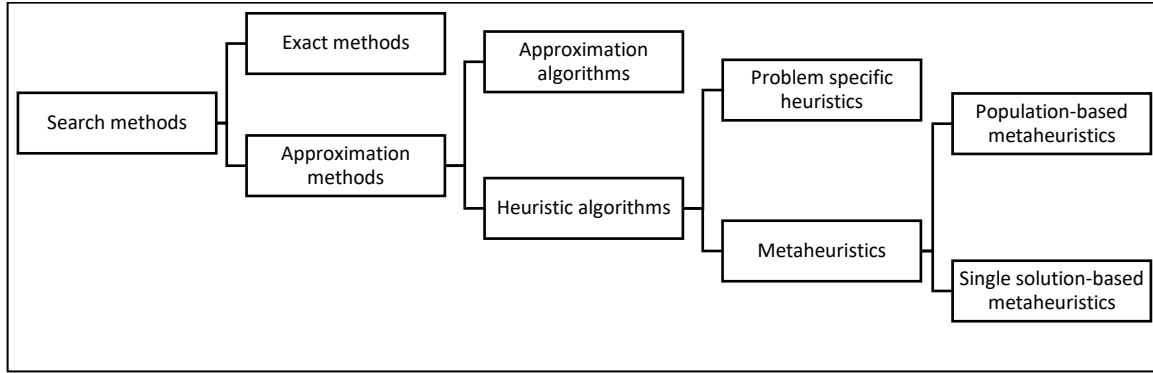


Figure 3.2 Search methods for optimization (Talbi, 2009)

Exact algorithms guarantee to produce the optimal solution in a finite amount of time for the problems in P -class or small / moderate sized instances of NP problems. Some examples for exact algorithms are constraint programming, dynamic programming, branch and bound algorithm etc. *Heuristic algorithms* are used when exact methods cannot find the solution in a reasonable amount of time. Heuristics explore large solution spaces of hard problems methodically. The purpose of heuristics is to solve large problems, to solve problems faster, and to be robust. Heuristics are categorized as problem specific heuristics which are designed for specific problems and metaheuristics which are designed to solve a variety of problems. There are several classifications of metaheuristics. One of them separates single solution-based metaheuristics from population-based metaheuristics. In *single solution-based metaheuristics*, the search starts from a single solution and a single solution is manipulated throughout the search to find an optimal solution.

The most important factor in a heuristic algorithm is to find a good balance between efficiency (i.e. exploration, diversification) and effectiveness (i.e. exploitation, intensification). *Efficiency* ensures that the algorithm can visit sufficiently sparse areas of the search space. As a side note, efficiency of an algorithm does not guarantee optimality in the found solution. The optimality is directly affected by the complexity of the concerned problem. *Effectiveness* is searching within the previously visited regions that had most promising solutions more thoroughly in the hope of finding better solutions. The balance between efficiency and effectiveness is a key issue of the overall search performance. This balance often relies on the adjustment of solution representation (encoding), generating the initial solution, neighbor generation method (i.e. move

operator, variation operator, transformation, manipulation), solution acceptance and stopping condition which are the important points that differentiates single solution-based algorithms.

Solution representation (Encoding) is the first step to solve a problem with an algorithm. Solution representation must represent all feasible solutions for the problem, it has to be easy to manipulate and feasible solutions in the search space must be connected. When there are two options in a problem, like yes-no, exists-doesn't exist, put it-don't put it etc., *binary solution representation* is used generally. There can also be more than two options in a problem, calculating base- n value of a number would represent the problem which is called *n -nary solution representation*. Popular uses include knapsack problem, assignment problem etc. Two examples are given as follows.

- Decide which letters from the set $\{A, B, C, D, E\}$ to put into a backpack. A solution representation, $[10101]$ is interpreted as to put A, C, E and not to put B and D.
- Find a numerical solution to a specified problem. A candidate solution can be $[10]_{10}$ or $[1010]_2$. Neighbors of 10 are 9 and 11. Neighbors of 1010 are 1110, 1011, 0010, 1000. Even though $[10]_{10}$ and $[1010]_2$ represent the same value, the number of neighbors that are connected to them and the value of the neighbors are different.

Permutation solution representation is used when n possibilities need to be ordered. Popular uses include sequencing problems, traveling salesman problem, scheduling problems etc. An example is given as follows.

- Find the order of 5 machines. A candidate solution can be $[5\ 3\ 4\ 2\ 1]$, and the interpretation is the machines will be operated in order 5-3-4-2-1.

Vector of real or discrete values solution representation consists of a vector of discrete or real values. Popular uses include continuous optimization, parameter identification, global optimization etc. An example is given as follows.

- Find a numerical solution to a specified problem. A candidate solution can be $[3\ 2\ 4\ 5.2\ 1.4\ 6.1]$, which means $x_1 = 3, x_2 = 2, x_3 = 4, x_4 = 5.2, x_5 = 1.4, x_6 = 6.1$.

Generation of an initial solution can be done in various ways. Every problem has its specific needs, and every search algorithm has its features. By analyzing them, a designer decides how to generate the initial solution in an algorithm. Initial solutions can be generated *randomly* which provides deviation in obtained solutions and is quick to converge while often giving worse quality local optima. Initial solutions can also be generated by a *greedy heuristic* that meet the needs of the problem. Also, a hybridization of different approaches can be used. When choosing a method, solution quality and run-time need to be considered.

Neighbor generation mechanism is defined as pre-defined rules to make small perturbations on solution vector $\mathbf{x}$ to generate a set of neighboring solutions $\mathbf{x}'$. Neighborhood of a solution $\mathbf{x}$ is the set of neighbors $\mathbf{x}' \in N(\mathbf{x})$. $N(\mathbf{x}) \subset S$ where S is the solution space. In the literature of single solution-based heuristics, there are mechanisms that are designed for a specific problem. There are also widely used neighbor generation mechanisms such as swap, insertion, and reversion. In *swap*, positions of two elements are exchanged, in *insertion*, an element is inserted to a different position and in *reversion*, a subsequence of elements is reversed. An example for each mechanism is given as follows.

Swap: $7 - [6] - 5 - 4 - 3 - [2] - 1 \rightarrow 7 - [2] - 5 - 4 - 3 - [6] - 1$

Insertion: $7 - [6] - 5 - 4 - 3 - 2 - 1 \rightarrow 7 - 5 - 4 - 3 - 2 - [6] - 1$

Reversion: $7 - [6 - 5] - 4 - 3 - 2 - 1 \rightarrow 7 - [5 - 6] - 4 - 3 - 2 - 1$

In Section 5, the results obtained from the algorithms when swap and insertion mechanisms are embedded separately, are used as a comparative study. Swap generates $\frac{n(n-1)}{2}$ neighbors while insertion generates $n(n-1)$ neighbors for an n -sized problem.

Solution acceptance is decided at design stage of the algorithm and dictates how a solution $\mathbf{x}$ is chosen amongst the neighbors $\mathbf{x}'$. There are multiple principles used by local search-based algorithms in terms of solution selection based on the algorithm. Firstly, one solution can be iterated, multiple solutions can be iterated, or one/multiple solutions can be iterated by using multiple neighbor generation methods. Secondly, there can be many principles on choosing a neighbor as follows.

- *Select a random solution.* Partial or whole neighborhood is generated and a “better” solution is selected randomly. Gives variation to the solutions and provides fast convergence to local optima while being unpredictable about the solution quality.
- *Select the first better solution encountered.* Partial or whole (in the case of not finding a better solution ‘until the end’ or ‘at all’) neighborhood is generated and the first encountered “better” solution is selected. Gives variation to the solutions and usually provides faster convergence to local optima than selecting the best solution.
- *Select the best solution in the neighborhood.* Whole neighborhood is generated, and the best solution is selected.
- *Select a better solution but accept worse solutions with a probability.* A neighbor is generated randomly. Used in some algorithms to not get trapped at local optimum.

Stopping Condition is decided at design stage of the algorithm and determines when the algorithm stops. Some examples are given as follows.

- Stop when ‘the first time’ or ‘for a specified number of times’, there is not any better solutions in the neighborhood.
- Stop when a pre-defined number of iterations are executed.
- Stop when a predefined run-time has passed.
- Stop when the solution is within in a pre-determined acceptable zone.

3.1.2. Local search algorithm

Local Search (LS) is the most basic single solution-based heuristic. In general, the search starts from an initial solution. That solution is manipulated by a neighboring mechanism to generate a whole or partial neighborhood. The neighborhood is evaluated. Then, a solution is picked by a selection strategy. The algorithm stops when a stopping condition is fulfilled. Designer chooses the selection strategy and stopping condition by considering their advantages.

More specifically, let x be a feasible solution and let X be the set of feasible solutions

which makes $\mathbf{x} = [x_1, x_2, \dots, x_n]$, $\mathbf{x} \in \mathbf{X}$. Let $\mathbf{x}'$ be a neighbor that can be generated from $\mathbf{x}$ and let $N(\mathbf{x})$ be the set of $\mathbf{x}'$. Let $f(\mathbf{x})$ be objective function value of $\mathbf{x}$. In this study, the best improvement is used as the selection strategy and the algorithm stops the first time there is no better solution among the explored neighbors. A pseudo code for LS is given in Figure 3.3.

-
1. **Initialization:** Generate an initial solution $\mathbf{x}$ ($\mathbf{x} \in \mathbf{X}$) and $\mathbf{x}_{best} = \mathbf{x}$
 2. Calculate $f(\mathbf{x})$ and $f(\mathbf{x}_{best}) = f(\mathbf{x})$
 3. **Repeat**
 4. **Generate** $N(\mathbf{x})$ and calculate $f(\mathbf{x}')$ for every neighbor $\mathbf{x}'$
 5. **Chose** minimum $f(\mathbf{x}') = \min f(\mathbf{x}')$
 6. **If** $\min f(\mathbf{x}') < f(\mathbf{x})$ **Then** $\mathbf{x}' \leftarrow \mathbf{x}$
 7. **End if**
 8. **If** $f(\mathbf{x}') < f(\mathbf{x}_{best})$ **Then** $\mathbf{x}' \leftarrow \mathbf{x}_{best}$
 9. **End if**
 10. **Until** $f(\mathbf{x}') \geq f(\mathbf{x})$
 11. **Output:** $\mathbf{x}_{best}$ and $f(\mathbf{x}_{best})$
-

Figure 3.3 Pseudo code of LS heuristic

It must be noted that, it is very hard to guess the quality of the local optimal solutions meaning, it is hard to know if LS which is being used is a good algorithm or not. Because in LS, the search easily gets trapped in local optima. In Figure 3.4. objective function values of feasible solutions in an example search space is given. Points A, B, C and D are local optima for minimization objective function as they are the lowest points in their respective valleys, consequently they are the final points that can be reached iteratively in a decreasing direction. In LS, a neighbor can be picked by random selection, best improvement or first improvement strategies. All these strategies seek to encounter a better solution than the current solution. The search stops when a better solution cannot be found. Therefore, if the search gets stuck at point A, B or C, there is no way to climb the valley in an increasing direction to find a better local optimum. Therefore, there is no way to get to point D, which is the optimal solution, without accepting solutions in another way. In the literature, different strategies like accepting worse solutions, doing the search with multiple solutions, using multiple neighbor generation methods, and changing the problem data are used in various algorithms to challenge local optimality traps.

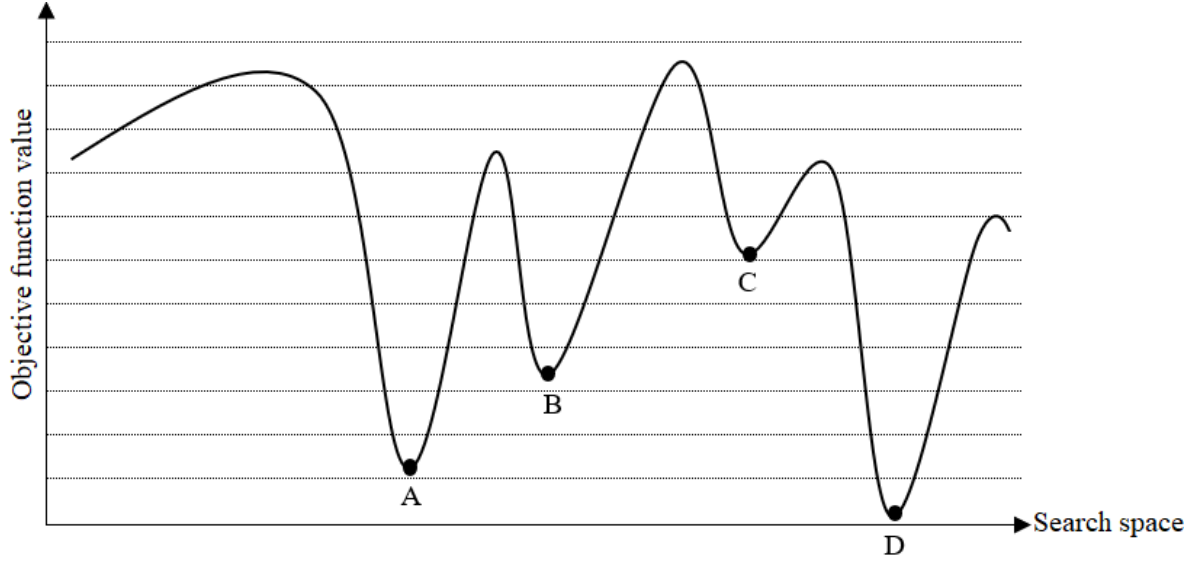


Figure 3.4 Objective function values of feasible solutions in a search space

3.1.3 Simulated annealing algorithm

Simulated Annealing (SA) is proposed by Kirkpatrick et al. (1983) and uses solid annealing process from statistical mechanics as its analogy. SA uses probabilistic neighbor generation and selecting strategy. Specifically, SA algorithm accepts better solutions as they come along. However, in the course of the run, if the algorithm encounters a worse solution, it accepts that solution with a *Probability of Acceptance* which is given in equation (3.1). When $p(\Delta f, T) > U [0,1]$, a worse solution is accepted. Consequently, the probability of accepting a solution decreases as the iterations increases.

$$p(\Delta f, T) = e^{-\frac{\Delta f}{T}} \quad (3.1)$$

In equation (3.1), $\Delta f = f(x') - f(x)$ in which f is the minimization objective function, x and x' are the current and neighbor solutions, respectively, and T is the temperature parameter.

The temperature parameter is decreased according to a cooling procedure as the algorithm proceeds to force searching for better solutions. It is called a *cooling schedule* because in the analogy of SA, this refers to the slow cooling of a solid until it reaches its low energy ground state. There are various cooling schedules in the literature. Some simple cooling schedules presented in Kirkpatrick et al. (1983) are given in equation (3.2) and (3.3).

$$T_i = \left(\frac{T_1}{T_0}\right)^i T_0 \quad (3.2)$$

$$T_{i+1} = T_i \times \alpha \quad (3.3)$$

Equation (3.2) is an exponential schedule where $\frac{T_1}{T_0}$ usually equals to 0.9. Equation (3.3) is a simplified version of equation (3.2), and it is called geometric cooling schedule. In the literature α is generally set to a value between 0.85 to 0.99. It is also important to mention that the cooling schedule should provide equilibrium state (ϑ) at each temperature level. ϑ is the number of iterations which temperature T remains the same before updating T . ϑ value is generally used as a function of n .

A linear schedule given in Fielding (2000), Nourani and Andresen (1998), Cohn and Fielding (1999) is given in equation (3.4).

$$T_i = T_0 - \eta i \quad (3.4)$$

A schedule presented in Aarts and Van Laarhoven (1985) is given in equation (3.5).

$$T_{i+1} = \frac{T_i}{1 + \frac{T_i \log(1+\delta)}{3\sigma_i}} \quad (3.5)$$

In equation (3.5), $\delta = 1$ is recommended and σ_i is the standard deviation of the values of the cost function observed during i^{th} loop of the algorithm.

Triki et al. (2005) mentioned that the schedule, given in equation (3.6), which is proposed by Hajek (1988), ensures to find optimum solution. But it is too slow which makes the interest on the schedule theoretical. A similar schedule proposed by Geman and Geman (1984) is given in equation (3.7).

$$T_i = \frac{c}{\ln(1+i)} \quad (3.6)$$

$$T_i = \frac{c}{\log(d+i)} \quad (3.7)$$

Although, some examples are given here, there are more schedules proposed in the literature. There are also, adaptive cooling schedules. In this study, geometric cooling schedule is used. Because it is an effective and fast schedule which is widely used.

Notice that, there are some parameters in cooling schedules. For example, in geometric cooling schedule, α (cooling rate), ϑ (equilibrium state), T_0 (initial temperature), T_f (final temperature) need to be tuned so that the algorithm can converge efficiently.

“Parameter Tuning” is executed by doing test runs using alternative variations of

parameters. After a set of parameters that gives the best solutions in best run-times are determined, actual run of the algorithm starts. (There is also self-adaptive parameter tuning, which is making the computer adapt the parameters in the course of the run.). The parameter tuning executed for this study is explained in Section 5.

A basic pseudo code for SA algorithm is given in Figure 3.5. In the pseudo code, $\mathbf{X}$ is a set of feasible solutions and $(\mathbf{x} \in \mathbf{X})$ and $\mathbf{x} = [x_1, x_2, \dots, x_n]$. $f(\mathbf{x})$ is the minimization objective function, $N(\mathbf{x})$ is the set of $\mathbf{x}'$.

```

1. Initialization: Determine  $T_0, T_i, \alpha, \partial$ , parameters,  $r = 0$ 
2.       Generate an initial solution  $\mathbf{x}$  ( $\mathbf{x} \in \mathbf{X}$ ) and  $\mathbf{x}_{\text{best}} = \mathbf{x}$ 
3.       Calculate  $f(\mathbf{x})$  and  $f(\mathbf{x}_{\text{best}}) = f(\mathbf{x})$ 
4.       Define the cooling schedule  $T_i = T_{i-1} \times \alpha$ 
5. Repeat
6. Generate a random neighbor  $\mathbf{x}' \in N(\mathbf{x})$  and calculate  $f(\mathbf{x}')$ 
7.   If  $f(\mathbf{x}') - f(\mathbf{x}) \leq 0$  Then
8.      $\mathbf{x} \leftarrow \mathbf{x}'$ 
9.   Else
10.    Choose a random value,  $p \approx U[0,1]$ 
11.    If  $p \leq e^{-\frac{f(\mathbf{x}') - f(\mathbf{x})}{T}}$  Then
12.       $\mathbf{x} \leftarrow \mathbf{x}'$ 
13.       $r = r + 1$ 
14.    End
15.    If  $r = \partial$  Then
16.       $T_i = T_{i-1} \times \alpha$ 
17.       $r = 0$ 
18.    End
19.  End
20. If  $f(\mathbf{x}) < f(\mathbf{x}_{\text{best}})$  Then  $\mathbf{x}_{\text{best}} \leftarrow \mathbf{x}$ ,  $f(\mathbf{x}_{\text{best}}) = f(\mathbf{x})$  End
21. Until ( $T < T_f$ )
22. Output:  $\mathbf{x}_{\text{best}}$  and  $f(\mathbf{x}_{\text{best}})$ 

```

Figure 3.5 Pseudo code of SA heuristic

3.2. Benchmark Problems

In the field of optimization, it is common to make comparisons between methods using widely known, standard optimization problems. Two widely known combinatorial optimization problems are TSP and QAP.

3.2.1. Travelling salesman problem

In *Travelling Salesman Problem (TSP)*, a salesman starting from a city, has to visit n

cities (from a known list of cities) exactly once and then he has to return to the first city. Salesman wants to select an order to visit the cities with minimum total distance travelled. More formally, let $G = (V, A)$ be a graph where V is a set of n vertices. A is a set of arcs (i.e. edges) and let $C = C_{ij}$ be a distance matrix associated with A . In TSP, the objective is to determine a *tour* that result in minimum distance while visiting each vertex once and leaving that vertex only once. Such a tour is also known as a *Hamiltonian cycle*. A common categorization of TSP instances is that C can be a symmetric ($C_{ij} = C_{ji}$ for $\forall i, j$) or an asymmetric ($C_{ij} \neq C_{ji}$) matrix.

The mathematical model given by equations (3.8) to (3.12) is a common formulation for TSP where x_{ij} is the binary variable which takes the value of 1 if the salesman goes from city i to city j directly or takes the value of 0 otherwise and C_{ij} is the distance between cities i and j . In the model, equation (3.8) is the objective function, equation (3.9) and equation (3.10) make sure that the salesman visits and leaves every city exactly once and equation (3.11) is the sub-tour elimination constraint which guarantees having a complete tour where $\hat{S}$ is the subsets of the city set N .

$$\min \sum_{ij} C_{ij} x_{ij} \quad (3.8)$$

$$\sum_j x_{ij} = 1 \quad \text{for } \forall i \quad (3.9)$$

$$\sum_i x_{ij} = 1 \quad \text{for } \forall j \quad (3.10)$$

$$\sum_{i \in \hat{S}} \sum_{j \in \hat{S}} x_{ij} \leq |\hat{S}| - 1 \quad \text{for } \forall \hat{S} \subset N, \hat{S} \neq \emptyset \quad (3.11)$$

$$x_{ij} \in \{0,1\} \quad \text{for } \forall i, j \quad (3.12)$$

TSP instances used in the computational study are taken from TSPLIB which is a library of TSP instances that is prepared by Reinelt (1991). TSPLIB contains symmetric and asymmetric instances and best-known solutions of them.

3.2.2. Quadratic assignment problem

In *Quadratic Assignment Problem (QAP)*, there are a set of facilities and a set of locations. The distances between locations and the flows between the facilities are fixed. The objective is to assign each facility to each location while minimizing the sum of the products of flows and distances. Mathematically, the problem can be formulated by a

flow matrix F sized $n \times n$ whose element f_{ij} represents the flow between facilities i and j , in addition a distance matrix D sized $n \times n$ whose element d_{ij} represents the distance between locations i and j . The goal is the minimization of $\sum_i \sum_j f_{ij} d_{x(i)x(j)}$ over the set of all assignments, where the vector X represents an assignment. A common categorization of QAP instances is that F and D can be a symmetric ($f_{ij} = f_{ji}$ and $d_{ij} = d_{ji}$ for $\forall i, j$) or an asymmetric ($f_{ij} \neq f_{ji}$ and/or $d_{ij} \neq d_{ji}$) matrix. The mathematical model defined by equations (3.13) to (3.16) is a common formulation for QAP where variable x_{ij} takes the value of 1 if facility i is assigned to location j or takes the value of 0 otherwise. In the model, objective function is given by equation (3.13), assigning each facility to a single location and assigning each location to a single facility are stipulated by equation (3.14) and equation (3.15), respectively.

$$\min \sum_i \sum_j f_{ij} d_{x(i)x(j)} x_{ij} \quad (3.13)$$

$$\sum_j x_{ij} = 1 \quad \text{for } \forall i \quad (3.14)$$

$$\sum_i x_{ij} = 1 \quad \text{for } \forall j \quad (3.15)$$

$$x_{ij} \in \{0,1\} \quad \text{for } \forall i, j \quad (3.16)$$

QAP instances used in the computational study are taken from QAPLIB which is a library of QAP instances that is prepared by Burkard et al. (1997). QAPLIB contains an explanation of the problem, problem instances, best-known solutions of the instances and solutions methods used to find the given solutions.

3.3. Fractal Geometry

Fractal geometry is first introduced to the world by Benoît B. Mandelbrot. A famous quote from Mandelbrot (1983) is “*Clouds are not spheres, mountains are not cones, coastlines are not circles, and bark is not smooth, nor does lightning travel in a straight line. Nature is rough.*” Although there is not a definitive definition for fractals, they generally are unmeasurable, have statistical self-similarity and have a fractal dimension (Alligood et al., 1997) unlike proportional shapes which are measurable, have a formula and an integer dimension. Simply, a fractal is a mathematical set that has a repeating pattern at every scale, so that it is described by a recursive algorithm. Therefore, fractal geometry provides both a description and mathematical model for these complex forms.

Fractal geometry is being used in physics, chemistry, biology, geology, meteorology, and materials science, graphic designing, digital imaging, coastline complexity, neuroscience, soil mechanics, chaos etc. (Voss, 1988). Figure 3.6. shows a magnified snowflake which shows fractal characteristics.



Figure 3.6 Snowflake

The recursive algorithm can change for different shapes. Some examples of fractal shapes; cantor set, von Koch Curve, Julia sets, Feigenbaum attractor, Gosper island, Henon map, Sierpinski triangle, Mandelbrot set etc. Some famous fractal sets are shown in Figure 3.7.

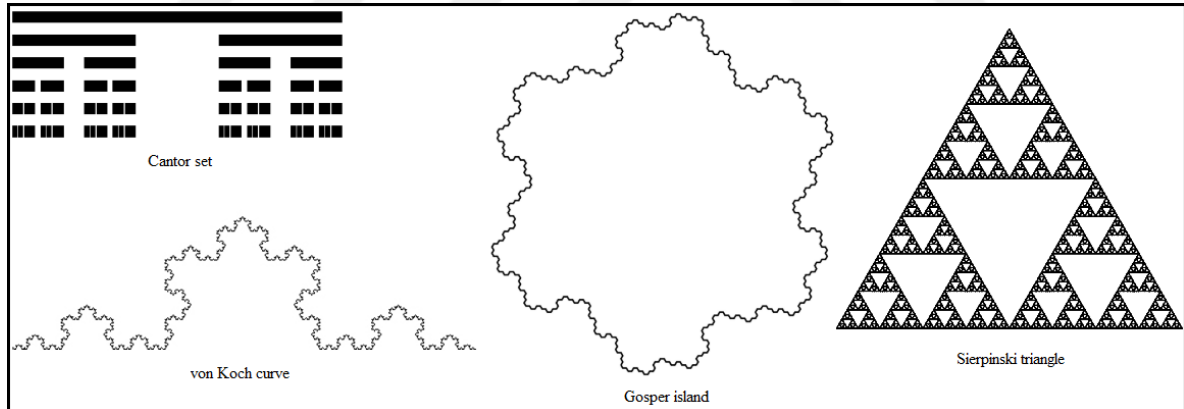


Figure 3.7 Fractal set examples

In summary, fractal geometry aims to calculate, model, and make sense to life-like patterns using systems of recursive / interactive functions and many other techniques. In addition, it aims to model the relation between determinism and prediction to make sense to systems that can be described as deterministic but unpredictable (Aho et al., 1975).

3.3.1. Recursive algorithms of fractal sets

There are mainly two types of recursive algorithms of fractal sets, deterministic or random

algorithms. Deterministic fractals are created by algorithms that have specific values in them. Famous examples are given in Figure 3.7. Random fractals can be generated by using the random values that obeys probability distributions and stochastic rules such as Levy flight, Gaussian walks, random walks, percolation clusters, self-avoiding walks, fractal landscapes, trajectories of Brownian motion etc. Sometimes, the previous steps do not affect the future steps and other times, future steps are dependent on previous steps (Mandelbrot, 1983).

3.3.2. Deterministic generation of cantor sets

The steps in the recursive algorithm of deterministic cantor set (middle-thirds cantor set) are given in Figure 3.8. In Figure 3.8. $L = [0,1]$ in step 1. In step 2, L_l (left), L_m (middle) and L_r (right) and $L_l = [0, \frac{1}{3}] \cup L_m = [\frac{1}{3}, \frac{2}{3}] \cup L_r = [\frac{2}{3}, 1]$.

-
1. Draw a line with a length $L = [0,1]$ and this set is called C_0 .
 2. Repeat
 3. Divide L equally into three parts: L_l , L_m and L_r . Remove L_m . $C_i = C_{i+1}$.
 4. For $L = L_l$ repeat Step 2.
 5. For $L = L_r$ repeat Step 2.
 6. Until $C_n \rightarrow C_\infty$
-

Figure 3.8 The recursive algorithm that generates a deterministic cantor set

As a side note, the ratio does not have to be $\frac{1}{3}$. Figure 3.9. visualizes the middle-thirds cantor set. Middle-thirds cantor set is said to have a 0.63-dimension. Because it does not fill the 2-dimensional space completely, but it is not only a line either.

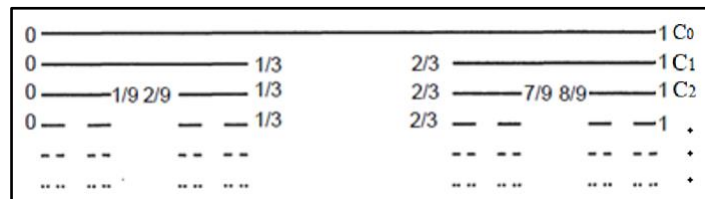


Figure 3.9 Middle-thirds cantor set

Deterministic generation is easier to understand as the intervals can be given specifically. However, when applying stochastic generation, in step (2), the line is not divided into three parts with equal lengths, but it is divided into three parts with random lengths (Alligood et al., 1996, Voss, 1988).

3.3.3. Random generation of cantor sets

A fractal shape can also be sufficiently self-similar. By relating the size of the random variations to the scale, the fractal can be statistically self-similar. This means that enlargements of small parts would have the same statistical distribution as the whole set (Falconer, 2004) which makes the smaller parts not exactly self-similar to the whole (Edgar, 2013).

A random fractal should display randomness at all scales; therefore, a random element at each stage of generation must be introduced (Falconer, 2004). However, there is not a standard rule to construct a random fractal.



4. CANTOR-SET BASED METHOD

CB method stands for *cantor set-based method* and it is inspired by the recursive algorithm that generates cantor sets which are fractal sets. Basic information on recursive algorithms that generates cantor sets is given in Section 3. Figure 4.1 is given as a visual representation of strings and sub-strings to make the explanation of CB easier. Please note that the distribution of strings and substrings looks like a deterministic cantor set and mid-substrings are void to improve visibility. Notice that the concerned part is divided into three parts and the middle sections are removed based on the rules of generating cantor sets. However, if a section is removed from the solution representation, the integrity of the representation (which is the permutation of n elements) will be lost. Therefore, the dividing process is executed in CB, but the middle parts are committed to memory. CB mechanism produces similar neighbors that are different enough, by using the self-similar aspect of fractal shapes. Main motivation of CB mechanism is to provide good chunks inside the neighborhoods to be explored throughout the search process as a side effect.

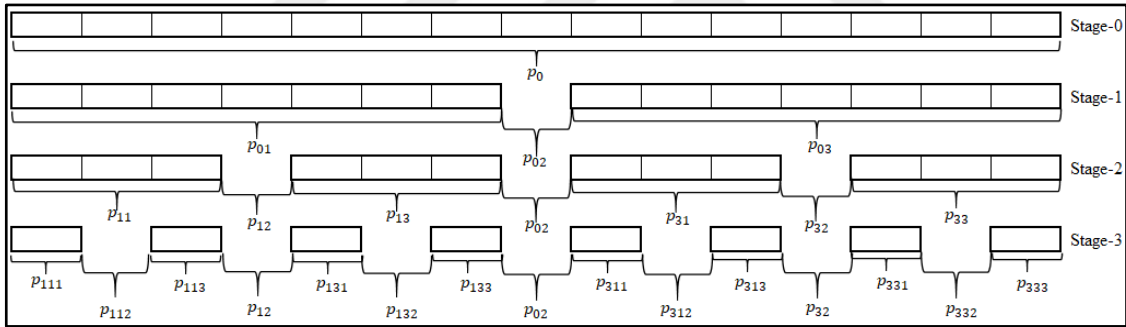


Figure 4.1 A representative figure for how substrings in each stage of CB would look like

In the initial stage (Stage-0), the permutation solution representation, $x = [x_1, x_2, \dots, x_n]$, is represented by a complete string (p_0) of size n , where n is the problem size. In stage-1, p_0 is divided into three substrings. For this purpose, a starting element, s , is systematically increased from 1 to $(n - 2)$. For each s , an index number, m , is increased from 1 to $(n - s - 1)$ such that the first substring (p_{01}) contains the variables $[x_1, x_2, \dots, x_s]$, the second (p_{02}) contains $[x_{s+1}, x_{s+2}, \dots, x_{s+m}]$, and finally, the last substring (p_{03}) contains $[x_{s+m+1}, x_{s+m+2}, \dots, x_n]$. For the sake of clarity, the dividing process in stage-1 is explained as follows:

- (1) Substring $p_{01} = [x_1, x_2 \dots x_s]$ is changed for each $s = 1, 2 \dots n - 2$. This means that p_{01} always starts from x_1 and the number of elements in p_0 is increased from 1 to $n - 2$ sequentially. Therefore, the number of times p_{01} is changed depends on problem size n .
- (2) Substring p_{02} always starts with the element x_{s+1} . The number of elements in p_{02} is changed from 1 element, up to $n - 2$ and for each different p_{02} , p_{03} consists of the remaining elements such that, $p_{03} = [x_{s+2} \dots x_n]$ for $p_{02} = [x_{s+1}]$, $p_{03} = [x_{s+3} \dots x_n]$ for $p_{02} = [x_{s+1}, x_{s+2}]$ and so on. The last possible choice for p_{02} is x_{n-1} which leads to $p_{03} = [x_n]$.

Notice that the previously explained dividing process is also valid for stage-2 and stage-3. For stage-2, steps (1) and (2) are applied to p_{01} and p_{03} (left and right parts in the recursive algorithm which generates a cantor set), separately. The mid-substring (p_{02}) is never divided to maintain the self-similarity structure. From p_{01} , the substrings p_{11} , p_{12} and p_{13} are determined and the substrings p_{31} , p_{32} and p_{33} are determined from p_{03} where p_{ij} means that j^{th} substring of string- i . In stage-3, steps (1) and (2) are applied to p_{11} , p_{13} , p_{31} and p_{33} separately. Hence, the substrings p_{111} , p_{112} and p_{113} are generated from p_{11} , the substrings p_{131} , p_{132} and p_{133} are generated from p_{13} , the substrings p_{311} , p_{312} and p_{313} are generated from p_{31} , and finally the substrings p_{331} , p_{332} and p_{333} are generated from p_{33} .

Also, it is important to mention that the lengths of the substrings are changed systematically as explained if the objective is to generate a deterministic neighborhood. However, the lengths are random if the objective is to generate a random neighbor.

Once the substrings in the respective stages are obtained, the neighbor solutions are generated by scheme-1 or scheme-2. In scheme-1, the concerned subsequent substrings are permuted to generate neighbors. In scheme-2, the position of p_{ij} substring(s) are changed to generate neighbors. In its pure form, CB method produces too many neighbors. As n increases, the neighbor size becomes very large to handle by a heuristic algorithm that use CB which makes the algorithm too slow. Therefore, several choices in the design of CB method are further analyzed to increase its effectiveness and especially the efficiency. These design choices are also explained in 5th Section in detail. A pseudo code for general structure of CB method is presented in Figure 4.2 where $f(\mathbf{x})$ is the minimization objective function, in order to explain the method more thoroughly.

```

1. Input:  $\mathbf{x} = [x_1, x_2, \dots, x_n]$ ,  $R = 0$ ,  $i = 0$ , Scheme = 1 or 2
2. Select stage number for moving process  $R_0 \in \{1, 2, 3\}$ .
3. Repeat
4.  $R = R + 1$ 
5.   For starting point  $s = 1 : (n - 2)$  Do
6.     If Scheme = 1 or 2 Then Size =  $(n - s - 1)$ 
7.       For  $m = 1 : \text{Size}$  Do
8.         If  $R \leq 2$  Then
9.            $p_{i1} = [x_1, \dots, x_s]$ 
10.           $p_{i2} = [x_{s+1}, \dots, x_{s+m}]$ 
11.           $p_{i3} = [x_{s+m+1}, \dots, x_n]$ 
12.          If Scheme = 1 Then Generate neighborhood  $N'(\mathbf{x})$  for  $\mathbf{x}$  which
              consists of permutations of  $p_{i1}$ ,  $p_{i2}$ ,  $p_{i3}$ 
13.          Else Generate neighborhood  $N'(\mathbf{x})$  for  $\mathbf{x}$  which consists of
              permutations  $\forall p_{i2}$ 
14.           $N(\mathbf{x}) = N(\mathbf{x}) \cup N'(\mathbf{x})$ 
15.          End
16.        Else
17.           $p_{ij1} = [x_1, \dots, x_s]$ 
18.           $p_{ij2} = [x_{s+1}, \dots, x_{s+m}]$ 
19.           $p_{ij3} = [x_{s+m+1}, \dots, x_n]$ 
20.          If Scheme = 1 Then Generate neighborhood  $N'(\mathbf{x})$  for  $\mathbf{x}$  which
              consists of permutations  $p_{ij1}$ ,  $p_{ij2}$ ,  $p_{ij3}$ 
21.          Else Generate neighborhood  $N'(\mathbf{x})$  for  $\mathbf{x}$  which consists of
              permutations  $\forall p_{ij2}$ 
22.           $N(\mathbf{x}) = N(\mathbf{x}) \cup N'(\mathbf{x})$ 
23.          End
24.        End
25.      End
26.    End
27.  End
28. If  $R = 2$  Then Take  $p_{01}$  and  $p_{03}$  separately and Repeat the steps 5-27 for  $i = 1$  and
    3 and  $n = n'$  where  $n'$  is the size of the corresponding substring  $p_{01}$  or  $p_{03}$  End
29. If  $R = 3$  Then Take  $p_{11}$ ,  $p_{13}$ ,  $p_{31}$ , and  $p_{33}$  separately and Repeat the steps 16-27
    for  $i, j = (1, 1), (1, 3), (3, 1), (3, 3)$  and  $n = n'$  where  $n'$  is the size of the
    corresponding substring  $p_{ij}$  End
30. Until  $R = R_0$ 
31. Output.  $N(\mathbf{x})$ 

```

Figure 4.2. Pseudo code of CB method

A flow chart is given in Figure 4.3 to explain the method in a visual way.

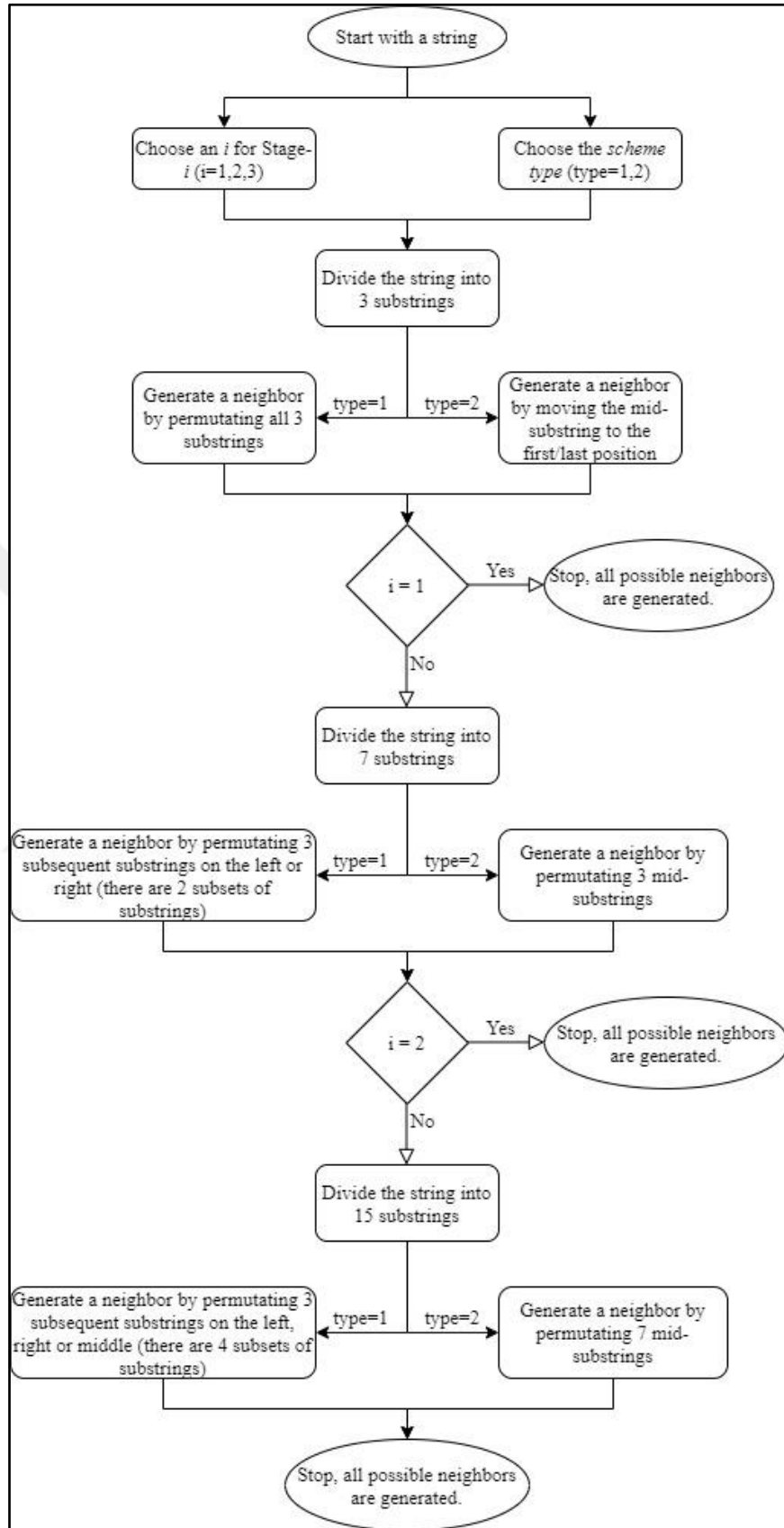


Figure 4.3. A flow chart of CB mechanism steps

Two random solution representations, which are $\mathbf{x} = [7, 6, 5, 4, 3, 2, 1]$ and $\mathbf{x} = [15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]$, are used to give examples for CB. $n = 7$ and 15, where n is the size of string, are the two minimum numbers of n that can visualize stage-2 and stage-3 respectively, which is why they are chosen. The further details can be seen as stage-1 is applied to the $n = 7$ solution representation in Table 4.1, because the minimum number of n to visualize stage-1 is 3. In the examples, the neighbors are generated for each stage separately, although it must be stage-1 and stage-2 for stage-2 and stage-1, stage-2, and stage-3 for stage-3. Otherwise the neighbor lists would be too long.

Table 4.1 demonstrates seventy-five possible neighbors that can be obtained from the string with size $n = 7$ using a CB that uses stage-1 dividing and neighbors are generated by scheme-1. To further explain, for the first five neighbors, the substrings are $[p_{01}] = [7]$, $[p_{02}] = [6]$, $[p_{03}] = [5, 4, 3, 2, 1]$ and they are permuted.

Table 4.1. Whole neighborhood of $\mathbf{x} = [7, 6, 5, 4, 3, 2, 1]$ in stage-1 using scheme-1

$[p_{01}]=[7]; [p_{02}]=[6]; [p_{03}]=[5, 4, 3, 2, 1]$								Order	$[p_{01}]=[7]; [p_{02}]=[6, 5]; [p_{03}]=[4, 3, 2, 1]$							
$x'_{(1)}$	7	5	4	3	2	1	6	$[p_{01}] - [p_{03}] - [p_{02}]$	$x'_{(6)}$	7	4	3	2	1	6	5
$x'_{(2)}$	6	7	5	4	3	2	1	$[p_{02}] - [p_{01}] - [p_{03}]$	$x'_{(7)}$	6	5	7	4	3	2	1
$x'_{(3)}$	6	5	4	3	2	1	7	$[p_{02}] - [p_{03}] - [p_{01}]$	$x'_{(8)}$	6	5	4	3	2	1	7
$x'_{(4)}$	5	4	3	2	1	7	6	$[p_{03}] - [p_{01}] - [p_{02}]$	$x'_{(9)}$	4	3	2	1	7	6	5
$x'_{(5)}$	5	4	3	2	1	6	7	$[p_{03}] - [p_{02}] - [p_{01}]$	$x'_{(10)}$	4	3	2	1	6	5	7
$[p_{01}]=[7]; [p_{02}]=[6, 5, 4]; [p_{03}]=[3, 2, 1]$								Order	$[p_{01}]=[7]; [p_{02}]=[6, 5, 4, 3]; [p_{03}]=[2, 1]$							
$x'_{(11)}$	7	3	2	1	6	5	4	$[p_{01}] - [p_{03}] - [p_{02}]$	$x'_{(16)}$	7	2	1	6	5	4	3
$x'_{(12)}$	6	5	4	7	3	2	1	$[p_{02}] - [p_{01}] - [p_{03}]$	$x'_{(17)}$	6	5	4	3	7	2	1
$x'_{(13)}$	6	5	4	3	2	1	7	$[p_{02}] - [p_{03}] - [p_{01}]$	$x'_{(18)}$	6	5	4	3	2	1	7
$x'_{(14)}$	3	2	1	7	6	5	4	$[p_{03}] - [p_{01}] - [p_{02}]$	$x'_{(19)}$	2	1	7	6	5	4	3
$x'_{(15)}$	3	2	1	6	5	4	7	$[p_{03}] - [p_{02}] - [p_{01}]$	$x'_{(20)}$	2	1	6	5	4	3	7

Table 4.1.(cont'd) Whole neighborhood of $\mathbf{x} = [7, 6, 5, 4, 3, 2, 1]$ in stage-1 using scheme-1

[p ₀₁]=[7]; [p ₀₂]=[6,5,4,3,2]; [p ₀₃]=[1]								Order	[p ₀₁]=[7,6]; [p ₀₂]=[5]; [p ₀₃]=[4,3,2,1]							
x'(₂₁)	7	1	6	5	4	3	2	[p ₀₁] - [p ₀₃] - [p ₀₂]	x'(₂₆)	7	6	4	3	2	1	5
x'(₂₂)	6	5	4	3	2	7	1	[p ₀₂] - [p ₀₁] - [p ₀₃]	x'(₂₇)	5	7	6	4	3	2	1
x'(₂₃)	6	5	4	3	2	1	7	[p ₀₂] - [p ₀₃] - [p ₀₁]	x'(₂₈)	5	4	3	2	1	7	6
x'(₂₄)	1	7	6	5	4	3	2	[p ₀₃] - [p ₀₁] - [p ₀₂]	x'(₂₉)	4	3	2	1	7	6	5
x'(₂₅)	1	6	5	4	3	2	7	[p ₀₃] - [p ₀₂] - [p ₀₁]	x'(₃₀)	4	3	2	1	5	7	6
[p ₀₁]=[7,6]; [p ₀₂]=[5,4]; [p ₀₃]=[3,2,1]								Order	[p ₀₁]=[7,6]; [p ₀₂]=[5,4,3]; [p ₀₃]=[2,1]							
x'(₃₁)	7	6	3	2	1	5	4	[p ₀₁] - [p ₀₃] - [p ₀₂]	x'(₃₆)	7	6	2	1	5	4	3
x'(₃₂)	5	4	7	6	3	2	1	[p ₀₂] - [p ₀₁] - [p ₀₃]	x'(₃₇)	5	4	3	7	6	2	1
x'(₃₃)	5	4	3	2	1	7	6	[p ₀₂] - [p ₀₃] - [p ₀₁]	x'(₃₈)	5	4	3	2	1	7	6
x'(₃₄)	3	2	1	7	6	5	4	[p ₀₃] - [p ₀₁] - [p ₀₂]	x'(₃₉)	2	1	7	6	5	4	3
x'(₃₅)	3	2	1	5	4	7	6	[p ₀₃] - [p ₀₂] - [p ₀₁]	x'(₄₀)	2	1	5	4	3	7	6
[p ₀₁]=[7,6]; [p ₀₂]=[5,4,3,2]; [p ₀₃]=[1]								Order	[p ₀₁]=[7,6,5]; [p ₀₂]=[4]; [p ₀₃]=[3,2,1]							
x'(₄₁)	7	6	1	5	4	3	2	[p ₀₁] - [p ₀₃] - [p ₀₂]	x'(₄₆)	7	6	5	3	2	1	4
x'(₄₂)	5	4	3	2	7	6	1	[p ₀₂] - [p ₀₁] - [p ₀₃]	x'(₄₇)	4	7	6	5	3	2	1
x'(₄₃)	5	4	3	2	1	7	6	[p ₀₂] - [p ₀₃] - [p ₀₁]	x'(₄₈)	4	3	2	1	7	6	5
x'(₄₄)	1	7	6	5	4	3	2	[p ₀₃] - [p ₀₁] - [p ₀₂]	x'(₄₉)	3	2	1	7	6	5	4
x'(₄₅)	1	5	4	3	2	7	6	[p ₀₃] - [p ₀₂] - [p ₀₁]	x'(₅₀)	3	2	1	4	7	6	5
[p ₀₁]=[7,6,5]; [p ₀₂]=[4,3]; [p ₀₃]=[2,1]								Order	[p ₀₁]=[7,6,5]; [p ₀₂]=[4,3,2]; [p ₀₃]=[1]							
x'(₅₁)	7	6	5	2	1	4	3	[p ₀₁] - [p ₀₃] - [p ₀₂]	x'(₅₆)	7	6	5	1	4	3	2
x'(₅₂)	4	3	7	6	5	2	1	[p ₀₂] - [p ₀₁] - [p ₀₃]	x'(₅₇)	4	3	2	7	6	5	1
x'(₅₃)	4	3	2	1	7	6	5	[p ₀₂] - [p ₀₃] - [p ₀₁]	x'(₅₈)	4	3	2	1	7	6	5
x'(₅₄)	2	1	7	6	5	4	3	[p ₀₃] - [p ₀₁] - [p ₀₂]	x'(₅₉)	1	7	6	5	4	3	2
x'(₅₅)	2	1	4	3	7	6	5	[p ₀₃] - [p ₀₂] - [p ₀₁]	x'(₆₀)	1	4	3	2	7	6	5

Table 4.1.(cont'd) Whole neighborhood of $\mathbf{x} = [7, 6, 5, 4, 3, 2, 1]$ in stage-1 using scheme-1

[p ₀₁]=[7,6,5,4]; [p ₀₂]=[3]; [p ₀₃]=[2,1]								Order	[p ₀₁]=[7,6,5,4]; [p ₀₂]=[3,2]; [p ₀₃]=[1]							
x'(₆₁)	7	6	5	4	2	1	3	[p ₀₁] - [p ₀₃] - [p ₀₂]	x'(₆₆)	7	6	5	4	1	3	2
x'(₆₂)	3	7	6	5	4	2	1	[p ₀₂] - [p ₀₁] - [p ₀₃]	x'(₆₇)	3	2	7	6	5	4	1
x'(₆₃)	3	2	1	7	6	5	4	[p ₀₂] - [p ₀₃] - [p ₀₁]	x'(₆₈)	3	2	1	7	6	5	4
x'(₆₄)	2	1	7	6	5	4	3	[p ₀₃] - [p ₀₁] - [p ₀₂]	x'(₆₉)	1	7	6	5	4	3	2
x'(₆₅)	2	1	3	7	6	5	4	[p ₀₃] - [p ₀₂] - [p ₀₁]	x'(₇₀)	1	3	2	7	6	5	4
[p ₀₁]=[7,6,5,4,3]; [p ₀₂]=[2]; [p ₀₃]=[1]								Order								
x'(₇₁)	7	6	5	4	3	1	2	[p ₀₁] - [p ₀₃] - [p ₀₂]								
x'(₇₂)	2	7	6	5	4	3	1	[p ₀₂] - [p ₀₁] - [p ₀₃]								
x'(₇₃)	2	1	7	6	5	4	3	[p ₀₂] - [p ₀₃] - [p ₀₁]								
x'(₇₄)	1	7	6	5	4	3	2	[p ₀₃] - [p ₀₁] - [p ₀₂]								
x'(₇₅)	1	2	7	6	5	4	3	[p ₀₃] - [p ₀₂] - [p ₀₁]								

Table 4.2 demonstrates ten possible neighbors that can be obtained from the string with $n = 7$ using a CB that uses stage-2 dividing and neighbors are generated by scheme-1. Normally stage-2 dividing would contain both stage-1 and stage-2 dividing which means that the neighborhood contains all eighty-five neighbors shown in Tables 4.1 and 4.2. To further explain, for the first five neighbors, the substrings are [p₁₁] = [7], [p₁₂] = [6], [p₁₃] = [5], [p₀₂] = [4], [p₀₃] = [3,2,1] and they are permuted.

Table 4.2. Whole neighborhood of $\mathbf{x} = [7, 6, 5, 4, 3, 2, 1]$ in stage-2 using scheme-1

	$[\mathbf{p}_{11}]=[7]; [\mathbf{p}_{12}]=[6]; [\mathbf{p}_{13}]=[5]; [\mathbf{p}_{02}]=[4]; [\mathbf{p}_{03}]=[3,2,1]$							Order
$x'(1)$	7	5	6	4	3	2	1	$[\mathbf{p}_{11}] - [\mathbf{p}_{13}] - [\mathbf{p}_{12}]$
$x'(2)$	6	7	5	4	3	2	1	$[\mathbf{p}_{12}] - [\mathbf{p}_{11}] - [\mathbf{p}_{13}]$
$x'(3)$	6	5	7	4	3	2	1	$[\mathbf{p}_{12}] - [\mathbf{p}_{13}] - [\mathbf{p}_{11}]$
$x'(4)$	5	7	6	4	3	2	1	$[\mathbf{p}_{13}] - [\mathbf{p}_{11}] - [\mathbf{p}_{12}]$
$x'(5)$	5	6	7	4	3	2	1	$[\mathbf{p}_{13}] - [\mathbf{p}_{12}] - [\mathbf{p}_{11}]$
	$[\mathbf{p}_{01}]=[7, 6, 5]; [\mathbf{p}_{02}]=[4]; [\mathbf{p}_{31}]=[3]; [\mathbf{p}_{32}]=[2]; [\mathbf{p}_{33}]=[1]$							Order
$x'(6)$	7	6	5	4	3	1	2	$[\mathbf{p}_{31}] - [\mathbf{p}_{33}] - [\mathbf{p}_{32}]$
$x'(7)$	7	6	5	4	2	3	1	$[\mathbf{p}_{32}] - [\mathbf{p}_{31}] - [\mathbf{p}_{33}]$
$x'(8)$	7	6	5	4	2	1	3	$[\mathbf{p}_{32}] - [\mathbf{p}_{33}] - [\mathbf{p}_{31}]$
$x'(9)$	7	6	5	4	1	3	2	$[\mathbf{p}_{33}] - [\mathbf{p}_{31}] - [\mathbf{p}_{32}]$
$x'(10)$	7	6	5	4	1	2	3	$[\mathbf{p}_{33}] - [\mathbf{p}_{32}] - [\mathbf{p}_{31}]$

Table 4.3 demonstrates twenty possible neighbors that can be obtained from the string with size $n = 15$ using a CB that uses stage-3 dividing and neighbor are generated by scheme-1. To further explain, for the first five neighbors, the substrings are $[\mathbf{p}_{111}] = [15]$, $[\mathbf{p}_{112}] = [14]$, $[\mathbf{p}_{113}] = [13]$, $[\mathbf{p}_{12}] = [12]$, $[\mathbf{p}_{13}] = [11, 10, 9]$, $[\mathbf{p}_{02}] = [8]$, $[\mathbf{p}_{31}] = [7, 6, 5]$, $[\mathbf{p}_{32}] = [4]$, $[\mathbf{p}_{33}] = [3, 2, 1]$ and they are permuted. All possible neighbors and the order of the substrings are given in the table.

Table 4.3. Neighborhood of $\mathbf{x} = [15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]$ in stage-3 using scheme-1

$[\mathbf{p}_{111}]=[15]; [\mathbf{p}_{112}]=[14]; [\mathbf{p}_{113}]=[13]; [\mathbf{p}_{12}]=[12];$ $[\mathbf{p}_{13}]=[11,10,9]; [\mathbf{p}_{02}]=[8]; [\mathbf{p}_{31}]=[7,6,5]; [\mathbf{p}_{32}]=[4]; [\mathbf{p}_{33}]=[3,2,1]$															Order	
$x'_{(1)}$	15	13	14	12	11	10	9	8	7	6	5	4	3	2	1	$[\mathbf{p}_{111}] - [\mathbf{p}_{113}]$ - $[\mathbf{p}_{112}]$
$x'_{(2)}$	14	15	13	12	11	10	9	8	7	6	5	4	3	2	1	$[\mathbf{p}_{112}] - [\mathbf{p}_{111}]$ - $[\mathbf{p}_{113}]$
$x'_{(3)}$	14	13	15	12	11	10	9	8	7	6	5	4	3	2	1	$[\mathbf{p}_{112}] - [\mathbf{p}_{113}]$ - $[\mathbf{p}_{111}]$
$x'_{(4)}$	13	15	14	12	11	10	9	8	7	6	5	4	3	2	1	$[\mathbf{p}_{113}] - [\mathbf{p}_{111}]$ - $[\mathbf{p}_{112}]$
$x'_{(5)}$	13	14	15	12	11	10	9	8	7	6	5	4	3	2	1	$[\mathbf{p}_{113}] - [\mathbf{p}_{112}]$ - $[\mathbf{p}_{111}]$

Table 4.3.(cont'd) Neighborhood of $\mathbf{x} = [15,14,13,12,11,10,9,8,7,6,5,4,3,2,1]$ in stage-3
using scheme-1

[p ₁₁]=[15,14,13]; [p ₁₂]=[12]; [p ₁₃₁]=[11]; [p ₁₃₂]=[10]; [p ₁₃₃]=[9]; [p ₀₂]=[8]; [p ₃₁]=[7,6,5]; [p ₃₂]=[4]; [p ₃₃]=[3,2,1]															Order	
x' ₍₆₎	15	14	13	12	11	9	10	8	7	6	5	4	3	2	1	[p ₁₃₁] - [p ₁₃₃] - [p ₁₃₂]
x' ₍₇₎	15	14	13	12	10	11	9	8	7	6	5	4	3	2	1	[p ₁₃₂] - [p ₁₃₁] - [p ₁₃₃]
x' ₍₈₎	15	14	13	12	10	9	11	8	7	6	5	4	3	2	1	[p ₁₃₂] - [p ₁₃₃] - [p ₁₃₁]
x' ₍₉₎	15	14	13	12	9	11	10	8	7	6	5	4	3	2	1	[p ₁₃₃] - [p ₁₃₁] - [p ₁₃₂]
x' ₍₁₀₎	15	14	13	12	9	10	11	8	7	6	5	4	3	2	1	[p ₁₃₃] - [p ₁₃₂] - [p ₁₃₁]
[p ₁₁]=[15,14,13]; [p ₁₂]=[12]; [p ₁₃]=[11,10,9]; [p ₀₂]=[8]; [p ₃₁₁]=[7]; [p ₃₁₂]=[6]; [p ₃₁₃]=[5]; [p ₃₂]=[4]; [p ₃₃]=[3,2,1]															Order	
x' ₍₁₁₎	15	14	13	12	11	10	9	8	7	5	6	4	3	2	1	[p ₃₁₁] - [p ₃₁₃] - [p ₃₁₂]
x' ₍₁₂₎	15	14	13	12	11	10	9	8	6	7	5	4	3	2	1	[p ₃₁₂] - [p ₃₁₁] - [p ₃₁₃]
x' ₍₁₃₎	15	14	13	12	11	10	9	8	6	5	7	4	3	2	1	[p ₃₁₂] - [p ₃₁₃] - [p ₃₁₁]
x' ₍₁₄₎	15	14	13	12	11	10	9	8	5	7	6	4	3	2	1	[p ₃₁₃] - [p ₃₁₁] - [p ₃₁₂]
x' ₍₁₅₎	15	14	13	12	11	10	9	8	5	6	7	4	3	2	1	[p ₃₁₃] - [p ₃₁₂] - [p ₃₁₁]
[p ₁₁]=[15,14,13]; [p ₁₂]=[12]; [p ₁₃]=[11,10,9]; [p ₀₂]=[8]; [p ₃₁]=[7,6,5]; [p ₃₂]=[4]; [p ₃₃₁]=[3]; [p ₃₃₂]=[2]; [p ₃₃₃]=[1]															Order	
x' ₍₁₆₎	15	14	13	12	11	10	9	8	7	6	5	4	3	1	2	[p ₃₃₁] - [p ₃₃₃] - [p ₃₃₂]
x' ₍₁₇₎	15	14	13	12	11	10	9	8	7	6	5	4	2	3	1	[p ₃₃₂] - [p ₃₃₁] - [p ₃₃₃]
x' ₍₁₈₎	15	14	13	12	11	10	9	8	7	6	5	4	2	1	3	[p ₃₃₂] - [p ₃₃₃] - [p ₃₃₁]
x' ₍₁₉₎	15	14	13	12	11	10	9	8	7	6	5	4	1	3	2	[p ₃₃₃] - [p ₃₃₁] - [p ₃₃₂]
x' ₍₂₀₎	15	14	13	12	11	10	9	8	7	6	5	4	1	2	3	[p ₃₃₃] - [p ₃₃₂] - [p ₃₃₁]

Table 4.4 demonstrates thirty possible neighbors that can be obtained from $n = 7$ string using a CB that uses stage-1 dividing and neighbor are generated by scheme-2. To further explain, for the first two neighbors, the substrings are $[p_{01}] = [7]$, $[p_{02}] = [6]$, $[p_{03}] = [5,4,3,2,1]$ and p_{02} are moved to generate neighbors.

Table 4.4. Whole neighborhood of $\mathbf{x} = [7, 6, 5, 4, 3, 2, 1]$ in stage-1 using scheme-2

$[p_{01}]=[7]; [p_{02}]=[6];$ $[p_{03}]=[5,4,3,2,1]$		Order	$[p_{01}]=[7]; [p_{02}]=[6,5];$ $[p_{03}]=[4,3,2,1]$	
$x'(1)$	7 5 4 3 2 1 6	$[p_{01}] - [p_{03}] - [p_{02}]$	$x'(3)$	6 5 7 4 3 2 1
$x'(2)$	6 7 5 4 3 2 1	$[p_{02}] - [p_{01}] - [p_{03}]$	$x'(4)$	7 4 3 2 1 6 5
$[p_{01}]=[7]; [p_{02}]=[6,5,4];$ $[p_{03}]=[3,2,1]$		Order	$[p_{01}]=[7]; [p_{02}]=[6,5,4,3];$ $[p_{03}]=[2,1]$	
$x'(5)$	6 5 4 7 3 2 1	$[p_{01}] - [p_{03}] - [p_{02}]$	$x'(7)$	6 5 4 3 7 2 1
$x'(6)$	7 3 2 1 6 5 4	$[p_{02}] - [p_{01}] - [p_{03}]$	$x'(8)$	7 2 1 6 5 4 3
$[p_{01}]=[7]; [p_{02}]=[6,5,4,3,2];$ $[p_{03}]=[1]$		Order	$[p_{01}]=[7,6]; [p_{02}]=[5];$ $[p_{03}]=[4,3,2,1]$	
$x'(9)$	6 5 4 3 2 7 1	$[p_{01}] - [p_{03}] - [p_{02}]$	$x'(11)$	5 7 6 4 3 2 1
$x'(10)$	7 1 6 5 4 3 2	$[p_{02}] - [p_{01}] - [p_{03}]$	$x'(12)$	7 6 4 3 2 1 5
$[p_{01}]=[7,6]; [p_{02}]=[5,4];$ $[p_{03}]=[3,2,1]$		Order	$[p_{01}]=[7,6]; [p_{02}]=[5,4,3];$ $[p_{03}]=[2,1]$	
$x'(13)$	5 4 7 6 3 2 1	$[p_{01}] - [p_{03}] - [p_{02}]$	$x'(15)$	5 4 3 7 6 2 1
$x'(14)$	7 6 3 2 1 5 4	$[p_{02}] - [p_{01}] - [p_{03}]$	$x'(16)$	7 6 2 1 5 4 3
$[p_{01}]=[7,6]; [p_{02}]=[5,4,3,2];$ $[p_{03}]=[1]$		Order	$[p_{01}]=[7,6,5];$ $[p_{02}]=[4]; [p_{03}]=[3,2,1]$	
$x'(17)$	5 4 3 2 7 6 1	$[p_{01}] - [p_{03}] - [p_{02}]$	$x'(19)$	4 7 6 5 3 2 1
$x'(18)$	7 6 1 5 4 3 2	$[p_{02}] - [p_{01}] - [p_{03}]$	$x'(20)$	7 6 5 3 2 1 4
$[p_{01}]=[7,6,5];$ $[p_{02}]=[4,3]; [p_{03}]=[2,1]$		Order	$[p_{01}]=[7,6,5];$ $[p_{02}]=[4,3,2]; [p_{03}]=[1]$	
$x'(21)$	4 3 7 6 5 2 1	$[p_{01}] - [p_{03}] - [p_{02}]$	$x'(23)$	4 3 2 7 6 5 1
$x'(22)$	7 6 5 2 1 4 3	$[p_{02}] - [p_{01}] - [p_{03}]$	$x'(24)$	7 6 5 1 4 3 2
$[p_{01}]=[7,6,5,4];$ $[p_{02}]=[3]; [p_{03}]=[2,1]$		Order	$[p_{01}]=[7,6,5,4];$ $[p_{02}]=[3,2]; [p_{03}]=[1]$	
$x'(25)$	3 7 6 5 4 2 1	$[p_{01}] - [p_{03}] - [p_{02}]$	$x'(27)$	3 2 7 6 5 4 1
$x'(26)$	7 6 5 4 2 1 3	$[p_{02}] - [p_{01}] - [p_{03}]$	$x'(28)$	7 6 5 4 1 3 2
$[p_{01}]=[7,6,5,4,3];$ $[p_{02}]=[2]; [p_{03}]=[1]$		Order		
$x'(29)$	2 7 6 5 4 3 1	$[p_{01}] - [p_{03}] - [p_{02}]$		
$x'(30)$	7 6 5 4 3 1 2	$[p_{02}] - [p_{01}] - [p_{03}]$		

Table 4.5 demonstrates five possible neighbors that can be obtained from $n = 7$ string using a CB that uses stage-2 dividing and neighbor are generated by scheme-2. To further explain, all neighbors are generated from the substrings $[p_{11}] = [7]$, $[p_{12}] = [6]$, $[p_{13}] = [5]$, $[p_{02}] = [4]$, $[p_{31}] = [3]$, $[p_{32}] = [2]$, $[p_{33}] = [1]$ and p_{12} , p_{02} , p_{32} are permuted to generate neighbors.

Table 4.5. Whole neighborhood of $\mathbf{x} = [7, 6, 5, 4, 3, 2, 1]$ in stage-2 using scheme-2

	$[p_{11}]=[7]; [p_{12}]=[6]; [p_{13}]=[5]; [p_{02}]=[4]; [p_{31}]=[3]; [p_{32}]=[2]; [p_{33}]=[1]$							Order
$x'(1)$	7	6	5	2	3	4	1	$[p_{12}] - [p_{32}] - [p_{02}]$
$x'(2)$	7	4	5	6	3	2	1	$[p_{02}] - [p_{12}] - [p_{32}]$
$x'(3)$	7	4	5	2	3	6	1	$[p_{02}] - [p_{32}] - [p_{12}]$
$x'(4)$	7	2	5	6	3	4	1	$[p_{32}] - [p_{12}] - [p_{02}]$
$x'(5)$	7	2	5	4	3	6	1	$[p_{32}] - [p_{02}] - [p_{12}]$

Table 4.6 demonstrates twenty-four possible neighbors that can be obtained from $n = 15$ string using a CB that uses stage-3 dividing and neighbor are generated by scheme-2. To further explain, all neighbors are generated from the substrings $[p_{111}] = [15]$, $[p_{112}] = [14]$, $[p_{113}] = [13]$, $[p_{12}] = [12]$, $[p_{131}] = [11]$, $[p_{132}] = [10]$, $[p_{133}] = [9]$, $[p_{02}] = [8]$, $[p_{311}] = [7]$, $[p_{312}] = [6]$, $[p_{313}] = [5]$, $[p_{32}] = [4]$, $[p_{331}] = [3]$, $[p_{332}] = [2]$, $[p_{333}] = [1]$ and p_{112} , p_{132} , p_{312} , p_{332} are permuted to generate neighbors. As a side note, p_{12} , p_{02} , p_{32} must be permuted as well but that would mean 5040 possible neighbors which is too much to show here.

Table 4.6. Whole neighborhood of $\mathbf{x} = [15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]$ in stage-3 using scheme-2

[p ₁₁₁]=[15]; [p ₁₁₂]=[14]; [p ₁₁₃]=[13]; [p ₁₂]=[12]; [p ₁₃₁]=[11]; [p ₁₃₂]=[10]; [p ₁₃₃]=[9]; [p ₀₂]=[8]; [p ₃₁₁]=[7]; [p ₃₁₂]=[6]; [p ₃₁₃]=[5]; [p ₃₂]=[4]; [p ₃₃₁]=[3]; [p ₃₃₂]=[2]; [p ₃₃₃]=[1]																
x'(1)	15	2	13	12	11	6	9	8	7	10	5	4	3	14	1	[p ₃₃₂]-[p ₃₁₂]- [p ₁₃₂]- [p ₁₁₂]
x'(2)	15	2	13	12	11	6	9	8	7	14	5	4	3	10	1	[p ₃₃₂]-[p ₃₁₂]- [p ₁₁₂]- [p ₁₃₂]
x'(3)	15	2	13	12	11	10	9	8	7	6	5	4	3	14	1	[p ₃₃₂]-[p ₁₃₂]- [p ₃₁₂]- [p ₁₁₂]
x'(4)	15	2	13	12	11	10	9	8	7	14	5	4	3	6	1	[p ₃₃₂]-[p ₁₃₂]- [p ₁₁₂]- [p ₃₃₂]
x'(5)	15	2	13	12	11	14	9	8	7	6	5	4	3	10	1	[p ₃₃₂]-[p ₁₁₂]- [p ₃₁₂]- [p ₁₃₂]
x'(6)	15	2	13	12	11	14	9	8	7	10	5	4	3	6	1	[p ₃₃₂]-[p ₁₁₂]- [p ₁₃₂]- [p ₃₃₂]
x'(7)	15	6	13	12	11	2	9	8	7	10	5	4	3	14	1	[p ₃₁₂]-[p ₃₃₂]- [p ₁₃₂]- [p ₁₁₂]
x'(8)	15	6	13	12	11	2	9	8	7	14	5	4	3	10	1	[p ₃₁₂]-[p ₃₃₂]- [p ₁₁₂]- [p ₁₃₂]
x'(9)	15	6	13	12	11	10	9	8	7	2	5	4	3	14	1	[p ₃₁₂]-[p ₁₃₂]- [p ₃₃₂]- [p ₁₁₂]
x'(10)	15	6	13	12	11	10	9	8	7	14	5	4	3	2	1	[p ₃₁₂]-[p ₁₃₂]- [p ₁₁₂]- [p ₃₃₂]
x'(11)	15	6	13	12	11	14	9	8	7	2	5	4	3	10	1	[p ₃₁₂]-[p ₁₁₂]- [p ₃₃₂]- [p ₁₃₂]
x'(12)	15	6	13	12	11	14	9	8	7	10	5	4	3	2	1	[p ₃₁₂]-[p ₁₁₂]- [p ₁₃₂]- [p ₃₃₂]
x'(13)	15	10	13	12	11	2	9	8	7	6	5	4	3	14	1	[p ₁₃₂]-[p ₃₃₂]- [p ₃₁₂]- [p ₁₁₂]
x'(14)	15	10	13	12	11	2	9	8	7	14	5	4	3	6	1	[p ₁₃₂]-[p ₃₃₂]- [p ₁₁₂]- [p ₃₁₂]
x'(15)	15	10	13	12	11	6	9	8	7	2	5	4	3	14	1	[p ₁₃₂]-[p ₃₁₂]- [p ₃₃₂]- [p ₁₁₂]
x'(16)	15	10	13	12	11	6	9	8	7	14	5	4	3	2	1	[p ₁₃₂]-[p ₃₁₂]- [p ₁₁₂]- [p ₃₃₂]
x'(17)	15	10	13	12	11	14	9	8	7	2	5	4	3	6	1	[p ₁₃₂]-[p ₁₁₂]- [p ₃₃₂]- [p ₃₁₂]
x'(18)	15	10	13	12	11	14	9	8	7	6	5	4	3	2	1	[p ₁₃₂]-[p ₁₁₂]- [p ₃₁₂]- [p ₃₃₂]
x'(19)	15	14	13	12	11	2	9	8	7	6	5	4	3	10	1	[p ₁₁₂]-[p ₃₃₂]- [p ₃₁₂]- [p ₁₃₂]
x'(20)	15	14	13	12	11	2	9	8	7	10	5	4	3	6	1	[p ₁₁₂]-[p ₃₃₂]- [p ₁₃₂]- [p ₃₁₂]
x'(21)	15	14	13	12	11	6	9	8	7	2	5	4	3	10	1	[p ₁₃₂]-[p ₃₁₂]- [p ₃₃₂]- [p ₁₃₂]
x'(22)	15	14	13	12	11	6	9	8	7	10	5	4	3	2	1	[p ₁₃₂]-[p ₃₁₂]- [p ₁₃₂]- [p ₃₃₂]
x'(23)	15	14	13	12	11	10	9	8	7	2	5	4	3	6	1	[p ₁₁₂]-[p ₁₃₂]- [p ₃₃₂]- [p ₃₁₂]
x'(24)	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	[p ₁₁₂]-[p ₁₃₂]- [p ₃₁₂]- [p ₃₃₂]

Straightforward examples are given in this section. It is observed that the stage number is directly proportional with variation in the neighborhood for both schemes. It can also be argued that scheme-1 is more efficiency oriented than scheme-2 while scheme-2 is more effectiveness oriented than scheme-1. However, there are design choices that can affect the efficiency and effectiveness of CB. An experiment is conducted to test these design choices and the results are given in Section 5.



5. EXPERIMENTAL STUDY

In this section, the experiments that are executed to design CB mechanism is presented. The objectives were to test possible design choices and learn how they would affect the solution quality which resulted in choosing the best designs. The comparative performance in terms of solution quality of CB is analyzed by using both the experimental results of CB, swap, and insertion mechanisms.

The inspiration for CB comes from the recursive algorithms that creates stochastic cantor sets with the objective to create self-similar neighbors that are different enough as mentioned before. However, the recursive algorithm is a loose inspiration for creating neighbors and if a designer wanted to generate the whole neighborhood using the general idea behind CB, the number of neighbors would increase exponentially. That would result in the usage of extensive memory space to execute the algorithm which means an enormous increase in run-time. Therefore, certain design choices were experimented on using TSP and QAP benchmark instances to decide exactly how to create neighbors. Consequently, while designing the experiments, following questions (Q) are considered:

Q1: How to adapt the recursive algorithm that creates a cantor set as a neighbor generation mechanism?

Q2: How to decrease the memory space used?

Q3: How to decrease the run-time?

To find answers to Q1, Q2 and Q3, the following points are determined to be the issues to consider: the number of stages that CB proceeds, dividing procedure of the strings in each stage, reverse ordering of substrings and neighbor generation schemes. The next section contains a detailed explanation for these issues.

5.1. Design Issues in CB Method

The following design issues for CB method are developed to observe the performance in terms of solution quality of the algorithm:

1. *The number of stages that CB method proceeds (number of stages are $i = 1, 2, 3$). This issue is about the dividing procedure as the stage number determines how many substrings to use in neighbor generation.*

2. *Dividing procedure of the strings in each stage (specific or sequential dividing)*. This issue is about how to change the length of the substrings systematically.
3. *Reverse ordering of substrings (in each stage or selected stages)*. This issue is about introducing more variability to the neighborhood.
4. *Neighbor generation schemes (scheme-1 and scheme-2)*. This issue is about how to generate neighbors in an effective way while considering the run-time.

The reasons to consider these design issues which are run-time of the algorithm, memory used by the algorithm, variation introduced by the issue to the neighborhood are discussed in their respective sections.

5.1.1. The number of stages that CB method proceeds

This design issue adds variation to the neighborhood as i increases which results in needing more memory space and an increase of run-time. This issue affects how many sub-strings are to be created after the dividing process.

In stage-1 three sub-strings are created.

In stage-2 three sub-strings in stage-1 and seven sub-strings in stage-2 are created.

In stage-3 three sub-strings in stage-1, seven sub-strings in stage-2 and fifteen sub-strings for stage-3 are created.

The neighbors are generated using the substrings from the same stage- i in each stage separately.

5.1.2. Dividing procedure of the strings in each stage

Dividing procedure of the strings in each stage can make the algorithm too slow if the designer is not careful. Specific dividing is easier to design and faster to execute than sequential dividing. Sequential dividing adds variation and creates more neighbors which result in a slower algorithm as i and n increase.

Dividing issue is related to run-time. In the original design of CB method, all possible sizes of each substring must be considered. However, this creates an exponential number of neighbors that are very hard to handle. This issue is handled in sequential dividing by creating each substring for every possible size only in the considered stage. This means

that their substrings are not created within the same loop. They are created by starting over in the next stage. In specific dividing, dividing approach is forceful in that the solution representation is divided into halves and stage-2 proceeds from there. For stage-3 the solution representation is divided into quarters. For this reason, the versions that use sequential dividing is slower than the versions that use specific dividing. Now, the number of neighbors generated in each stage is considered.

Stage-1: It must be noted that sequential dividing is used every time in stage-1. The number of neighbors generated in stage-1 is given in equation (5.1).

$$a' = \frac{5(n-2)(n-1)}{2} \quad (5.1)$$

Stage-2: There are two dividing procedures considered, therefore two possible number of neighbors are generated based on this design choice.

1. *Specific Dividing:* The solution representation is divided into two parts. If n is an even number, the size of the parts will be equal, if not, size of p_{01} will be $\left\lceil \frac{n}{2} \right\rceil$ and size of p_{03} will be $n - \left\lceil \frac{n}{2} \right\rceil$. And notice that there will be no p_{02} . Therefore, this option will generate six substrings in stage-2. As a result, the number of neighbors generated in stage-2 is given in equation (5.2).

$$b = \frac{5(n-4)(n-2)}{2} + a' \quad (5.2)$$

2. *Sequential Dividing:* The solution representation is divided into three substrings with sizes from 2 to $(n - 1)$ one by one and seven substrings are produced. Notice that p_{01} , p_{02} and p_{03} will change sizes in every iteration of stage-2. Therefore, sequential dividing makes the algorithm slower than the specific dividing, generating the number of neighbors given in equation (5.3) in stage-2.

$$b' = \begin{cases} a', & n < 7 \\ \sum_{i=6}^{13} [5(n-i)]^+ + a', & 7 \leq n \leq 13 \\ 5n(n-6) + a', & n > 13 \end{cases} \quad \text{where } [(\cdot)]^+ = \begin{cases} (\cdot), & \text{if it is positive} \\ 0, & \text{otherwise} \end{cases} \quad (5.3)$$

It must be noted that even though $b' > b$, b' is still far less from the whole possible number of neighbors than can be created if the basic idea behind the CB mechanism was used, which is given in equation (5.4).

$$\text{Whole possible number of neighbors} = \left[\frac{(n-6)(n-5)(n-4)}{6} \right]^2 \cdot (7! - 1) \quad (5.4)$$

Stage-3: There are two dividing procedures considered, therefore two possible number of neighbors are generated based on this design choice.

1. *Specific Dividing*: The solution representation is divided into four substrings. The first three quarters are at of size $\left\lceil \frac{n}{4} \right\rceil$, while the remaining quarter is the size of $n - 3 \left\lceil \frac{n}{4} \right\rceil$. And notice that there will be no p_{02} , p_{12} and p_{32} . Therefore, this option will generate twelve substrings in stage-3. The number of neighbors generated in Stage-3 as a result is given in equation (5.5).

$$c = \frac{5(n-8)(n-4)}{2} + b \quad (5.5)$$

2. *Sequential Dividing*: The solution representation is divided into fifteen substrings with sizes from 2 to $(n - 14)$ one by one. Notice that p_{11} , p_{13} , p_{31} and p_{33} will change sizes in every iteration of stage-3 generating the number of neighbors given in equation (5.6).

$$c' = \begin{cases} b', & n < 15 \\ \sum_{i=14}^{28} [5(n-i)]^+ + b', & 15 \leq n \leq 28 \\ 5n(n-14) + b', & n > 28 \end{cases} \text{ where } [(\cdot)]^+ = \begin{cases} (\cdot), & \text{if it is positive} \\ 0, & \text{otherwise} \end{cases} \quad (5.6)$$

It must be noted that even though $c' > c$, c' is still far less from the whole possible number of neighbors than can be created if the basic idea behind the CB method was used, which is which is given in equation (5.7).

$$\text{Whole possible number of neighbors} = \left[\frac{(n-14)(n-13)(n-12)}{6} \right]^4 \cdot (15! - 1) \quad (5.7)$$

5.1.3. Reverse ordering of mid-substrings

This design choice exists to add more variation in a search thus giving the mechanism more effectiveness. Reverse ordering is only applied to p_{i2} and p_{ij2} substrings.

This method can be used in all stages if desired, or it can be used partially (for example only in stage-1 in a 3-stages process). It is obvious that it adds more variation if it is used in every stage. However, this point is made in consideration of run-time.

5.1.4. Neighbor generation schemes: scheme-1 and scheme-2

Different problems respond to different algorithm structures. These schemes are designed to be able to use the same principle neighbor generation mechanism with some changes for various problems and to observe which problem responds to which scheme.

Neighbor Generation Scheme-1: The neighbor set of stage-2 is generated by obtaining permutations of the substrings p_{kj} for each $k = 1, 3$ (left and right substrings) separately

while keeping the other substrings in the same position. Similarly, each substrings p_{ij} , but the mid-substrings (p_{i2}), are divided into three parts, $t = 1, 2$, and 3 , to create substrings, p_{ijt} , in stage-3. The neighbor set of stage-3 is created by the permutations of substrings p_{ijt} for each pair of i, j separately while fixing the other substrings in the same position creating $(3! - 1)$ neighbors for each pair.

Figure 5.1 shows a visual representation of scheme-1. It should be noted that the mid-substrings (p_{i2} , $i = 0, 1, 3$) are not divided into subparts and the permutations in each stage are obtained using only the three adjacent substrings for the sake of self-similarity structure of cantor sets.

Neighbor Generation Scheme-2: In the second scheme, dividing of the substrings is the same with the previous scheme. However, the permutation procedure differs in that it is only done between substrings p_{i2} and p_{ij2} without changing the position and order of other substrings as explained in Figure 5.2, visually. The tests are executed by using p_{i2} and p_{ij2} that have more than 1 element or by using p_{i2} and p_{ij2} that only have 1 element. This will be shown as $\text{size}(p_{i2}) \geq 1$ and $\text{size}(p_{i2}) = 1$ for ease of presentation from now on. The reason to test them separately, as a mechanism with $\text{size}(p_{i2}) = 1$ is a subset of a mechanism with $\text{size}(p_{i2}) \geq 1$, is to consider the problems with varying landscapes. Therefore, the experiments are done using neighbor generation mechanisms that have less variation and more variation in their neighborhoods. The comparisons are made by applying the mechanisms on different problems which has a steep landscape (TSP) and a flat landscape (QAP) to be able to analyze them thoroughly.

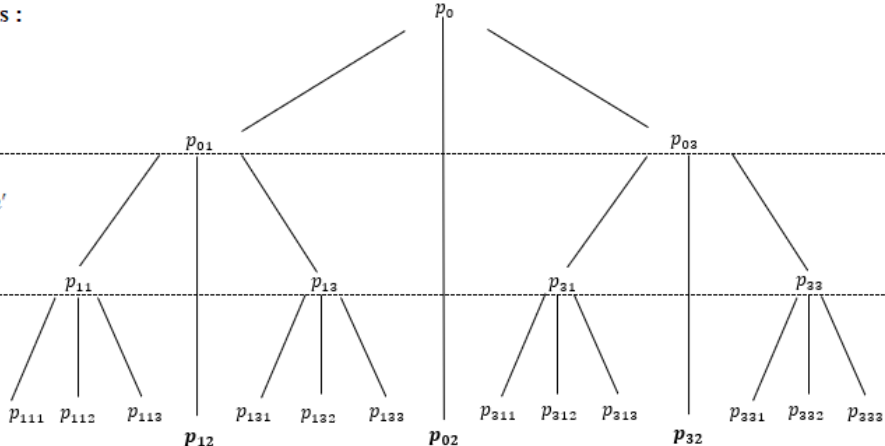
Neighboring Rules :	Number of Neighbors :	
Permutate p_{01}, p_{02}, p_{03}	$a' = \frac{5(n-2)(n-1)}{2}$	
Permutate p_{11}, p_{12}, p_{13} and keep others	Specific Dividing: $b = \frac{5(n-4)(n-2)}{2} + a'$	
Permutate p_{21}, p_{22}, p_{23} and keep others	Sequential Dividing: $b' = 5n(n-6) + a'$	
Permutate $p_{111}, p_{112}, p_{113}$ and keep others	Specific Dividing: $c = \frac{5(n-8)(n-4)}{2} + b$	
Permutate $p_{131}, p_{132}, p_{133}$ and keep others	Sequential Dividing: $c' = 5n(n-14) + b'$	
Permutate $p_{211}, p_{212}, p_{213}$ and keep others		
Permutate $p_{231}, p_{232}, p_{233}$ and keep others		

Figure 5.1. A visual representation of neighbor generation using scheme-1

Neighboring Rules :	Number of Neighbors :	
Move p_{02} to front and end of the solution representation	$2!(n-2)$	
Permutate p_{12}, p_{02}, p_{32} and keep others	$3!(n-4)$	
Permutate $p_{112}, p_{12}, p_{132}, p_{02}, p_{312}, p_{32}, p_{332}$ and keep others	$7!(n-8)$	

Figure 5.2. A visual representation of neighbor generation using scheme-2

The experiments are executed for each stage- i ($i = 1, 2, 3$), separately. The reason is that there are 34 experiments and presenting them in parts would give the explanations a structure. The experiments are combinations of probable design choices which are given temporary names in order of experimentation. The related experiments of CB mechanism are embedded into LS separately and quality of solutions obtained from the experiments are compared against one another, swap, and insertion.

The purpose of the experiments is to find reasonable answers to the questions presented in the following list for TSP and QAP, separately.

1. How many stages should the algorithm proceed?
2. Which dividing procedure should the mechanism use while considering the balance of the solution quality and the run-time?
3. Is straight ordering or reverse ordering more advantageous? Is there a meaningful difference between them?
4. Is scheme-1 or scheme-2 more advantageous? Is there a meaningful difference between them?
5. For scheme-1 and scheme-2, is $\text{size}(p_{i2}) \geq 1$ or $\text{size}(p_{i2}) = 1$ more advantageous?

5.2. Statistical Analysis of Design Issues in CB Mechanism

In the computational tests, MATLAB R2017b was used for algorithm application where the hardware was an Intel i5 -7200U CPU@2,50 GHz 2,70 GHz. To gain statistical insight from the comparison of the mechanisms, SPSS-24 was used as a statistical analyzer. The benchmark instances were taken from TSPLIB (Reinelt, 1991) and QAPLIB (Burkard et al., 2010).

In this section, all experiments are compared to widely known swap and insertion mechanisms in terms of efficiency and effectiveness.

In the experiments, CB variants, swap and insertion mechanisms were embedded into LS algorithm, separately. Every algorithm was run 30 times starting from the same initial solution for each neighboring mechanism solving the same test instances. It is important to note that, at every step of the experiments, the next set of experiments will be determined dynamically according to previous experiments' results. The promising design combinations will be included in the experiments and the combinations that are

proven to perform unsatisfactorily will be abandoned. As a side note, for all the instances chosen for TSP experiments, the optimal solution is known. But for QAP instances, only the optimal solutions of Tai15a, Tai25a are known. For the other instances, lower bounds are used in the calculations.

5.2.1. Stage-1 experiment report

The experimental design is shown in Figure 5.3. In stage-1 neighbor generation, testing any combination of specific dividing would be redundant because the mechanisms would generate the same neighbors repeatedly. Variation would be very limited in such a design. Also combining $size(p_{i2}) = 1$ with reverse ordering is unnecessary as it will generate the same neighbors as the combination of $size(p_{i2}) = 1$ and straight ordering (that has the same design choices for the other issues). Because, if the size of the p_{i2} which is the only substring that reverse ordering is applied to is one, p_{i2} will not change. This combination will only add a useless step to the algorithm which would result in an increase of run-time.

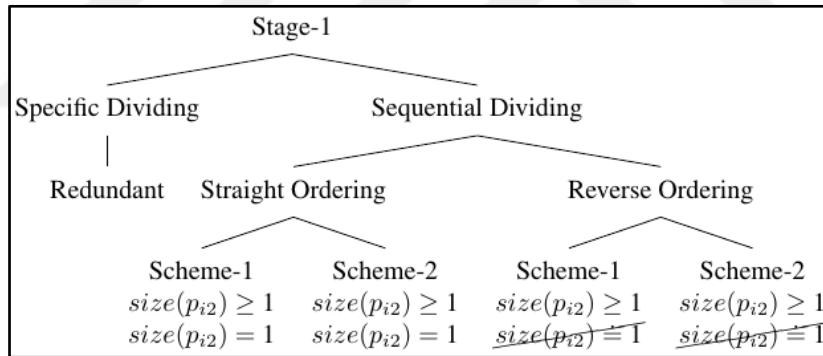


Figure 5.3. Structure of stage-1 experiments

The experiment (E) list which is prepared to consist promising combinations of the design choices to answer the design questions and to find the best possible combination is given below:

- E1: Stage-1, Sequential dividing, Straight ordering, Scheme-1, $size(p_{i2}) \geq 1$
- E2: Stage-1, Sequential dividing, Straight ordering, Scheme-1, $size(p_{i2}) = 1$
- E3: Stage-1, Sequential dividing, Straight ordering, Scheme-2, $size(p_{i2}) \geq 1$
- E4: Stage-1, Sequential dividing, Straight ordering, Scheme-2, $size(p_{i2}) = 1$
- E5: Stage-1, Sequential dividing, Reverse ordering, Scheme-1, $size(p_{i2}) \geq 1$

- E6: Stage-1, Sequential dividing, Reverse ordering, Scheme-2, $\text{size}(p_{i2}) \geq 1$

The performance in terms of solution quality of each experiment is given in Table 5.1, the number of neighbors generated until the algorithm is terminated is given in Table 5.2 and the run-time is given in Table 5.3. As a side note, deviation percentage from the best-known solution (5.8) is considered as performance in terms of solution quality criterion.

$$\text{Solution quality} = \frac{(\text{found objective value}) - (\text{best known objective value})}{(\text{best known objective value})} \cdot 100 \quad (5.8)$$

Table 5.1. Performance of stage-1 experiments in terms of solution quality

Instance	Swap	Insertion	E1	E2	E3	E4	E5	E6
Traveling Salesman Problem								
burma14	7.9	0.4	17.1	35.7	14	32.2	0.5	4.8
att48	44.6	17.8	54.4	124	33.9	134	2.6	5.7
st70	63.1	29.3	56.7	143.1	50.2	158.4	4.2	10.8
kroa100	102.9	42.6	91.7	234.2	81.2	245.4	4.4	17.7
ch150	102.5	38.1	96	265.8	79.4	258.9	7.2	34.3
Quadratic Assignment Problem								
Tai15a	4.4	19.6	12.7	15.6	15.2	17.1	11.3	13.2
Tai25a	5.4	19.2	13.7	16	15.6	17.6	13.3	14.8
Tai50a	4.8	16.8	13.5	15.4	14.5	15.8	13.1	14
Tai80a	3.7	13.6	11.3	12.7	12	13.2	11.1	11.9
Tai100a	3.4	12.5	10.7	12	11.3	12.4	10.6	11.2

In consideration of performance in terms of solution quality, E5 is the best among the experiments for the TSP. In addition, it outperforms both swap and insertion and it is chosen as a candidate for the mechanisms that effectively solve TSP. For QAP, swap gives the best performance. Among the CB experiments, E5 gives the best performance. However, it cannot compare to the swap. Therefore, in this set of experiments no candidate that can have a good performance for QAP is found.

E5 gives the best performance and it is the mechanism that creates the most neighbors for both problems among CB variants. While the variation it generates is a welcome property for TSP, it is not suitable for QAP. For the stage-1 experiments, this suggests that number of neighbors seem to affect the performance. However, before making a final decision on the subject, results of the other experiments were analyzed as well.

Table 5.2. Number of neighbors generated by stage-1 experiments

Instance	Swap	Insertion	E1	E2	E3	E4	E5	E6
Traveling Salesman Problem								
burma14	8.10^2	2.10^3	3.10^3	4.10^2	1.10^3	2.10^2	8.10^3	3.10^3
att48	5.10^4	1.10^5	2.10^5	7.10^3	8.10^4	3.10^3	5.10^5	2.10^5
st70	2.10^5	1.10^5	6.10^5	2.10^4	2.10^5	6.10^3	1.10^6	6.10^5
kroa100	6.10^5	1.10^6	2.10^6	3.10^4	7.10^5	1.10^4	5.10^6	2.10^6
ch150	2.10^6	5.10^6	5.10^6	7.10^4	2.10^6	3.10^4	2.10^7	6.10^6
Quadratic Assignment Problem								
Tai15a	9.10^2	5.10^2	2.10^3	2.10^2	5.10^2	7.10^1	3.10^3	1.10^3
Tai25a	5.10^3	2.10^3	6.10^3	5.10^2	2.10^3	1.10^2	1.10^4	4.10^3
Tai50a	4.10^4	9.10^3	3.10^4	1.10^3	1.10^4	3.10^2	7.10^4	2.10^4
Tai80a	2.10^5	2.10^4	1.10^5	2.10^3	3.10^4	5.10^2	2.10^5	6.10^4
Tai100a	3.10^5	4.10^4	2.10^5	2.10^3	4.10^4	7.10^2	3.10^5	1.10^5

E5 is the slowest mechanism for both problems while swap is the fastest for TSP and E4 is the fastest mechanism for QAP. However, this fact is not valuable as swap and E4 cannot also give favorable performances for TSP and QAP, respectively.

Table 5.3. Run time of stage-1 experiments (in seconds)

Instance	Swap	Insertion	E1	E2	E3	E4	E5	E6
Traveling Salesman Problem								
burma14	0.005	0.020	0.024	0.010	0.012	0.005	0.062	0.035
att48	0.130	0.666	0.984	0.089	0.547	0.048	2.583	1.114
st70	0.438	2.402	3.401	0.226	1.668	0.096	9.254	4.175
kroa100	1.660	8.970	10.934	0.480	6.298	0.214	31.950	14.017
ch150	7.949	39.835	42.619	1.039	24.863	0.559	125.219	49.241
Quadratic Assignment Problem								
Tai15a	0.008	0.011	0.029	0.007	0.010	0.003	0.041	0.020
Tai25a	0.033	0.024	0.094	0.016	0.035	0.009	0.175	0.067
Tai50a	0.892	0.227	1.260	0.075	0.369	0.030	2.488	0.872
Tai80a	8.725	1.463	8.425	0.317	2.223	0.090	17.126	4.980
Tai100a	27.486	3.877	19.993	0.571	5.381	0.165	41.721	13.985

Pairwise bar chart comparisons of run-time and solution quality performance for TSP and QAP are given in Figure 5.4 and in Figure 5.5. The run-time is affected by the steps in a specific mechanism and how effective that algorithm is. Because the termination of the

algorithm is sped up in an ineffective algorithm as they are trapped in the local optimum early in the run of the algorithm.

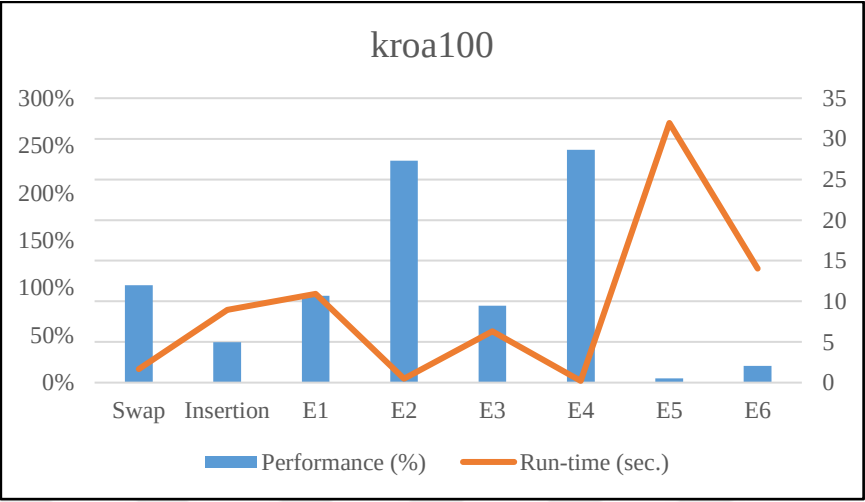


Figure 5.4. Pairwise bar chart comparisons of run-time solving TSP by stage-1 experiments

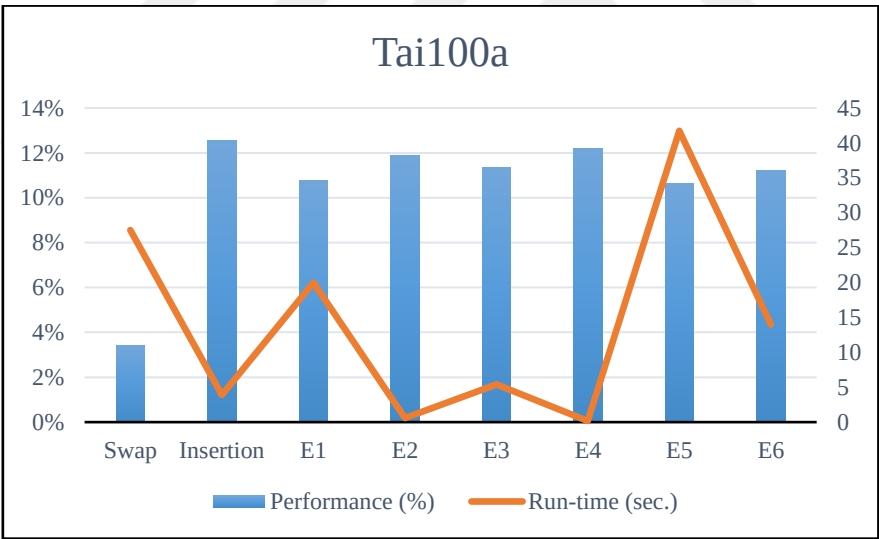


Figure 5.5. Pairwise bar chart comparisons of run-time solving QAP by stage-1 experiments

TSP performance results allow us to argue that the combination of reverse ordering, scheme-1, $\text{size}(p_{i2}) \geq 1$ is very suitable for the problem. However, the experiments for other design choices will be included as well to the stage-2 experiments to test if a faster mechanism can be found or not.

For QAP, the obtained results are not sufficient to choose a combination because all the results from CB experiments are worse than the swap mechanism. Therefore, only stage-1 neighbor generation is eliminated as a candidate for solving the QAP and all combinations will be tested for stage-2, to get a better sense of their performance.

To be able to analyze how the design issues affect the performance and run-time, experiments that differ in only one design issue will be compared to avoid making false assumptions because of an unthought-of influence. The comparisons will be done by performing Wilcoxon signed ranks tests between concerned experiments using QAP performance results. *Wilcoxon signed ranks test* is a non-parametric statistical hypothesis test that can analyze whether there is a statistically meaningful difference between the compared experiments or not. Because the experiments applied to QAP perform similarly, it is not clearly obvious if there is a statistically meaningful difference between them or not.

The performance data of each experiment is obtained by executing 30 independent runs for every n -sized problem to make reliable comparisons. Therefore, the data cannot be said to have a fixed distribution as for TSP, $n = \{14, 48, 70, 100, 150\}$ and for QAP, $n = \{15, 25, 50, 80, 100\}$. In addition, the performance of a heuristic algorithm is dependent on a problem instance, but it is usually independent on the size n . Therefore, a non-parametric (distribution free) test is executed to compare the experiments. Because, in these tests, no assumption of the distribution of the population is needed other than the data has symmetric continuous distribution.

In Wilcoxon signed ranks test $H_0: \mu = \mu_0$ is tested against the alternative hypothesis $H_1: \mu \neq \mu_0$ for a two-sided test. Under the assumption that the data $(X_1, X_2, \dots, X_n)$ is from a symmetric continuous distribution, the differences $X_i - \mu_0$ for $i = 1, 2, \dots, n$ are calculated and $|X_i - \mu_0|$ for $i = 1, 2, \dots, n$ are ranked in ascending order. Then, the sum of positive ranks (W^+) and the sum of negative ranks (W^-) are calculated. By using them $W = \min(W^+, W^-)$ is determined. W is compared to the critical W values (W_α). If $W \leq W_\alpha$ the null hypothesis is rejected (Montgomery and Runger, 2010). In this study we execute a two-tail test for the significance level $\alpha = 0.5$.

Firstly, to compare if there is there a statistically meaningful difference between straight ordering and reverse ordering and decide which one of them is more advantageous, comparisons are made between E1 - E5 and E3 - E6.

For TSP, all the mechanisms that have reverse ordering (E5 and E6) greatly outperforms the ones with straight ordering (E1, E2, E3 and E4). Therefore, it is obvious that they are statistically different. In conclusion, reverse ordering is very suitable for TSP.

For QAP, all the mechanisms that have reverse ordering slightly outperforms the ones with straight ordering. A Wilcoxon signed ranked test is executed to see if there is a statistically meaningful difference between the mechanisms. The results are shown in Table 5.4 and Figure 5.6.

Because the p values are 0 (based on the significance level 0.05) for both comparisons, we reject the null hypothesis and say that there is a meaningful difference between the mechanisms. Nevertheless, we withhold to make any conclusions just yet. Because no experiment outperformed the swap mechanism yet.

Table 5.4. Wilcoxon signed ranks test statistics to compare ordering issue for QAP

	E1 – E5	E3 – E6
Z	-4.461 ^a	-5.687 ^a
Asymp. Sig. (2-tailed)	0	0

a. Based on negative ranks.

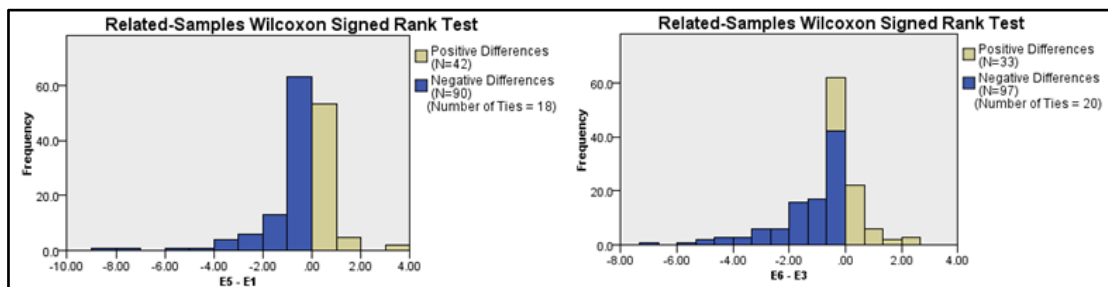


Figure 5.6. Positive and negative differences of performance data to compare ordering issue for QAP

Secondly, to compare if scheme-1 or scheme-2 more advantageous and if there is a statistically meaningful difference between them, the comparisons are made between E1 - E3, E2 - E4 and E5 - E6 because the only difference between them is that they either consist of scheme-1 or scheme-2. For TSP, there is no trend because E3 (uses scheme-2)

outperforms E1 (uses scheme-1). E2 and E5 (use scheme-1) outperforms E4 and E6 (use scheme-2). However, the best performance is given by E5.

For QAP, all the mechanisms that use scheme-1 slightly outperforms the ones with scheme-2. Because of this a Wilcoxon signed ranked test is executed to see if there is a statistically meaningful difference between the mechanisms. The results are shown in Table 5.5 and Figure 5.7.

Table 5.5 Wilcoxon signed ranks test statistics to compare neighbor generation scheme for QAP

	E1 – E3	E2 – E4	E5 – E6
Z	-8.894 ^a	-7.356 ^a	-7.957 ^a
Asymp. Sig. (2-tailed)	0	0	0

a. Based on positive ranks.

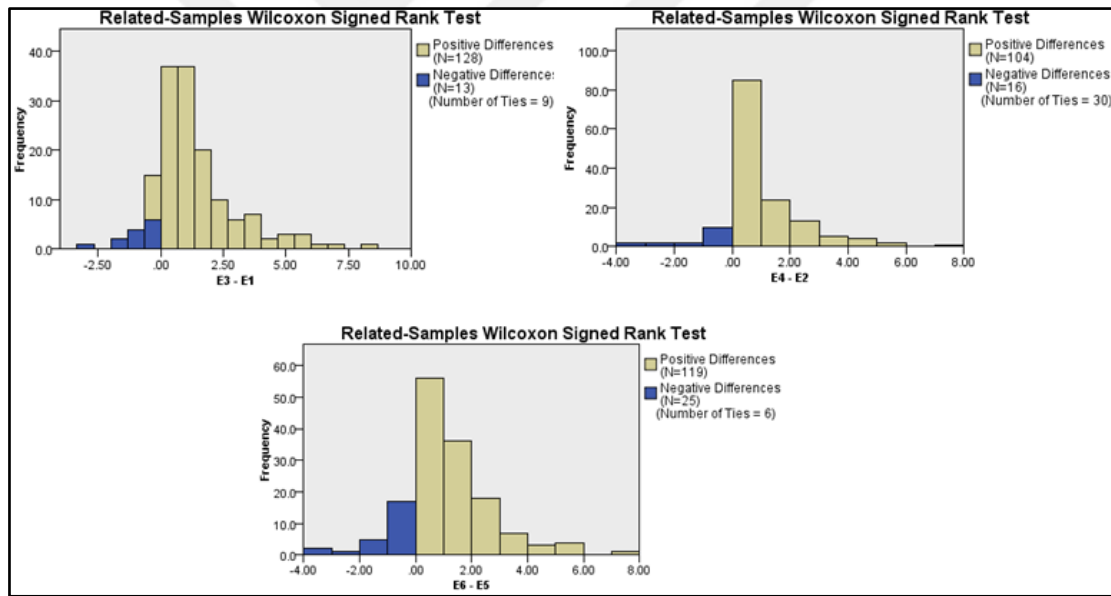


Figure 5.7. Positive and negative differences of performance data to compare neighbor generation scheme for QAP

According to the Wilcoxon signed ranked test, there is a statistically meaningful difference between E1 - E3, E2 - E4 and E5 - E6 (p values are 0 for all comparisons). Nevertheless, we withhold to make any conclusions just yet, because no experiment outperformed the swap mechanism yet.

Thirdly, to compare if $\text{size}(p_{i2}) \geq 1$ or $\text{size}(p_{i2}) = 1$ is more advantageous to be used in and / or scheme-1 and scheme-2, the comparisons are made between E1 - E2, E3 - E4.

For TSP, all the mechanisms that have $\text{size}(p_{i2}) \geq 1$ greatly outperforms the ones with $\text{size}(p_{i2}) = 1$. Therefore, we conclude that it is very suitable for TSP.

For QAP, all the mechanisms that have $\text{size}(p_{i2}) \geq 1$ slightly outperforms the ones with $\text{size}(p_{i2}) = 1$. In addition, according to the Wilcoxon signed ranked test results, which is shown in Table 5.6 and Figure 5.8, there is a meaningful difference between them. Nevertheless, we withhold to make any conclusions just yet, because no experiment outperformed the swap mechanism yet.

Table 5.6. Wilcoxon Signed Ranks Test Statistics to compare the size of p_{i2} for QAP

	E1 - E2	E3 - E4
Z	-9.945 ^a	-8.842 ^a
Asymp. Sig. (2-tailed)	0	0

a. Based on positive ranks.

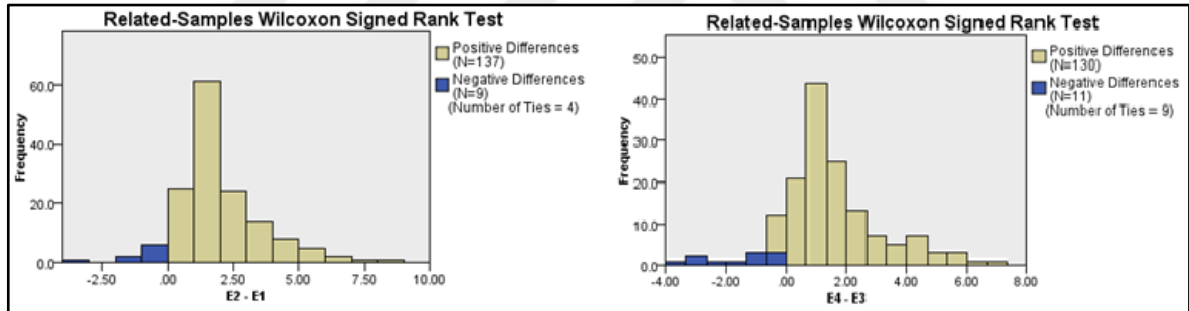


Figure 5.8. Positive and negative differences of performance data to compare the size of p_{i2} for QAP

5.2.2. Results of the stage-1 experiments

Conclusive decisions are very few in stage-1 experiment report. Therefore, some possibilities are marked as not concluded (NC). They can be observed in Table 5.7.

Even though, it is concluded that reverse ordering and $\text{size}(p_{i2}) \geq 1$ is advantageous for TSP, the experiments for the other combinations will continue in consideration of run-time. No other conclusions are made for other mechanisms. As a reminder E5 is the best mechanism for TSP at the current level of the experiments. In addition, it is obvious that there are statistically meaningful differences in the design choices. Therefore, promising

design issues which are the usage of reverse ordering, scheme-1, and $\text{size}(p_{i2}) \geq 1$ for both scheme-1 and scheme-2 will be experimented on stage-2 experiments.

Table 5.7. Conclusion summary for stage-1 experiments

Design Issue		TSP	QAP
Specific Dividing		NC	NC
Sequential dividing		NC	NC
Straight Ordering		✗	NC
Reverse Ordering		✓	NC
Scheme-1	$\text{size}(p_{i2}) \geq 1$	✓	NC
	$\text{size}(p_{i2}) = 1$	✗	NC
Scheme-2	$\text{size}(p_{i2}) \geq 1$	✓	NC
	$\text{size}(p_{i2}) = 1$	✗	NC

5.2.3.Stage-2 experiment report

In stage-2, first the neighbors are generated using stage-1 dividing and then the neighbors are generated using stage-2 dividing. It must be noted that all the design choices that is applied in stage-2 will be applied in stage-1 as well with one exception which is all stage-1 applications consist of sequential dividing. Notice that, in the experiments that consist of reverse ordering, the issue of applying the reverse ordering partially has also been considered. It is obvious that applying it to all the stages (stage-1: S1, stage-2: S2, stage-3: S3) will generate more neighbors since $(S1) \subset (S1-S2) \subset (S1-S2-S3)$. These experiments are added in consideration of run-time. The experiment design is shown in Figure 5.9.

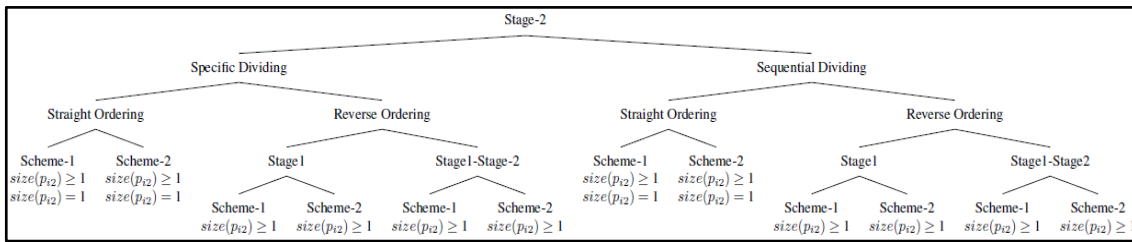


Figure 5.9. Structure of stage-2 experiments

The experiment list which is prepared to consist all combinations of the design choices that are previously deemed promising is given in the following list:

- E7: Stage-2, Specific dividing, Straight ordering, Scheme-1, $\text{size}(p_{i2}) \geq 1$

- E8: Stage-2, Specific dividing, Straight ordering, Scheme-1, $\text{size}(p_{i2}) = 1$
- E9: Stage-2, Specific dividing, Straight ordering, Scheme-2, $\text{size}(p_{i2}) \geq 1$
- E10: Stage-2, Specific dividing, Straight ordering, Scheme-2, $\text{size}(p_{i2}) = 1$
- E11: Stage-2, Specific dividing, Reverse ordering (S1), Scheme-1, $\text{size}(p_{i2}) \geq 1$
- E12: Stage-2, Specific dividing, Reverse Ordering(S1-S2), Scheme-1, $\text{size}(p_{i2}) \geq 1$
- E13: Stage-2, Specific dividing, Reverse ordering (S1), Scheme-2, $\text{size}(p_{i2}) \geq 1$
- E14: Stage-2, Specific dividing, Reverse ordering (S1-S2), Scheme-2, $\text{size}(p_{i2}) \geq 1$
- E15: Stage-2, Sequential dividing, Straight ordering, Scheme-1, $\text{size}(p_{i2}) \geq 1$
- E16: Stage-2, Sequential dividing, Straight ordering, Scheme-1, $\text{size}(p_{i2}) = 1$
- E17: Stage-2, Sequential dividing, Straight ordering, Scheme-2, $\text{size}(p_{i2}) \geq 1$
- E18: Stage-2, Sequential dividing, Straight ordering, Scheme-2, $\text{size}(p_{i2}) = 1$
- E19: Stage-2, Sequential dividing, Reverse ordering (S1), Scheme-1, $\text{size}(p_{i2}) \geq 1$
- E20: Stage-2, Sequential dividing, Reverse ordering (S1-S2), Scheme-1, $\text{size}(p_{i2}) \geq 1$
- E21: Stage-2, Sequential dividing, Reverse ordering (S1), Scheme-2, $\text{size}(p_{i2}) \geq 1$
- E22: Stage-2, Sequential dividing, Reverse ordering (S1-S2), Scheme-2, $\text{size}(p_{i2}) \geq 1$

The experimental results are presented in two parts. First set of tables consist the experiments that use specific dividing and the second set of tables consist the experiments that use sequential dividing. The performance in terms of solution quality of each experiment is given in Table 5.8 and Table 5.9.

Table 5.8 Performance of stage-2 experiments that use specific dividing in terms of solution quality

Instance	Swap	Insertion	E7	E8	E9	E10	E11	E12	E13	E14
Traveling Salesman Problem										
burma14	7.9	0.4	6.4	13.2	4.9	20.9	0.6	0.6	0.1	0.2
att48	44.6	17.8	22.7	61.8	20.9	71.8	2.3	2.4	3.1	2.8
st70	63.1	29.3	33.2	99.9	29.2	110.1	4.2	4.3	4.7	4.2
kroa100	102.9	42.6	44.9	140.7	50	148.7	4.1	4.2	4.5	4.7
ch150	102.5	38.1	56.7	164.3	52	178	6.9	5.9	6.4	6.2
Quadratic Assignment Problem										
Tai15a	4.4	19.6	10.2	13.3	12.7	10.9	9.1	9	10.9	10.9
Tai25a	5.4	19.2	12.4	15.1	14.1	13.2	12.2	11.9	13.2	13.2
Tai50a	4.8	16.8	12.6	14.5	13.7	12.8	12.4	12	12.8	12.8
Tai80a	3.7	13.6	10.8	12	11.3	10.9	10.7	10.6	10.8	10.9
Tai100a	3.4	12.5	10.2	11.5	10.8	10.4	9.9	9.9	10.4	10.4

In consideration of performance in terms of solution quality, E11, E12, E13 and E14 outperform both swap and insertion for the TSP. Therefore, they are chosen as candidates for the mechanisms for the TSP. For QAP, among the CB experiments, there is an improvement in the performance compared to stage-1 experiments. However, swap still gives the best performance. No candidate for the QAP can be found in this set of experiments too. Among E11, E12, E13 and E14, E12 generates the most neighbors but it cannot be said that E12 gives a better performance compared to the other candidates.

Table 5.9. Performance of stage-2 experiments that use sequential dividing in terms of solution quality

Instance	Swap	Insertion	E15	E16	E17	E18	E19	E20	E21	E22
Traveling Salesman Problem										
burma14	7.9	0.4	0.1	5.4	11.3	4.4	0	0	10.7	8.3
att48	44.6	17.8	3.8	33	28.2	33.5	1.6	0.8	25.7	21.4
st70	63.1	29.3	5.3	50.4	41.4	49.4	3.3	3	34.8	28.2
kroa100	102.9	42.6	5.4	79.7	45.3	80.4	3.1	1.6	46.6	42.4
ch150	102.5	38.1	7.9	77.7	63.3	79.6	4	3.3	57.2	48.6
Quadratic Assignment Problem										
Tai15a	4.4	19.6	7.2	9.5	12.5	3.7	7.2	10.3	11.2	10.3
Tai25a	5.4	19.2	8.9	11	12.8	4.4	8.8	11.4	12.5	11.4
Tai50a	4.8	16.8	8.6	10.6	11.7	4	8.7	11	11.5	11
Tai80a	3.7	13.6	7.2	8.5	9.4	2.8	7.3	8.9	9.5	8.9
Tai100a	3.4	12.5	6.9	8	8.9	2.6	6.9	8.2	8.9	8.2

E15, E19 and E20 outperform both swap and insertion for the TSP. Therefore, they are chosen as candidates for the mechanisms that solve the TSP. For QAP, E18 outperforms both swap and insertion mechanisms and chosen to be a candidate mechanism. Next, in the following tables (Table 5.10 and Table 5.11) the number of neighbors generated by stage-2 experiments that use specific dividing and sequential dividing until the algorithm is terminated are given, respectively.

Table 5.10. Number of neighbors generated by stage-2 experiments that use specific dividing

Instance	Swap	Insertion	E7	E8	E9	E10	E11	E12	E13	E14
Traveling Salesman Problem										
burma14	8.10^2	2.10^3	5.10^3	1.10^3	4.10^3	5.10^2	8.10^3	9.10^3	5.10^3	5.10^3
att48	5.10^4	1.10^5	3.10^5	2.10^4	3.10^5	1.10^4	5.10^5	6.10^5	3.10^5	3.10^5
st70	2.10^5	1.10^5	1.10^6	4.10^4	9.10^5	2.10^4	2.10^6	2.10^6	1.10^6	1.10^6
kroa100	6.10^5	1.10^6	3.10^6	8.10^4	2.10^6	5.10^4	6.10^6	7.10^6	3.10^6	3.10^6
ch150	2.10^6	5.10^6	1.10^7	2.10^5	9.10^6	1.10^5	2.10^7	2.10^7	1.10^7	1.10^7
Quadratic Assignment Problem										
Tai15a	9.10^2	5.10^2	3.10^3	5.10^2	2.10^3	2.10^3	5.10^3	6.10^3	2.10^3	2.10^3
Tai25a	5.10^3	2.10^3	1.10^4	1.10^3	8.10^3	8.10^3	2.10^4	2.10^4	8.10^3	8.10^3
Tai50a	4.10^4	9.10^3	6.10^4	3.10^3	4.10^4	5.10^4	1.10^5	1.10^5	5.10^4	5.10^4
Tai80a	2.10^5	2.10^4	2.10^5	5.10^3	1.10^5	1.10^5	3.10^5	4.10^5	2.10^5	1.10^5
Tai100a	3.10^5	4.10^4	3.10^5	6.10^3	2.10^5	3.10^5	6.10^5	7.10^5	3.10^5	3.10^5

Table 5.11. Number of neighbors generated by stage-2 experiments that use sequential dividing

Instance	Swap	Insertion	E15	E16	E17	E18	E19	E20	E21	E22
burma14	8.10^2	2.10^3	3.10^4	1.10^4	1.10^3	4.10^4	3.10^4	5.10^4	1.10^3	1.10^3
att48	5.10^4	1.10^5	6.10^6	3.10^6	5.10^4	1.10^7	6.10^6	1.10^7	6.10^4	5.10^4
st70	2.10^5	1.10^5	3.10^7	1.10^7	2.10^5	5.10^7	3.10^7	5.10^7	2.10^5	2.10^5
kroa100	6.10^5	1.10^6	1.10^8	8.10^7	5.10^5	2.10^8	1.10^8	2.10^8	5.10^5	5.10^5
ch150	2.10^6	5.10^6	7.10^8	4.10^8	2.10^6	1.10^9	7.10^8	1.10^9	2.10^6	2.10^6
Tai15a	9.10^2	5.10^2	2.10^4	4.10^3	7.10^2	5.10^4	2.10^4	7.10^2	7.10^2	1.10^3
Tai25a	5.10^3	2.10^3	2.10^5	2.10^4	2.10^3	4.10^5	2.10^5	3.10^3	3.10^3	4.10^3
Tai50a	4.10^4	9.10^3	3.10^6	1.10^5	2.10^4	7.10^6	3.10^6	2.10^4	2.10^4	2.10^4
Tai80a	2.10^5	2.10^4	2.10^7	6.10^5	6.10^4	5.10^7	2.10^7	7.10^4	6.10^4	8.10^4
Tai100a	3.10^5	4.10^4	5.10^7	1.10^6	1.10^5	1.10^8	5.10^7	1.10^5	1.10^5	2.10^5

E20 gives the best performance in terms of solution quality and creates the most neighbors while solving TSP. Similarly, E18 gives the best performance in terms of solution quality and creates the most neighbors while solving QAP. Finally, in Table 5.12 and Table 5.13, the run-time data is given.

Table 5.12. Run time of stage-2 experiments that use specific dividing (in seconds)

Instance	Swap	Insertion	E7	E8	E9	E10	E11	E12	E13	E14
Traveling Salesman Problem										
burma14	0.005	0.02	0.04	0.01	0.02	0.01	0.07	0.08	0.04	0.04
att48	0.13	0.67	1.82	0.13	0.92	0.07	3.36	3.79	1.57	1.78
st70	0.44	2.4	6.09	0.26	2.79	0.12	11.47	13.38	4.89	6.46
kroa100	1.66	8.97	20.21	0.62	9.38	0.3	43.41	48.76	17.69	24.34
ch150	7.95	39.83	78.56	1.52	38.69	0.73	148.49	189.21	67.18	85.07
Quadratic Assignment Problem										
Tai15a	0.01	0.01	0.05	0.02	0.03	0.07	0.07	0.11	0.05	0.05
Tai25a	0.03	0.02	0.19	0.03	0.08	0.26	0.28	0.38	0.14	0.16
Tai50a	0.89	0.23	2.33	0.21	0.97	4.38	4.05	5.64	1.87	2.22
Tai80a	8.73	1.46	16.14	0.94	6.82	24.8	25.06	32.89	11.1	13.38
Tai100a	27.49	3.88	41.66	1.72	15.46	71.05	77.54	94.35	29.21	36.8

Table 5.13. Run time of stage-2 experiments that use sequential dividing (in seconds)

Instance	Swap	Insertion	E15	E16	E17	E18	E19	E20	E21	E22
Traveling Salesman Problem										
burma14	0.01	0.02	0.04	0.04	0.03	0.04	0.1	0.1	0.04	0.04
att48	0.1	0.7	3.9	7.7	7.8	8.1	5.4	7	7.8	12
st70	0.4	2.4	17	48	34	52	19	32	42	79
kroa100	1.7	9	81	225	207	284	89	128	224	373
ch150	7.9	39	427	1084	1098	1866	448	664	1149	2256
Quadratic Assignment Problem										
Tai15a	0.01	0.01	0.2	0.1	0.03	0.1	0.3	0.1	0.04	0.1
Tai25a	0.03	0.02	2.2	0.3	0.2	1.2	2.4	0.6	0.3	0.6
Tai50a	0.9	0.2	113	5.4	11	58.3	111	26	11	23
Tai80a	8.7	1.5	1560	54	169	1062	1637	374	187	394
Tai100a	27.5	3.9	6111	157	639	3830	6372	1553	634	1624

E20 which is the best performing mechanism for TSP is not the slowest mechanism, but it is very slow to solve the TSP. E18 suffers from a similar impediment for QAP.

For TSP, all mechanisms, except E15, that give a good performance use reverse ordering. It means that reverse ordering is very suitable for TSP as it adds variation to the neighborhood to solve a problem that has a steep landscape. E15 is the only experiment that has straight ordering which gives a good performance. Consequently, the design combination of E15 will be added to stage-3 experiments just in case. Only one

combination is effective for QAP, and since there is a big performance difference between E18 and the other methods, it can safely be said that it has an ideal combination for QAP. Scheme-1 and scheme-2 work for solving TSP effectively as long as the mechanism also use $\text{size}(p_{i2}) \geq 1$. For QAP, sequential dividing, scheme-2 with combination $\text{size}(p_{i2}) = 1$ works best.

5.2.4. Results of the stage-2 experiments

Conclusive decisions obtained from both stage-1 and stage-2 experiments can be observed in Table 5.14. There are not additional conclusions for design issues when solving TSP compared to the conclusions arrived in stage-1 experiment report. However, it is concluded that the design issues sequential dividing, straight ordering, scheme-2, and $\text{size}(p_{i2}) = 1$ is suitable for CB when solving QAP.

Table 5.14. Conclusion summary of stage-2 experiments

Design Issue		TSP	QAP
Specific Dividing		NC	✗
Sequential dividing		NC	✓
Straight Ordering		✗	✓
Reverse Ordering		✓	✗
Scheme-1	$\text{size}(p_{i2}) \geq 1$	✓	✗
	$\text{size}(p_{i2}) = 1$	✗	✗
Scheme-2	$\text{size}(p_{i2}) \geq 1$	✓	✗
	$\text{size}(p_{i2}) = 1$	✗	✓

Pairwise bar chart comparisons of run-time and performance are given in Figure 5.11 for TSP and in Figure 5.12 for QAP. For TSP, E11, E12, E13, E14, E15, E19, and E20 give similar results in varying run-times. Therefore, the balance of performance in terms of solution quality and run-time needs to be considered in stage-3 experiments. For QAP, only E18 gave the satisfying results. However, the run-time of E18 is more than swap which is the other mechanism that gives satisfying results consistently. Therefore, the run-time of CB needs to be lowered.

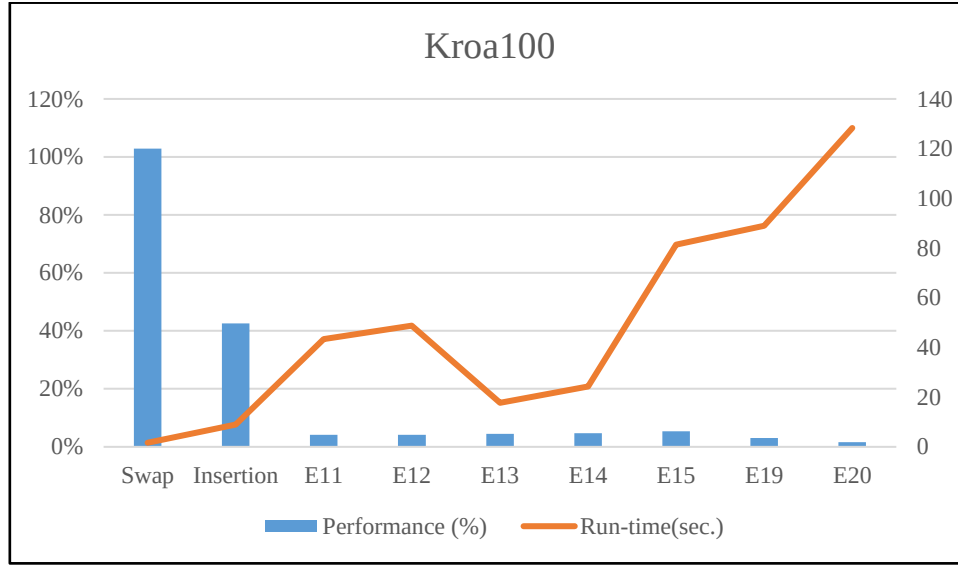


Figure 5.10. Pairwise bar chart comparisons of run-time solving TSP by stage-2 experiments

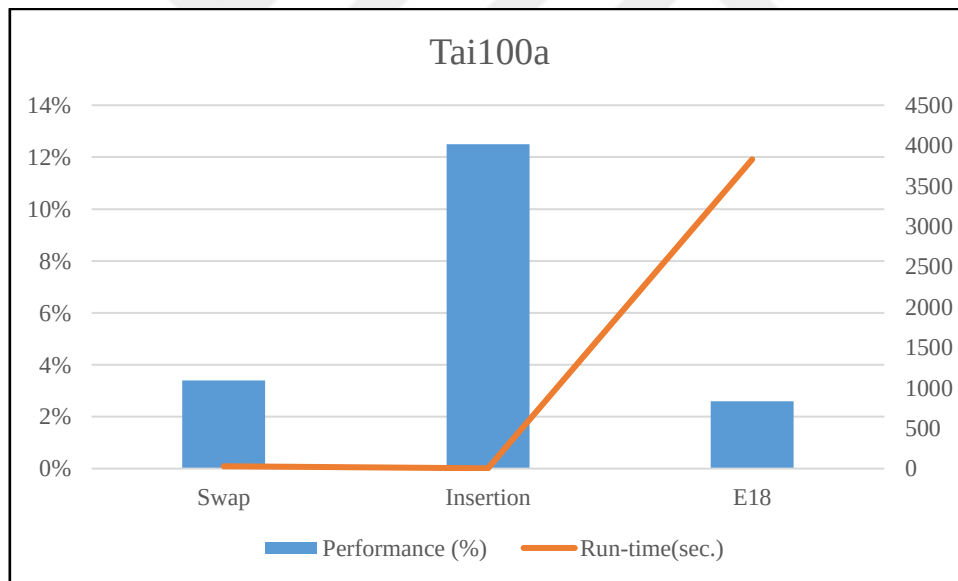


Figure 5.11. Pairwise bar chart comparisons of run-time solving QAP by stage-2 experiments

It is observed that the experiments that give the better performances in terms of solution quality are too slow. To make them faster, the following design choice which is the fifth design issue is implemented to the algorithms:

- *The fifth design issue* is that the neighbor that gives the minimum objective function value in the previous stage is picked and the process of the next stage is applied to the selected neighbor instead of complete set of neighbors of just previous stage.

Obviously, the time the algorithm takes for an iteration remains the same with the addition of this design choice. However, the algorithm in which the CB mechanism is embedded converges to a local optimum faster. Therefore, the number of iterations is reduced which makes the algorithm execute faster. The relevant observation can be obtained from Table 5.15 for solution quality and from Table 5.16 for run-time. All the experiments designed for stage-3 contain this design choice. Because the algorithms that have stage-3 dividing process would be slower than stage-2 with the same design choices as the 3rd stage would add more steps to the algorithm.

Table 5.15. Performance comparison in terms of solution quality after incorporating the fifth design issue

	TSP														QAP		
Instance	E11		E12		E13		E14		E15		E19		E20		E18		Instance
	B*	A*	B	A	B	A	B	A	B	A	B	A	B	A	B	A	
burma14	0.6	0.25	0.63	0.38	0.11	0.18	0.25	0.32	0.12	0.37	0	0	0	0	3.73	3.84	Tai15a
att48	2.31	2.51	2.36	2.46	3.11	2.65	2.78	2.5	3.77	3.25	1.61	1.52	0.83	1.21	4.36	4.17	Tai25a
st70	4.16	4.31	4.26	3.81	4.71	3.76	4.19	4.54	5.28	5.52	3.32	3.23	3.02	3.05	3.96	3.89	Tai50a
kroa100	4.06	4.08	4.21	3.33	4.55	4.18	4.68	3.71	5.42	6.02	3.15	3.05	1.55	2.33	2.83	2.64	Tai80a
ch150	6.86	6.05	5.86	6.13	6.41	5.6	6.17	6.05	7.86	7.44	3.98	3.47	3.34	3.12	2.58	2.45	Tai100a

B*: Before the fifth design issue is incorporated, A*: After the fifth design issue is incorporated

Table 5.16. Run-time (in seconds) comparison after incorporating the fifth design issue

	TSP														QAP		
	E11		E12		E13		E14		E15		E19		E20		E18		
Instance	B*	A*	B	A	B	A	B	A	B	A	B	A	B	A	B	A	Instance
burma14	0.07	0.06	0.08	0.06	0.04	0.03	0.04	0.04	0.04	0.03	0.07	0.03	0.08	0.04	0.08	0.04	Tai15a
att48	3.36	2	3.79	2.42	1.57	0.92	1.78	1.2	3.94	0.52	5.4	0.67	7.02	0.96	1.23	0.34	Tai25a
st70	11.5	7.23	13.4	7.81	4.89	3.34	6.46	3.95	17.8	1.28	18.9	1.93	32.4	2.5	58.3	11.3	Tai50a
kroa100	43.4	23.9	48.8	26.6	17.7	12.9	24.3	13.8	81.4	3.92	89	5.88	128	8.08	1062	129	Tai80a
ch150	148	92.7	189	104	67.2	43.6	85.1	49.4	427	14	449	25.5	664	28.6	3830	404	Tai100a

B*: Before the fifth design issue is incorporated, A*: After the fifth design issue is incorporated

Wilcoxon signed rank test is executed to see if there is a statistically meaningful difference between the mechanisms that use the main neighbor and their best neighbor counterparts. The descriptive statistics are shown in Table 5.17, Wilcoxon signed rank test results are shown in Table 5.18.

Table 5.17. Descriptive statistics

		N	Mean	Std. Deviation	Minimum	Maximum
E11	B	150	3.60	2.68	0	10.34
	A	150	3.44	2.60	0	10.37
E12	B	150	3.46	2.53	0	10.43
	A	150	3.22	2.47	0	11.54
E13	B	150	3.78	2.83	0	12.07
	A	150	3.27	2.39	0	9.71
E14	B	150	3.61	2.78	0	14.51
	A	150	3.42	2.69	0	10.97
E15	B	150	4.49	3.22	0	10.69
	A	150	4.52	3.20	0	11.96
E18	B	150	3.49	1.09	0.87	7.15
	A	150	3.40	1.15	0.99	6.43
E19	B	150	2.41	2.09	0	8.49
	A	150	2.25	1.90	0	7.01
E20	B	150	1.75	1.78	0	7.75
	A	150	1.94	1.71	0	7.75

Table 5.18. Wilcoxon Signed Ranks test statistics

	E11.B – E11.A	E12.B – E12.A	E13.B – E13.A	E14.B – E14.A	E15.B – E15.A	E18.B – E18.A	E19.B – E19.A	E20.B – E20.A
Z	-1.079 ^a	-1.266 ^a	-2.289 ^a	-.333 ^a	-.171 ^b	-1.485 ^a	-.443 ^a	-1.761 ^b
Asymp. Sig. (2- tailed)	0.281	0.206	0.022	0.739	0.864	0.138	0.658	0.078

a. Based on negative ranks.

b. Based on positive ranks.

According to Wilcoxon signed rank test experiments, E13 shows a statistically meaningful difference ($p = 0.022 < 0.05$) after the design issue is incorporated. There is no specific design choice that is detected to explain this difference. Other experiments have p values that are more than the significance level (based on the significance level

0.05) meaning that we accept the null hypothesis which means that there is no statistically meaningful difference between the compared experiments after the design issue is incorporated.

5.2.5.Stage-3 experiment report

The experimental design is shown in Figure 5.12. In stage-3 neighbor generation, as discussed in stage-1 and stage-2 experiments, $\text{size}(p_{i2}) = 1$ with reverse ordering is not included in the tests because those designs would give the exact results with their counterparts that use straight ordering. Therefore, the experiment that use sequential dividing, straight ordering, scheme-1, $\text{size}(p_{i2}) = 1$ is added to the experiments to be able to compensate for the fact that, that design combination would not be chosen from stage-2 experiments, therefore would not be represented in stage-3 experiments otherwise. The other combinations are the stage-3 variations of the experiments that gave promising results in stage-2 experiments. The experiment list is given below:

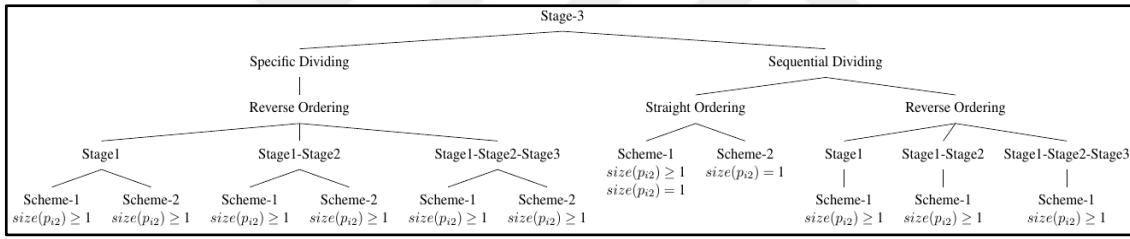


Figure 5.12. Structure of stage-3 experiments

- E23: Stage-3, Specific dividing, Reverse ordering(S1), Scheme-1, $\text{size}(p_{i2}) \geq 1$
- E24: Stage-3, Specific dividing, Reverse ordering(S1-S2), Scheme-1, $\text{size}(p_{i2}) \geq 1$
- E25: Stage-3, Specific dividing, Reverse ordering(S1-S2-S3), Scheme-1, $\text{size}(p_{i2}) \geq 1$
- E26: Stage-3, Specific dividing, Reverse ordering(S1), Scheme-2, $\text{size}(p_{i2}) \geq 1$
- E27: Stage-3, Specific dividing, Reverse ordering(S1-S2), Scheme-2, $\text{size}(p_{i2}) \geq 1$
- E28: Stage-3, Specific dividing, Reverse ordering(S1-S2-S3), Scheme-2, $\text{size}(p_{i2}) \geq 1$
- E29: Stage-3, Sequential dividing, Straight ordering, Scheme-1, $\text{size}(p_{i2}) \geq 1$
- E30: Stage-3, Sequential dividing, Straight ordering, Scheme-1, $\text{size}(p_{i2}) = 1$
- E31: Stage-3, Sequential dividing, Straight ordering, Scheme-2, $\text{size}(p_{i2}) = 1$
- E32: Stage-3, Sequential dividing, Reverse ordering(S1), Scheme-1, $\text{size}(p_{i2}) \geq 1$
- E33: Stage-3, Sequential dividing, Reverse ordering(S1-S2), Scheme-1, $\text{size}(p_{i2}) \geq 1$

- E34: Stage-3, Sequential dividing, Reverse ordering(S1-S2-S3), Scheme-1, $\text{size}(p_{i2}) \geq 1$

The performance in terms of solution quality of each experiment is given in Table 5.19. It is observed that every experiment, except for E30 and E31, give better results than swap and insertion mechanisms while solving TSP. With consideration of stage-2 results, this result was expected as E30 and E31 use straight ordering and $\text{size}(p_{i2}) = 1$ combination which is not suitable for TSP. The reason E30 performs better than E31 is that E30 uses scheme-1 neighbor generation which is more suitable to TSP. For QAP, E31 gives the best results which are similar to E18. However, a curious observation is that E33 and E34 give good results for both TSP and QAP. Even though they do not give the best results that CB experiments can produce, the fact that a mechanism gives good results for the problems that have very different characteristics needs exploring.

The number of neighbors generated until the algorithm is terminated is given in Table 5.20. The number of produced neighbors is determined by the sophistication of each individual experiment. However, if a design combination is not suitable for the problem, generation of more neighbors do not necessarily mean a better search capability therefore better performance. The run-time is given in Table 5.21. The run-time is affected by two factors: number of neighbor generation (steps in the algorithm) and efficiency (convergence to a local minimum) of the algorithm.

Table 5.19. Performance of stage-3 experiments in terms of solution quality

Instance	Swap	Insertion	E23	E24	E25	E26	E27	E28	E29	E30	E31	E32	E33	E34
Traveling Salesman Problem														
burma14	7.9	0.4	0.1	0.3	0.3	0.1	0.1	0.1	0.2	0.7	5.0	0	0	0
att48	44.6	17.8	2.3	2.4	2.5	3.0	2.5	3.1	3.5	6.3	33.4	1.5	1.7	1.4
st70	63.1	29.3	4.4	3.6	3.6	3.8	4.1	3.6	5.7	10.3	51.6	3.4	3.5	2.9
kroa100	102.9	42.6	3.5	4.0	5.4	4.7	4.5	3.9	5.7	14.7	78.2	2.4	3.1	2.8
ch150	102.5	38.1	5.4	5.7	5.3	6.7	6.7	5.4	7.2	19.0	80.4	3.3	3.8	3.5
Quadratic Assignment Problem														
Tai15a	4.4	19.6	9.5	9.2	9.0	10.4	10.4	10.4	7.3	10.2	4.0	6.7	5.5	5.5
Tai25a	5.4	19.2	11.4	11.4	10.6	11.6	11.6	11.5	8.5	10.9	4.5	8.7	8.0	8.2
Tai50a	4.8	16.8	11.3	11.1	11.0	10.9	10.7	10.6	8.6	10.5	4.0	8.5	7.6	7.8
Tai80a	3.7	13.6	9.8	9.6	9.7	8.9	8.9	8.7	6.9	8.4	2.8	7.0	6.5	6.5
Tai100a	3.4	12.5	9.4	9.3	9.2	8.4	8.2	8.1	6.6	8.0	2.6	6.5	6.1	6.0

Table 5.20. Number of neighbors generated by of stage-3 experiments

Instance	Swap	Insertion	E23	E24	E25	E26	E27	E28	E29	E30	E31	E32	E33	E34
Traveling Salesman Problem														
burma14	8.10^2	2.10^3	6.10^3	6.10^3	6.10^3	1.10^4	1.10^4	5.10^3	1.10^4	3.10^3	1.10^4	1.10^4	1.10^4	1.10^4
att48	5.10^4	1.10^5	3.10^5	3.10^5	3.10^5	8.10^5	9.10^5	3.10^5	8.10^5	8.10^4	1.10^6	1.10^6	1.10^6	1.10^6
st70	2.10^5	1.10^5	9.10^5	1.10^6	1.10^6	3.10^6	3.10^6	9.10^5	3.10^6	2.10^5	6.10^6	4.10^6	3.10^6	3.10^6
kroa100	6.10^5	1.10^6	3.10^6	3.10^6	3.10^6	1.10^7	1.10^7	3.10^6	8.10^6	5.10^5	3.10^7	1.10^7	1.10^7	1.10^7
ch150	2.10^6	5.10^6	9.10^6	1.10^7	1.10^7	3.10^7	4.10^7	1.10^7	3.10^7	1.10^6	1.10^8	5.10^7	5.10^7	4.10^7
Quadratic Assignment Problem														
Tai15a	9.10^2	5.10^2	3.10^3	4.10^3	4.10^3	5.10^3	5.10^3	2.10^3	1.10^4	6.10^3	1.10^4	1.10^4	2.10^4	2.10^4
Tai25a	5.10^3	2.10^3	1.10^4	2.10^4	2.10^4	3.10^4	2.10^4	8.10^3	7.10^4	3.10^4	1.10^5	7.10^4	1.10^5	1.10^5
Tai50a	4.10^4	9.10^3	9.10^4	1.10^5	1.10^5	3.10^5	3.10^5	8.10^4	9.10^5	3.10^5	2.10^6	9.10^5	2.10^6	2.10^6
Tai80a	2.10^5	2.10^4	3.10^5	3.10^5	4.10^5	2.10^6	1.10^6	4.10^5	5.10^6	2.10^6	1.10^7	4.10^6	1.10^7	8.10^6
Tai100a	3.10^5	4.10^4	5.10^5	5.10^5	6.10^5	3.10^6	3.10^6	8.10^5	1.10^7	4.10^6	3.10^7	1.10^7	2.10^7	2.10^7

Table 5.21. Run time of stage-3 experiments (in seconds)

Instance	Swap	Insertion	E23	E24	E25	E26	E27	E28	E29	E30	E31	E32	E33	E34
Traveling Salesman Problem														
burma14	0.005	0.02	0.1	0.1	0.1	0.03	0.04	0.04	0.1	0.03	0.1	0.1	0.1	0.1
att48	0.1	0.7	1.8	2.5	2.0	1.0	1.2	1.7	1.6	0.3	5.8	2.0	2.3	3.2
st70	0.4	2.4	6.0	8.2	6.3	3.8	4.0	5.4	4.7	0.6	23.9	7.1	6.5	10.6
kroa100	1.7	9.0	20.4	25.6	22.5	11.8	14.8	17.4	16.5	1.4	118.8	24.4	23.6	32.4
ch150	7.9	39.8	76.1	87.3	81.4	44.0	60.9	65.6	62.4	4.8	518.6	115.5	111.6	144.1
Quadratic Assignment Problem														
Tai15a	0.01	0.01	0.1	0.1	0.1	0.1	0.04	0.05	0.2	0.1	0.1	0.2	0.2	0.4
Tai25a	0.03	0.02	0.4	0.3	0.4	0.4	0.2	0.2	1.5	0.5	0.8	1.2	3.0	2.7
Tai50a	0.9	0.2	3.5	3.9	4.1	15.8	3.8	3.3	43.9	14.0	39.4	34.5	82.4	78.2
Tai80a	8.7	1.5	22.8	28.3	33.5	260.1	30.2	34.5	388.0	154.4	689.4	359.0	773.8	697.4
Tai100a	27.5	3.9	61.2	71.6	79.0	1017.6	118.5	112.3	1176.7	449.2	2528.1	1242.1	2541.8	2593.9

It must be noted that E30, while being worse than other CB variations, is considerably faster than them and gives results better than insertion while solving TSP. On the other hand, E30 is too slow while solving QAP.

Finally, pairwise comparison charts for solution quality performance and run-time of all the chosen experiments are presented (solving $n = 100$ problems). Figure 5.13 shows the comparisons solving TSP and Figure 5.14 shows the comparisons solving QAP. E20 is the fastest and most effective mechanism for TSP. That is why E20 is chosen over E32 which is equally effective as E20 but slower than E20. E18 is the fastest and most effective mechanism for QAP. Similarly, E18 is chosen over E31 because E18 is faster than E31.

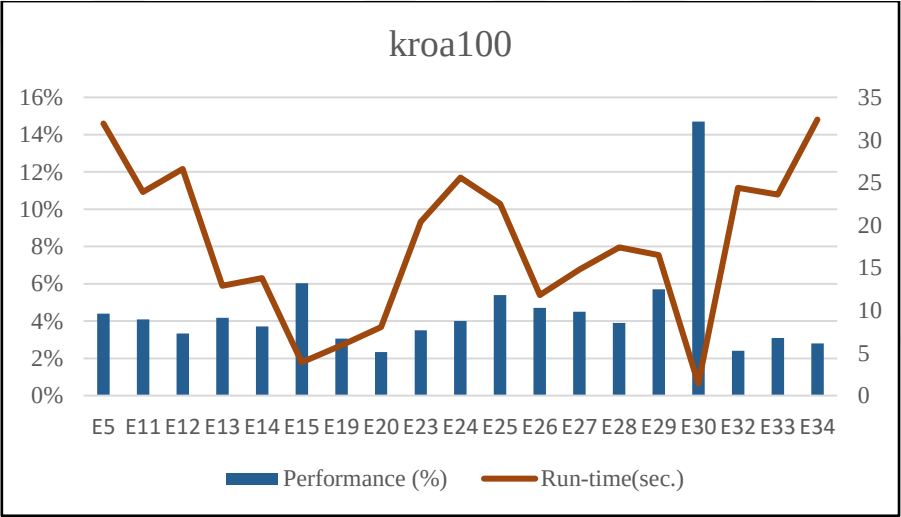


Figure 5.13. Pairwise comparisons of chosen mechanisms for TSP

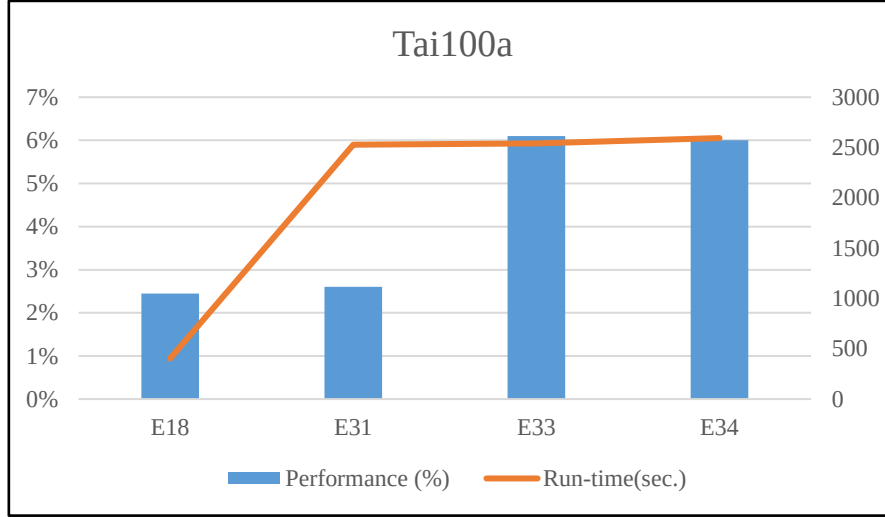


Figure 5.14. Pairwise comparisons of chosen mechanisms for QAP

As mentioned before, E33 and E34 solve both problems with a good performance of some degree. In terms of neighborhood, E34 generates all neighbors E33 can generate and then generates some unique ones as well. Therefore, E34 is included to chosen mechanisms.

5.3. Performance of CB Method on Additional Test Instances

In this section, E18 (gives the best results for QAP), E20 (gives the best results for TSP) and E34 (gives good results for both TSP and QAP) are experimented on using additional test instances. The reason to expand the experiments on additional instances is to further analyze the efficiency capability and robustness of the mechanisms.

Previously, the experiments were done by embedding the mechanisms into LS algorithm. In addition, the randomized versions of the chosen mechanisms are embedded into SA algorithm to analyze how the mechanisms perform embedded into a stochastic algorithm. As a side note, for all the instances chosen for TSP experiments, the optimal solution is known. But for QAP instances, only the optimal solutions of Tai25b, Chr25a, Nug27, Tho30, Nug30, Lipa90b are known. For the other instances, lower bounds are used in the calculation of solution quality.

5.3.1. Randomization of CB mechanisms

Until this point, experimental results are obtained from deterministic neighbor generations. Because the comparisons needed to be made in a deterministic environment to be able to obtain unquestionable results. For that reason, certain design issues needed

to be considered. Advantageous designs are chosen according to experimental results. In this section, the stochastic versions of the selected variants are tested, embedded into SA.

When the variants of CB mechanism are generated randomly, the number of stages that CB proceeds is still important however it is chosen randomly. Dividing procedure of the strings in each stage is not important as a point to start the dividing and the length of the p_{i2} is determined randomly. Reverse ordering of substrings is important because if a mechanism uses this design choice, neighbors are generated randomly using either straight or reverse ordering. However, the placement of the reverse ordering in each stage or selected stages is not important because it is chosen randomly. Neighbor generation scheme is important because this choice determines how the neighbors are generated. And as for the final design issue using the best solution instead of main solution is redundant as only one neighbor is generated in each iteration, not a neighborhood.

Because of the random determination of the time-consuming design issues, SA performs faster than LS.

5.3.2. Parameter tuning for simulated annealing

Geometric cooling schedule is used for the sake of simplicity as the purpose of the experiments is to compare the neighbor generation mechanisms and overall performance of the algorithm against the benchmark problems have second degree importance.

In geometric cooling schedule, the parameter set that need to be determined is cooling rate (α), initial temperature (T_0), final temperature (T_f) and the equilibrium state (ϑ). An experimental design is constructed to determine the proper parameter set for SA. An additional experiment set is executed for this purpose. For TSP and QAP small, medium, and large sized problem instances have been chosen. For all combinations of the parameter values that are given in the following list, the experiments are made 30 times starting from the same 30 initial solutions. A total of 4320 tests are recorded for each problem:

1. $\alpha = 0.90, 0.95, \text{ and } 0.99$
2. $T_0 = n, 5n, 50n \text{ and } 100n$
3. $T_f = 0.1n, 0.01n \text{ and } 0.001n$
4. $\vartheta = n, 5n, 10n \text{ and } 15n$

Notice that some parameter values are given as functions of n .

For TSP, the following parameters are determined:

1. $\alpha = 0.99$,
2. $T_0 = 5n$,
3. $T_f = 0.001n$,
4. $\vartheta = 15n$.

For QAP, the following parameters are determined:

1. $\alpha = 0.99$,
2. $T_0 = 100n$,
3. $T_f = 0.001n$,
4. $\vartheta = 5n$.

5.3.3. Local search experiments

Swap, insertion, E18, E20 and E34 are compared in terms of performance for solving TSP and QAP embedded into LS to further analyze the robustness of the mechanisms. Also, in the additional tests n is bigger than the previous experiments to see how the runtime will grow and how the performance will be affected. In Table 5.22 and Table 5.23, the performance results in terms of solution quality are given. CB variants perform similarly to the previous experiments which proves that they are indeed robust while solving both TSP and QAP. The mechanisms also give consistent performances for smaller n and bigger n .

Table 5.22. Performance of chosen variants while solving TSP embedded into LS in terms of solution quality

Instance	Swap	Insertion	E18	E20	E34
krob100	89.06	35.12	72.40	2.50	2.63
lin105	111.17	38.89	79.58	1.99	2.77
bier127	58.72	27.83	43.20	2.59	3.72
ch130	96.32	41.84	71.39	2.47	3.83
kroa150	116.96	45.81	84.45	2.88	3.33
krob150	125.15	48.53	92.50	2.24	3.38
ch150	102.59	38.23	75.82	3.19	3.58
d198	153.70	56.33	118.27	1.60	1.86
kroa200	143.44	56.27	105.44	3.22	3.73
krob200	146.37	51.46	99.92	4.28	4.19
gil262	142.27	52.23	102.07	4.20	4.51
pr264	393.17	141.29	305.88	4.92	5.53
pr299	191.64	54.28	125.99	4.36	4.60
lin318	187.93	71.93	139.10	3.29	3.59
fl417	729.35	127.08	489.05	1.96	2.26

Table 5.23. Performance of chosen variants while solving QAP embedded into LS in terms of solution quality

Instance	Swap	Insertion	E18	E20	E34
Tai25b	17.42	48.89	12.62	25.83	16.91
Chr25a	60.84	167.53	50.95	134.41	92.17
Nug27	4.32	14.54	3.60	13.98	6.98
Tho30	4.98	12.32	3.71	12.42	6.96
Nug30	4.32	12.99	2.79	11.75	6.76
Esc32h	4.03	16.38	2.16	11.90	5.54
Esc32d	6.03	33.27	4.93	20.23	13.80
Tho40	4.80	15.22	3.01	15.08	7.80
Wil50	1.55	5.02	1.21	5.40	2.54
Tai60a	4.96	10.15	3.37	10.17	7.49
Tai60b	7.79	26.22	5.41	17.53	13.46
Tai80b	6.11	24.06	4.96	13.63	10.65
Lipa90b	21.93	25.72	21.07	25.76	24.06
Tai100b	5.73	22.92	3.90	20.32	10.82
Wil100	1.07	5.17	0.84	4.58	2.42

E34 again gives a good performance for both TSP and QAP while not being the best mechanism for both. E20 gives the best results for TSP while being relatively robust but

it performs slowly. For QAP, E18 gives the best results while being more robust than swap. However, it performs slower than swap. The relative run-time data is given in Table 5.24 and Table 5.25 for TSP and QAP, respectively. For TSP, E20 gives the best performance and the fastest mechanism among the CB variants. For QAP, E18 gives the best performance and it is the fastest mechanism among CB variants. However, it is slower than swap mechanism which gives a similar performance. For both TSP and QAP, E34 is too slow while giving relatively good results.

Table 5.24. Average time required (in seconds) for TSP embedded into LS

	Swap	Insertion	E18	E20	E34
krob100	1.7	9.6	11.3	8.3	32.6
lin105	2	10.5	14.5	9.4	38.4
bier127	4	19	23.2	16.2	74
ch130	4.5	21.7	26.8	18.3	78.9
kroa150	9.1	40.8	51	29.1	127.5
krob150	8.8	37.7	53.7	27.9	121.5
ch150	8.1	34.8	45.5	26.4	138
d198	23.1	107.5	158	71.5	369.5
kroa200	23.2	112.7	128.6	72.1	351.1
krob200	23.9	105.5	129.4	70.1	380.4
gil262	62.6	273.7	421.7	164.5	961.7
pr264	63.5	271.3	429.3	182.2	881.4
pr299	115.3	475.6	679.4	257.5	1447.8
lin318	134.3	548.8	848	302	1659.2
fl417	440.6	1750.2	2752.9	721.1	4362.8
Average	61.65	254.63	384.89	131.77	734.99

Table 5.25. Average time required (in seconds) for QAP embedded into LS

	Swap	Insertion	E18	E20	E34
Tai25b	0.1	0.1	0.3	0.9	2.4
Chr25a	0.03	0.05	0.3	0.5	1.9
Nug27	0.1	0.1	0.4	0.6	3.2
Tho30	0.1	0.2	0.8	1.6	4.8
Nug30	0.1	0.1	0.9	1.5	4.7
Esc32h	0.1	0.1	0.8	1.3	5.1
Esc32d	0.1	0.1	0.7	1.2	4.9
Tho40	0.4	0.6	3.6	6.9	20.2
Wil50	1.5	2	11.6	23.5	55
Tai60a	2.1	2.1	28.8	43	159.1
Tai60b	3.8	4.5	33	73	205.9
Tai80b	17.6	15.7	114.1	279.4	1063.7
Lipa90b	16.5	15.3	220.4	296.8	1197.2
Tai100b	59.1	53.2	416.9	634.3	3585.7
Wil100	59	59.2	502.2	866.9	2334.8
Average	10.71	10.22	88.99	148.76	576.57

It is important to note that the additional experiment results and the design experiment results are the same for both TSP and QAP. This proves that CB variants are robust. Also, E34 is proven to perform relatively well for both problems that have very different landscapes in their solution spaces. In conclusion it is advisable to start from that variant first while solving a problem because it can respond more general problem characteristics.

Table 5.26 and Table 5.27 show the descriptive statistics of percentage deviation from the best-known solutions for TSP and QAP, respectively. The tables show the data comparison for both design experiments and additional experiments. Similar to the results of the design experiments, E18 solving QAP and E20 solving TSP outperform swap and insertion mechanisms. The mechanisms are also proven to have consistent performances. However, they are very slow, and the algorithm designs need improving.

Table 5.26. Descriptive statistics of the mechanisms in terms of percentage deviation from the best-known solution for TSP embedded into LS

	N	Minimum	Maximum	Mean	Std. Deviation
Design Experiments					
Swap	150	0	129.34	64.20	38.23
Insertion	150	0	60.72	25.63	17.22
E18	150	0	114.27	49.45	31.01
E20	150	0	7.75	1.75	1.78
E34	150	0	7.80	2.12	1.80
Additional Experiments					
Swap	450	40.34	890.45	185.86	165.31
Insertion	450	17.94	196.8	59.14	33.87
E18	450	28.76	585.57	133.67	113.36
E20	450	0.03	13.41	3.05	1.70
E34	450	0	13.99	3.57	1.80

Table 5.27. Descriptive statistics of the mechanisms in terms of percentage deviation from the best-known solution for QAP embedded into LS

	N	Minimum	Maximum	Mean	Std. Deviation
Design Experiments					
Swap	150	0.87	8.44	4.35	1.33
Insertion	150	11.68	26.29	16.35	3.47
E18	150	0.87	7.15	3.49	1.09
E20	150	3.24	15.01	9.98	1.78
E34	150	1.59	10.54	6.80	1.49
Additional Experiments					
Swap	450	0	98.74	10.39	15.33
Insertion	450	3.27	222.71	29.36	39.78
E18	450	0	74.03	8.3	13.25
E20	450	3.58	174.45	22.87	31.13
E34	450	0.72	123.55	15.22	21.99

5.3.4. Simulated annealing experiments

SA experiments are executed to analyze how stochastic CB variants perform. One of the advantages of designing CB stochastically is that since everything is decided randomly, there are fewer design choices that matters than deterministic CB, as explained previously. SA executes considerably faster than LS while giving a similar performance. In Table 5.28 and Table 5.29, the performance results are given. Similar to LS, E20 and E34 give a good performance for TSP and E18 gives a good performance for QAP.

Table 5.28. Performance of chosen variants while solving TSP embedded into SA in terms of solution quality

	Swap	Insertion	E18	E20	E34
krob100	19.88	6.12	10.07	1.35	1.52
lin105	23.85	8.43	15.86	1.04	1.28
bier127	22.45	7.91	14.67	1.85	2.10
ch130	36.56	24.40	15.36	2.39	2.43
kroa150	160.36	146.43	15.14	2.49	2.77
krob150	159.17	147.25	13.23	2.22	2.22
ch150	134.76	122.47	15.61	3.39	3.99
d198	426.78	420.16	18.22	1.62	1.71
kroa200	419.11	408.20	19.34	3.69	3.72
krob200	410.76	399.82	18.85	3.65	4.25
gil262	562.86	555.05	26.26	6.16	6.71
pr264	1192.87	1166.69	33.72	5.20	5.84
pr299	888.72	880.14	26.65	5.12	5.17
lin318	816.33	809.68	35.24	5.07	5.49
fl417	2900.69	2878.91	55.41	8.24	8.40

Table 5.29. Performance of chosen variants while solving QAP embedded into SA in terms of solution quality

	Swap	Insertion	E18	E20	E34
Tai25b	13.96	51.86	15.08	22.49	30.50
Chr25a	25.89	103.77	34.72	94.55	82.62
Nug27	1.05	9.31	1.79	8.61	7.26
Tho30	0.95	5.44	1.37	6.28	5.13
Nug30	1.18	6.77	1.37	7.03	6.43
Esc32h	0.30	7.32	0.26	4.69	4.16
Esc32d	0.73	12.43	0.67	10.90	9.63
Tho40	1.37	8.39	1.83	9.72	8.19
Wil50	1.55	5.02	0.45	3.08	2.67
Tai60a	4.96	10.15	3.64	8.61	8.57
Tai60b	7.79	26.22	6.56	19.02	18.14
Tai80b	6.11	24.06	6.52	19.30	16.91
Lipa90b	21.17	25.04	21.32	24.81	24.87
Tai100b	5.73	22.92	4.71	18.22	17.51
Wil100	0.31	2.64	0.44	3.27	3.06

In Table 5.30 and Table 5.31, the run-time results are given. The execution time of the algorithm depends more on the cooling schedule than convergence to the local optimum. Therefore, the mechanisms execute in similar times.

Table 5.30. Average time required (in seconds) for TSP embedded into SA

	Swap	Insertion	E18	E20	E34
krob100	23.5	26.96	48.32	113.05	99.44
lin105	26.59	32.11	50.22	99.64	104.92
bier127	33.98	35.83	55.97	118.48	126.34
ch130	32.39	38.53	60.13	126.12	137.66
kroa150	36.29	43.34	67.44	144.05	151.79
krob150	34.4	39.56	70.48	141.88	151.36
ch150	38.44	46.08	79.02	156.28	159.64
d198	51.21	60.76	96.26	217.23	187.04
kroa200	50.55	61.43	92.56	215.71	186.91
krob200	47.36	55.62	92.35	218.43	184.94
gil262	82.01	99.29	154.57	383.84	296.22
pr264	67.24	80.52	119.04	337.28	251.64
pr299	79.82	90.76	135.46	288.8	283.23
lin318	83.75	96.95	147.3	341.66	304.57
fl417	119.44	138.39	211.7	471.03	422.24
Average	53.80	63.08	98.72	224.90	203.20

Table 5.31. Average time required (in seconds) for QAP embedded into SA

	Swap	Insertion	E18	E20	E34
Tai25b	3.07	3.87	6.62	11.73	10.48
Chr25a	3.06	3.43	5.88	14.14	10.78
Nug27	4.4	4.24	7.47	15.96	12.95
Tho30	4.27	4.5	7.14	16.8	13.98
Nug30	5.29	5.21	8.45	18.66	16.83
Esc32h	7.98	6.61	11.96	22.31	22.78
Esc32d	8.24	7.91	15.77	26.11	31.42
Tho40	7.58	7.26	12.06	23.02	27.44
Wil50	1.51	2	23.22	36.75	38.64
Tai60a	2.13	2.11	28.5	42.06	43.36
Tai60b	3.84	4.46	30.07	43.52	43.57
Tai80b	17.55	15.65	50.6	62.25	71.42
Lipa90b	44.67	46.34	56.04	77.42	94.82
Tai100b	59.05	53.24	83.49	99	109.64
Wil100	79.81	79.12	97.47	115	120.31
Average	16.83	16.40	29.65	41.65	44.56

Table 5.32 and Table 5.33 show the descriptive statistics of percentage deviation from the best-known solutions for TSP and QAP, respectively. Table 5.32 shows that E20 and E34 gives a similar performance. Table 5.33 shows that E18 is still better for QAP.

Table 5.32. Descriptive statistics of the mechanisms in terms of percentage deviation from the best-known solution for TSP embedded into SA

	N	Minimum	Maximum	Mean	Std. Deviation
Design Experiments					
Swap	150	0	156.82	37.93	49.43
Insertion	150	0	153.76	27.30	48.61
E18	150	0	152.95	31.59	49.16
E20	150	0	9.20	2.04	2.06
E34	150	0	6.24	1.46	1.48
Additional Experiments					
Swap	450	8.74	3032.87	545.01	720.51
Insertion	450	0.27	2999.82	532.11	718.15
E18	450	3.9	82.19	22.24	12.78
E20	450	0	16.95	3.57	2.55
E34	450	0	15.65	3.84	2.61

Table 5.33. Descriptive statistics of the mechanisms in terms of percentage deviation from the best-known solution for QAP embedded into SA

	N	Minimum	Maximum	Mean	Std. Deviation
Design Experiments					
Swap	150	0.97	5.06	2.99	0.83
Insertion	150	4.92	12.32	9.08	1.45
E18	150	0	6.17	3.13	0.97
E20	150	3.70	16.02	10.79	2.54
E34	150	2.40	11.74	7.82	1.24
Additional Experiments					
Swap	450	0	48.95	6.2	8.56
Insertion	450	2.15	135.99	21.42	26.31
E18	450	0	49.37	6.71	10.1
E20	450	1.83	128.13	17.37	22.49
E34	450	1.37	104.16	16.38	19.94

Last of all, an experimental study on the design possibilities of CB method is analyzed in this section. In the 6th section, conclusions drawn from the results are presented.

6. CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

6.1. Conclusions

In this thesis, a novel neighbor generation mechanism, called CB mechanism for local search-based algorithms are presented where permutation solution representation is used. The inspiration for CB method comes from the recursive algorithm that creates stochastic cantor set which is expected to generate self-similar neighbors that are different enough in the context of local search. The recursive algorithm gives a rough idea for the design. However, there are decision points to generate neighbors based on the recursive algorithm. At first, the solution representation is treated as a complete string. Then, it is divided into three substrings. What is the length of these substrings? How can these substrings be manipulated to generate neighbors? Furthermore, to generate more neighbors the previously created substrings can be divided according to the rules of the recursive algorithm. However, how many times the diving procedure is done? Therefore, to answer these questions, different variants of CB are designed and tested. In the study, maximum number of stages to further the dividing procedure is three. This number can be increased. However, the experimental study showed that 3 stages are sufficient considering the balance of run-time and performance. However, this can be evaluated when another problem is solved using CB.

CB mechanism, basically, divides a specific section of the solution representation in each stage according to the set rules of the concerned stage and permutes the divided substrings by leaving the undivided section of the solution representation untouched. At last, a neighbor is generated by combining the permuted substrings and the undivided section.

However, the specifics need to be determined as to design an efficient neighbor generation mechanism that execute in a reasonable run-time. Therefore, design possibilities should be explored to know the best way to design CB.

In Section 5, first the experiments to design CB mechanism are presented on two well-known and widely used combinatorial optimization problems which are TSP and QAP. In the experiments, all design issues are analyzed to determine their effects on the performance in order to choose the best design combinations. The comparisons are made

by using both the experimental results of CB mechanism and classical neighbor generation mechanisms (swap and insertion).

Design issues are specified as the number of stages that CB proceeds, dividing procedure of the string and substrings in each stage, reverse ordering of substrings and permutation scheme to generate neighbors (in scheme-1 the neighbors of each stage are generated by permutating appropriate subsequent substrings, in scheme-2 the neighbors of each stage are generated by permutating mid-substrings). In the design of experiments, how to decrease the necessary memory space and how to decrease the run-time, while maximizing the solution quality as much as possible, were the main points to consider.

The experiments are presented in several phases to avoid complication (based on the number of stages that CB mechanism proceeds). Also, the experiments are conducted systematically and successively, meaning preceding experiments affect the succeeding experiments in design choices. At first, all possibilities of the design combinations were experimented on. However, as the experiments progressed, the design combinations that would give bad results are abandoned and a partial set of design combinations were considered.

First, all the design combinations for stage-1 progression were analyzed. It was clear that some design combinations were suitable to TSP because CB variants were already giving a relatively good performance. In detail, variants with reverse ordering and $\text{size}(p_{i2}) \geq 1$ (p_{i2} is the second part in stage- i which is the middle part and $\text{size}(p_{i2}) \geq 1$ stands for the size of p_{i2} is more than 1 element) perform better for TSP as these design choices add variation to the neighborhood to solve a problem that has a steep landscape. However, the experiments for the other combinations were added to stage-2 experiment set in consideration of run-time. No conclusions were made for QAP because the variants performed unsatisfyingly in this set of experiments for solving QAP. For both problems E5 (stage-1, sequential dividing, reverse ordering, scheme-1, $\text{size}(p_{i2}) \geq 1$) performed the best among CB variants.

Secondly, all the design combinations for stage-2 progression were analyzed. The variants that give the best performance for the problems were E20 (stage-2, sequential dividing, reverse ordering(S1-S2), scheme-1, $\text{size}(p_{i2}) \geq 1$) for TSP and E18 (stage-2, sequential dividing, straight ordering, scheme-2, $\text{size}(p_{i2}) = 1$) for QAP. Unlike stage-1

experiments, a variant that gives a better performance for QAP is found this time, which is E18. Since there is a big performance difference between E18 and the other variants, it can safely be said that it has an ideal combination for QAP. However, both variants have been found to be very slow compared to swap.

After stage-2 experiments, it was concluded that scheme-1 and scheme-2 work for solving TSP effectively as long as the variant use $\text{size}(p_{i2}) \geq 1$. For QAP, sequential dividing, scheme-2 with combination $\text{size}(p_{i2}) = 1$ work best.

In stage-2 experiments, the variants started to get too slow. Therefore, a fifth design choice was added to the experiments from this point on which is generating the neighbors from the best neighbor in each stage instead of using the main neighbor that the algorithm holds at the start of the neighbor generation step. In addition, stage-2 experiments were repeated with this additional design choice and the statistical analysis of the results showed that there was no statistically meaningful difference in the performance after incorporating the fifth design choice between the previous and current results with one exception which is E13 (stage-2, specific dividing, reverse ordering(S1), scheme-2, $\text{size}(p_{i2}) \geq 1$). There is not a specific design choice that is detected to explain this difference. Both E13 and E26 (stage-3, specific dividing, reverse ordering(S1), scheme-2, $\text{size}(p_{i2}) \geq 1$) do not show a good performance for neither TSP nor QAP. Therefore, a further analysis of this phenomenon is not done. Hence, it is safe to say that the fifth design choice only speeds the algorithm up as it causes the algorithm to converge faster even though run-time of an iteration is the same as before.

It was obvious after the experiments that the number of neighbors in a neighborhood is determined by the sophistication of each individual experiment. However, if a design combination is not suitable for the problem at hand, the size of the neighborhood does not necessarily mean a better search capability therefore better performance. The run-time is affected by two factors; number of neighbors generated (directly proportional with steps taken in CB mechanism) and efficiency (speed of convergence to a local minimum) of the algorithm.

A curious observation from the stage-3 experiments was that E33 (stage-3, sequential dividing, reverse ordering(S1-S2), scheme-1, $\text{size}(p_{i2}) \geq 1$) and E34 (stage-3, sequential dividing, reverse ordering(S1-S2-S3), scheme-1, $\text{size}(p_{i2}) \geq 1$) solve both TSP and QAP

with a relatively good performance. In terms of neighbor variations, E34 generates all neighbors that E33 can generate and then generates some unique ones as well. Therefore, E34 was included to chosen variants. It is advisable that a researcher should start from E34 to embed into a local search-based algorithm because it is capable to respond to various types of problems.

However, no other variants from stage-3 experiments were chosen. Even though some experiments gave good performances, the variants started to get slower and slower and the performances were not significantly different from the variants chosen from the stage-2 experiments to justify including a slower variant.

After the experiments that are designed to choose the best variants were completed, experiments on additional instances were conducted using the chosen variants to test the robustness and efficiency of CB mechanism. The results were in parallel with the previous results thus proving that the variants are indeed robust and have high efficiency.

In addition, the chosen variants were embedded into SA to use them in randomized way and to test their capabilities in a stochastic algorithm. SA has faster run-time compared to LS and the variants need less design choices to design them as neighbors are generated randomly not systematically.

In the end, it is concluded that it is possible to design a general mechanism that gives consistently good results for both TSP and QAP. It is also possible to solve TSP satisfactorily with some CB variations and a CB variation that outperforms swap when solving QAP.

It must be mentioned that swap or insertion are not the best mechanisms to solve TSP. In the literature, there are mechanisms that are tailored to solve TSP more effectively and efficiently. On the other hand, swap is the best mechanism to solve QAP. However, swap and insertion are not chosen based on their capability to solve TSP or QAP. There are two reasons for choosing swap and insertion. First one is to have baseline results in the experiments to compare CB when solving the test problems. The second one is because they are the most used mechanisms for permutation solution representation in the literature.

6.2. Future Research Directions

CB is an alternative neighbor generation mechanism that shows very promising results. However, it is a relatively slow mechanism. The design of CB mechanism needs to be improved in the future from the point of efficiency.

In addition, TSP and QAP were used as the test problems in the experiments. Even though they are problems that have very different characteristics, they are not enough to generalize the capabilities of CB mechanism to other combinatorial optimization problems. Also, beside the benchmark problems, real-life problems can be used to see performance of CB in practice.

Variants of CB were embedded into LS and SA in the experiments. There is also room to explore as to test how the variants will perform embedded into other local search based algorithms such as tabu search, variable neighborhood search, iterated local search, and so on.

REFERENCES

- Aarts, E. H., & Van Laarhoven, P. J. (1985). Statistical cooling: A general approach to combinatorial optimization problems. *Philips J. Res.*, 40(4), 193-226.
- Afonso, L. D., Mariani, V. C., & dos Santos Coelho, L. (2013). Modified imperialist competitive algorithm based on attraction and repulsion concepts for reliability-redundancy optimization. *Expert Systems with Applications*, 40(9), 3794-3802.
- Agarwal, M., & Gupta, R. (2006). Genetic search for redundancy optimization in complex systems. *Journal of Quality in maintenance Engineering*, 12(4), 338-353.
- Agarwal, M., & Sharma, V. K. (2010). Ant colony approach to constrained redundancy optimization in binary systems. *Applied Mathematical Modelling*, 34(4), 992-1003.
- Aho, A., Hopcroft, J., & Ullman, J. (1975). *The Design and Analysis of Algorithms*. Addison and Wesley. Reading, MA.).
- Alligood, K. T., Sauer, T. D., & Yorke, J. A. (1996). *Chaos* (pp. 105-147). Springer New York.
- Alomoush, M. I. (2020). Optimal Combined Heat and Power Economic Dispatch Using Stochastic Fractal Search Algorithm. *Journal of Modern Power Systems and Clean Energy*, 8(2), 276-286.
- Alomoush, M. I., & Oweis, Z. B. (2018). Environmental-economic dispatch using stochastic fractal search algorithm. *International Transactions on Electrical Energy Systems*, 28(5), e2530.
- Angel, E., & Zissimopoulos, V. (2001). On the landscape ruggedness of the quadratic assignment problem. *Theoretical computer science*, 263(1-2), 159-172.
- Ardakan, M. A., & Hamadani, A. Z. (2014). Reliability–redundancy allocation problem with cold-standby redundancy strategy. *Simulation Modelling Practice and Theory*, 42, 107-118.
- Ashraf, Z., Malhotra, D., Muhuri, P. K., & Lohani, Q. D. (2017, June). Hybrid biogeography-based optimization for solving vendor managed inventory system. In *2017 IEEE Congress on Evolutionary Computation (CEC)* (pp. 2598-2605). IEEE.
- Awad, N. H., Ali, M. Z., Suganthan, P. N., & Jaser, E. (2016). A decremental stochastic fractal differential evolution for global numerical optimization. *Information*

Sciences, 372, 470-491.

Balamurugan, R., & Subramanian, S. (2007). A simplified recursive approach to combined economic emission dispatch. *Electric Power Components and Systems*, 36(1), 17-27.

Baluja, S. (1994). Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning (No. CMU-CS-94-163). Carnegie-Mellon Univ Pittsburgh Pa Dept Of Computer Science.

Basu, M. (2011). Economic environmental dispatch using multi-objective differential evolution. *Applied soft computing*, 11(2), 2845-2853.

Beham, A., Affenzeller, M., & Pitzer, E. (2015). Metaheuristic Algorithms for the Quadratic Assignment Problem: Performance and Comparison. In *Innovative Technologies in Management and Science* (pp. 171-190). Springer, Cham.

Ben-Daya, M., & Al-Fawzan, M. (1998). A tabu search approach for the flow shop scheduling problem. *European journal of operational research*, 109(1), 88-95.

Betka, A., Terki, N., Toumi, A., Hamiane, M., & Ourchani, A. (2019). A new block matching algorithm based on stochastic fractal search. *Applied Intelligence*, 49(3), 1146-1160.

Bhattacharjee, K., Bhattacharya, A., & Dey, S. H. (2014). Solution of economic emission load dispatch problems of power systems by real coded chemical reaction algorithm. *International Journal of Electrical Power & Energy Systems*, 59, 176-187.

Bhattacharya, A., & Chattopadhyay, P. K. (2011). Application of biogeography-based optimisation to solve different optimal power flow problems. *IET generation, transmission & distribution*, 5(1), 70-80.

Brest, J., & Maučec, M. S. (2008). Population size reduction for the differential evolution algorithm. *Applied Intelligence*, 29(3), 228-247.

Burkard, R. E., Karisch, S. E., & Rendl, F. (1997). QAPLIB—A quadratic assignment problem library. *Journal of Global optimization*, 10(4), 391-403.

Cai, J., Li, Q., Li, L., Peng, H., & Yang, Y. (2012). A hybrid FCASO-SQP method for solving the economic dispatch problems with valve-point effects. *Energy*, 38(1), 346-353.

Cárdenas-Barrón, L. E., Treviño-Garza, G., & Wee, H. M. (2012). A simple and better

algorithm to solve the vendor managed inventory control system of multi-product multi-constraint economic order quantity model. *Expert Systems with Applications*, 39(3), 3888-3895.

Chen, T. C. (2006). IAs based approach for reliability redundancy allocation problems. *Applied Mathematics and Computation*, 182(2), 1556-1567.

Chen, Y. P., Peng, W. C., & Jian, M. C. (2007). Particle swarm optimization with recombination and dynamic linkage discovery. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(6), 1460-1470.

Cheng, M. Y., & Prayogo, D. (2014). Symbiotic organisms search: a new metaheuristic optimization algorithm. *Computers & Structures*, 139, 98-112.

Choudhury, H. A., & Saikia, M. (2014, April). Survey on block matching algorithms for motion estimation. In *2014 International Conference on Communication and Signal Processing* (pp. 036-040). IEEE.

Chuan, S. U. N., WEI, Z. Q., ZHOU, C. J., & Bin, W. A. N. G. (2016). Stochastic fractal search algorithm for 3d protein structure prediction. *DEStech Transactions on Computer Science and Engineering*, (aics)

Ciornei, I., & Kyriakides, E. (2013). Recent methodologies and approaches for the economic dispatch of generation in power systems. *International Transactions on Electrical Energy Systems*, 23(7), 1002-1027.

Civicioglu, P. (2013). Backtracking search optimization algorithm for numerical optimization problems. *Applied Mathematics and Computation*, 219(15), 8121-8144.

Coello, C. C., & Lechuga, M. S. (2002, May). MOPSO: A proposal for multiple objective particle swarm optimization. In *Proceedings of the 2002 Congress on Evolutionary Computation*. CEC'02 (Cat. No. 02TH8600) (Vol. 2, pp. 1051-1056). IEEE.

Cohn, H., & Fielding, M. (1999). Simulated annealing: Searching for an optimal temperature schedule. *SIAM Journal on Optimization*, 9(3), 779-802.

Cuevas, E., Echavarría, A., & Ramírez-Ortegón, M. A. (2014). An optimization algorithm inspired by the States of Matter that improves the balance between exploration and exploitation. *Applied intelligence*, 40(2), 256-272.

- Cuevas, E., Zaldivar, D., Pérez-Cisneros, M., & Oliva, D. (2013). Block-matching algorithm based on differential evolution for motion estimation. *Engineering Applications of Artificial Intelligence*, 26(1), 488-498.
- Çelik, E. (2018). Incorporation of stochastic fractal search algorithm into efficient design of PID controller for an automatic voltage regulator system. *Neural Computing and Applications*, 30(6), 1991-2002.
- Çelik, E. (2020). Improved stochastic fractal search algorithm and modified cost function for automatic generation control of interconnected electric power systems. *Engineering Applications of Artificial Intelligence*, 88, 103407.
- Çelik, E., & Gör, H. (2019). Enhanced speed control of a DC servo system using PI+ DF controller tuned by stochastic fractal search technique. *Journal of the Franklin Institute*, 356(3), 1333-1359.
- Çelik, E., & Öztürk, N. (2018). First application of symbiotic organisms search algorithm to off-line optimization of PI parameters for DSP-based DC motor drives. *Neural Computing and Applications*, 30(5), 1689-1699.
- Damerchilu, B., Norouzzadeh, M. S., & Meybodi, M. R. (2016). Motion estimation using learning automata. *Machine Vision and Applications*, 27(7), 1047-1061.
- Deb, K., & Jain, H. (2013). An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. *IEEE transactions on evolutionary computation*, 18(4), 577-601.
- Deb, K., Agrawal, S., Pratap, A., & Meyarivan, T. (2000, September). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *International conference on parallel problem solving from nature* (pp. 849-858). Springer, Berlin, Heidelberg.
- Deep, K. (2009). Dipti. Reliability optimization of complex systems through C-SOMGA. *Journal of Information and Computing Science*, 4(3), 163-72.
- Deroussi, L., Gourgand, M., & Tchernev, N. (2008). A simple metaheuristic approach to the simultaneous scheduling of machines and automated guided vehicles. *International Journal of Production Research*, 46(8), 2143-2164.

- Díaz-Cortés, M. A., Cuevas, E., & Rojas, R. (2017). Motion estimation algorithm using block-matching and harmony search optimization. In *Engineering Applications of Soft Computing* (pp. 13-44). Springer, Cham.
- Dorigo, M., & Di Caro, G. (1999, July). Ant colony optimization: a new meta-heuristic. In *Proceedings of the 1999 congress on evolutionary computation-CEC99* (Cat. No. 99TH8406) (Vol. 2, pp. 1470-1477). IEEE.
- dos Santos Coelho, L. (2009). An efficient particle swarm approach for mixed-integer programming in reliability–redundancy optimization applications. *Reliability Engineering & System Safety*, 94(4), 830-837.
- Dubey, H. M., Pandit, M., & Panigrahi, B. K. (2018). An overview and comparative analysis of recent bio-inspired optimization techniques for wind integrated multi-objective power dispatch. *Swarm and Evolutionary Computation*, 38, 12-34.
- Duman, S., Güvenç, U., Sönmez, Y., & Yörükeren, N. (2012). Optimal power flow using gravitational search algorithm. *Energy Conversion and Management*, 59, 86-95.
- Edelkamp, S., & Schroedl, S. (2011). *Heuristic search: theory and applications*. Elsevier.
- Edgar, G. A. (2013). *Integral, probability, and fractal measures*. Springer Science & Business Media.
- El-Fergany, A. A., & Hasanien, H. M. (2017). Optimized settings of directional overcurrent relays in meshed power networks using stochastic fractal search algorithm. *International Transactions on Electrical Energy Systems*, 27(11), e2395.
- Elsayed, S. M., Sarker, R. A., Essam, D. L., & Hamza, N. M. (2014, July). Testing united multi-operator evolutionary algorithms on the CEC2014 real-parameter numerical optimization. In *2014 IEEE Congress on Evolutionary Computation (CEC)* (pp. 1650-1657). IEEE.
- Eskandar, H., Sadollah, A., Bahreininejad, A., & Hamdi, M. (2012). Water cycle algorithm—A novel metaheuristic optimization method for solving constrained engineering optimization problems. *Computers & Structures*, 110, 151-166.
- Falconer, K. (2004). *Fractal geometry: mathematical foundations and applications*. John Wiley & Sons.

- Fielding, M. (2000). Simulated annealing with an optimal fixed temperature. *SIAM Journal on Optimization*, 11(2), 289-307.
- Fogel, L. J., Owens, A. J., & Walsh, M. J. (1966). Intelligent decision making through a simulation of evolution. *Behavioral science*, 11(4), 253-272.
- Gandomi, A. H., & Alavi, A. H. (2012). Krill herd: a new bio-inspired optimization algorithm. *Communications in nonlinear science and numerical simulation*, 17(12), 4831-4845.
- Gandomi, A. H., Yang, X. S., & Alavi, A. H. (2013). Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. *Engineering with computers*, 29(1), 17-35.
- Ganguly, S., & Samajpati, D. (2017). Distributed generation allocation with on-load tap changer on radial distribution networks using adaptive genetic algorithm. *Applied Soft Computing*, 59, 45-67.
- Geem, Z. W., Kim, J. H., & Loganathan, G. V. (2001). A new heuristic optimization algorithm: harmony search. *simulation*, 76(2), 60-68.
- Geman, S., & Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6), 721-741.
- Ghasemi, M., Aghaei, J., Akbari, E., Ghavidel, S., & Li, L. (2016). A differential evolution particle swarm optimizer for various types of multi-area economic dispatch problems. *Energy*, 107, 182-195.
- Glover, F. (1977). Heuristics for integer programming using surrogate constraints. *Decision sciences*, 8(1), 156-166.
- Glover, F. (1991). Multilevel tabu search and embedded search neighborhoods for the traveling salesman problem. Graduate School of Business, University of Colorado.
- Glover, F. (1996). Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, 65(1-3), 223-253.
- Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*.

- Güvenç, U. (2010). Combined economic emission dispatch solution using genetic algorithm based on similarity crossover. *Scientific Research and Essays*, 5(17), 2451-2456.
- Güvenç, U., Sönmez, Y., Duman, S., & Yörükeren, N. (2012). Combined economic and emission dispatch solution using gravitational search algorithm. *Scientia Iranica*, 19(6), 1754-1762.
- Haghray, A., Nazari-Heris, M., & Mohammadi-Ivatloo, B. (2016). Solving combined heat and power economic dispatch problem using real coded genetic algorithm with improved Mühlhenbein mutation. *Applied Thermal Engineering*, 99, 465-475.
- Hajek, B. (1988). Cooling schedules for optimal annealing. *Mathematics of operations research*, 13(2), 311-329
- He, Q., Hu, X., Ren, H., & Zhang, H. (2015). A novel artificial fish swarm algorithm for solving large-scale reliability–redundancy application problem. *ISA transactions*, 59, 105-113.
- Herbadji, O., Nadhir, K., Slimani, L., & Bouktir, T. (2013b, April). Optimal power flow with emission controlled using firefly algorithm. In *2013 5th International Conference on Modeling, Simulation and Applied Optimization (Icmsao)* (pp. 1-6). IEEE.
- Herbadji, O., Slimani, L., & Bouktir, T. (2013a). Biogeography based optimization approach for solving optimal power flow problem. *International Journal of Hybrid Information Technology*, 6(5), 183-196.
- Holland, J. H. (1975). *Adaptation in natural and artificial systems* Ann Arbor. The University of Michigan Press, 1, 975.
- Holland, J. H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.
- Hsieh, Y. C., & You, P. S. (2011). An effective immune based two-phase approach for the optimal reliability–redundancy allocation problem. *Applied Mathematics and Computation*, 218(4), 1297-1307.
- Hsieh, Y. C., Chen, T. C., & Bricker, D. L. (1998). Genetic algorithms for reliability design problems. *Microelectronics Reliability*, 38(10), 1599-1605.

Huang, C. L. (2015). A particle-based simplified swarm optimization algorithm for reliability redundancy allocation problems. *Reliability Engineering & System Safety*, 142, 221-230.

Hung, D. Q., & Mithulanathan, N. (2011). Multiple distributed generator placement in primary distribution networks for loss reduction. *IEEE Transactions on industrial electronics*, 60(4), 1700-1708.

Injeti, S. K., & Kumar, N. P. (2013). A novel approach to identify optimal access point and capacity of multiple DGs in a small, medium and large scale radial distribution systems. *International Journal of Electrical Power & Energy Systems*, 45(1), 142-151.

Jamian, J. J., Mustafa, M. W., & Mokhlis, H. (2015). Optimal multiple distributed generation output through rank evolutionary particle swarm optimization. *Neurocomputing*, 152, 190-198.

Jeyakumar, D. N., Venkatesh, P., & Lee, K. Y. (2007, August). Application of multi objective evolutionary programming to combined economic emission dispatch problem. In *2007 International Joint Conference on Neural Networks* (pp. 1162-1167). IEEE.

Kanagaraj, G., Ponnambalam, S. G., & Jawahar, N. (2013). A hybrid cuckoo search and genetic algorithm for reliability–redundancy allocation problems. *Computers & Industrial Engineering*, 66(4), 1115-1124.

Kansal, S., Kumar, V., & Tyagi, B. (2016). Hybrid approach for optimal placement of multiple DGs of multiple types in distribution networks. *International Journal of Electrical Power & Energy Systems*, 75, 226-235.

Karaboga, D. (2005). An idea based on honey bee swarm for numerical optimization (Vol. 200, pp. 1-10). Technical report-tr06, Erciyes university, engineering faculty, computer engineering department.

Kaur, S., Kumbhar, G., & Sharma, J. (2014). A MINLP technique for optimal placement of multiple DG units in distribution systems. *International Journal of Electrical Power & Energy Systems*, 63, 609-617.

Kennedy, J., & Eberhart, R. (1995, November). Particle swarm optimization (PSO). In *Proc. IEEE International Conference on Neural Networks*, Perth, Australia (pp. 1942-1948).

Khalilpourazari, S., & Khalilpourazary, S. (2018). A Robust Stochastic Fractal Search approach for optimization of the surface grinding process. *Swarm and Evolutionary Computation*, 38, 173-186.

Khalilpourazari, S., Naderi, B., & Khalilpourazary, S. (2020). Multi-Objective Stochastic Fractal Search: a powerful algorithm for solving complex multi-objective optimization problems. *Soft Computing*, 24(4), 3037-3066.

Khalilpourazari, S., Pasandideh, S. H. R., & Niaki, S. T. A. (2016). Optimization of multi-product economic production quantity model with partial backordering and physical constraints: SQP, SFS, SA, and WCA. *Applied Soft Computing*, 49, 770-791.

Khanam, I., & Parmar, G. (2017a, July). Application of stochastic fractal search in order reduction of large scale LTI systems. In *2017 International Conference on Computer, Communications and Electronics (Comptelix)* (pp. 190-194). IEEE.

Khanam, I., & Parmar, G. (2017b). Control of DC Motor by SFS/PID Approach and A Comparative Study. *International Journal of Engineering Technology, Management and Applied Sciences (IJETMAS)*, 5(5), 8-14.

Khishe, M., Mosavi, M. R., & Moridi, A. (2018). Chaotic fractal walk trainer for sonar data set classification using multi-layer perceptron neural network and its hardware implementation. *Applied Acoustics*, 137, 121-139.

Khoa, T. H., Nallagownden, P., Baharudin, Z., & Dieu, V. N. (2017, November). One rank cuckoo search algorithm for optimal placement of multiple distributed generators in distribution networks. In *TENCON 2017-2017 IEEE Region 10 Conference* (pp. 1715-1720). IEEE.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, 220(4598), 671-680.

Koga, T. (1981). Motion compensated interframe coding for video-conferencing. In *Proc. Nat. Telecommun. Conf.* (pp. G5-3).

Kuo, C. C. (2008). A novel coding scheme for practical economic dispatch by modified particle swarm approach. *IEEE Transactions on Power Systems*, 23(4), 1825-1835.

Lai, L. L., Ma, J. T., Yokoyama, R., & Zhao, M. (1997). Improved genetic algorithms for

optimal power flow under both normal and contingent operation states. *International Journal of Electrical Power & Energy Systems*, 19(5), 287-292.

Lee, K. M., Hsu, M. R., Chou, J. H., & Guo, C. Y. (2011). Improved differential evolution approach for optimization of surface grinding process. *Expert Systems with Applications*, 38(5), 5680-5686.

Lee, K. Y., Park, Y. M., & Ortiz, J. L. (1985). A united approach to optimal real and reactive power dispatch. *IEEE Transactions on power Apparatus and systems*, (5), 1147-1153.

Li, B. (2016). Atomic potential matching: an evolutionary target recognition approach based on edge features. *Optik-International Journal for Light and Electron Optics*, 127(5), 3162-3168.

Li, R., Zeng, B., & Liou, M. L. (1994). A new three-step search algorithm for block motion estimation. *IEEE transactions on circuits and systems for video technology*, 4(4), 438-442.

Lin, J., & Wang, Z. J. (2017). Parameter identification for fractional-order chaotic systems using a hybrid stochastic fractal search algorithm. *Nonlinear Dynamics*, 90(2), 1243-1255.

Lin, J., & Wang, Z. J. (2019). Multi-area economic dispatch using an improved stochastic fractal search algorithm. *Energy*, 166, 47-58

Lin, S., & Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2), 498-516.

Lin, W. M., Cheng, F. S., & Tsay, M. T. (2002). An improved tabu search for economic dispatch with multiple minima. *IEEE Transactions on power systems*, 17(1), 108-112.

Lin, X., & Li, H. (2008, December). Enhanced Pareto particle swarm approach for multi-objective optimization of surface grinding process. In *2008 Second International Symposium on Intelligent Information Technology Application* (Vol. 2, pp. 618-623). IEEE.

Liu, Y., & Qin, G. (2014). A hybrid TS-DE algorithm for reliability redundancy optimization problem. *Journal of Computers*, 9(9), 2050-7.

- Liu, Y., & Qin, G. (2014). A modified particle swarm optimization algorithm for reliability redundancy optimization problem. *Journal of Computers*, 9(9), 2124-2131.
- Liu, Y., & Qin, G. (2014). Reliability Redundancy Optimization Algorithm based on Eagle Strategy and PSO. *International Journal of Multimedia and Ubiquitous Engineering*, 9(12), 375-382.
- Liu, Y., & Qin, G. (2015). A DE algorithm combined with Lévy flight for reliability redundancy allocation problems. *International Journal of Hybrid Information Technology*, 8(5), 113-118.
- Lu, J., & Liou, M. L. (1997). A simple and efficient search algorithm for block-matching motion estimation. *IEEE Transactions on Circuits and Systems for Video Technology*, 7(2), 429-433.
- Lynn, N., & Suganthan, P. N. (2015). Heterogeneous comprehensive learning particle swarm optimization with enhanced exploration and exploitation. *Swarm and Evolutionary Computation*, 24, 11-24.
- Mahdad, B. (2019). Optimal reconfiguration and reactive power planning based fractal search algorithm: A case study of the Algerian distribution electrical system. *Engineering Science and Technology, an International Journal*, 22(1), 78-101.
- Maheri, M. R., & Narimani, M. M. (2014). An enhanced harmony search algorithm for optimum design of side sway steel frames. *Computers & Structures*, 136, 78-89.
- Mahmoud, K., Yorino, N., & Ahmed, A. (2015). Optimal distributed generation allocation in distribution systems for loss minimization. *IEEE Transactions on Power Systems*, 31(2), 960-969.
- Mallipeddi, R., Suganthan, P. N., Pan, Q. K., & Tasgetiren, M. F. (2011). Differential evolution algorithm with ensemble of parameters and mutation strategies. *Applied soft computing*, 11(2), 1679-1696.
- Mandal, B., & Roy, P. K. (2013). Optimal reactive power dispatch using quasi-oppositional teaching learning based optimization. *International Journal of Electrical Power & Energy Systems*, 53, 123-134.
- Mandelbrot, B. B. (1983). *The fractal geometry of nature* (Vol. 173, p. 51). New York:

WH freeman.

Mehrabian, A. R., & Lucas, C. (2006). A novel numerical optimization algorithm inspired from weed colonization. *Ecological informatics*, 1(4), 355-366.

Mellal, M. A., & Zio, E. (2016). A penalty guided stochastic fractal search approach for system reliability optimization. *Reliability Engineering & System Safety*, 152, 213-227.

Michalewicz, Z. (2013). *Genetic algorithms+ data structures= evolution programs*. Springer Science & Business Media.

Mills, P., Tsang, E., & Ford, J. (2003). Applying an extended guided local search to the quadratic assignment problem. *Annals of Operations Research*, 118(1-4), 121-135.

Mirjalili, S. (2015a). The ant lion optimizer. *Advances in Engineering Software*, 83, 80-98.

Mirjalili, S. (2015b). Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm. *Knowledge-Based Systems*, 89, 228-249.

Mirjalili, S., Mirjalili, S. M., & Lewis, A. (2014). Grey wolf optimizer. *Advances in engineering software*, 69, 46-61.

Mirjalili, S., Saremi, S., Mirjalili, S. M., & Coelho, L. D. S. (2016). Multi-objective grey wolf optimizer: a novel algorithm for multi-criterion optimization. *Expert Systems with Applications*, 47, 106-119.

Misevičius, A. (2000). An intensive search algorithm for the quadratic assignment problem. *Informatica*, 11(2), 145-162.

Montgomery, D. C., & Runger, G. C. (2010). *Applied statistics and probability for engineers*. John Wiley & Sons.

Moradi, M. H., & Abedini, M. (2012). A combination of genetic algorithm and particle swarm optimization for optimal DG location and sizing in distribution systems. *International Journal of Electrical Power & Energy Systems*, 34(1), 66-74.

Moradi, M. H., & Abedini, M. (2016). A novel method for optimal DG units capacity and location in Microgrids. *International Journal of Electrical Power & Energy Systems*, 75, 236-244.

Moradi, M. H., Zeinalzadeh, A., Mohammadi, Y., & Abedini, M. (2014). An efficient

hybrid method for solving the optimal sitting and sizing problem of DG and shunt capacitor banks simultaneously based on imperialist competitive algorithm and genetic algorithm. *International Journal of Electrical Power & Energy Systems*, 54, 101-111.

Mosavi, M. R., Khishe, M., Hatam Khani, Y., & Shabani, M. (2017). Training radial basis function neural network using stochastic fractal search algorithm to classify sonar dataset. *Iran J Electr Electron Eng*, 13(1), 100-111.

Mosbah, H., & El-Hawary, M. (2016, May). Power system tracking state estimation based on stochastic fractal search technique under sudden load changing conditions. In 2016 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE) (pp. 1-6). IEEE.

Mosbah, H., & El-Hawary, M. E. (2017). Optimization of neural network parameters by Stochastic Fractal Search for dynamic state estimation under communication failure. *Electric Power Systems Research*, 147, 288-301.

Murty, B. S. N., & Reddy, P. J. (1999). Reliability optimization of complex systems using modified random-to-pattern search (MRPS) algorithm. *Quality and reliability engineering international*, 15(3), 239-243.

Muthukumar, K., & Jayalalitha, S. (2016). Optimal placement and sizing of distributed generators and shunt capacitors for power loss minimization in radial distribution networks using hybrid heuristic search optimization technique. *International Journal of Electrical Power & Energy Systems*, 78, 299-319.

Nearchou, A. C. (2004). A novel metaheuristic approach for the flow shop scheduling problem. *Engineering Applications of Artificial Intelligence*, 17(3), 289-300.

Nekooei, K., Farsangi, M. M., Nezamabadi-Pour, H., & Lee, K. Y. (2013). An improved multi-objective harmony search for optimal placement of DGs in distribution systems. *IEEE Transactions on smart grid*, 4(1), 557-567.

Nguyen, T. P., & Vo, D. N. (2018). A novel stochastic fractal search algorithm for optimal allocation of distributed generators in radial distribution systems. *Applied Soft Computing*, 70, 773-796.

Nguyen, T. P., Tran T. T., & Vo, D. N. (2018). Improved stochastic fractal search algorithm with chaos for optimal determination of location, size, and quantity of

distributed generators in distribution systems. *Neural Computing and Applications*, 1-26.

Nguyen, T. T., Vo, D. N., & Dinh, B. H. (2016). Cuckoo search algorithm for combined heat and power economic dispatch. *International Journal of Electrical Power & Energy Systems*, 81, 204-214.

Nguyen, T. T., Vo, D. N., Van Tran, H., & Van Dai, L. (2019). Optimal Dispatch of Reactive Power Using Modified Stochastic Fractal Search Algorithm. *Complexity*, 2019.

Niknam, T., Narimani, M. R., Aghaei, J., & Azizipanah-Abarghooee, R. (2012). Improved particle swarm optimisation for multi-objective optimal power flow considering the cost, loss, emission and voltage stability index. *IET generation, transmission & distribution*, 6(6), 515-527.

Nourani, Y., & Andresen, B. (1998). A comparison of simulated annealing cooling strategies. *Journal of Physics A: Mathematical and General*, 31(41), 8373.

Ouyang, H. B., Gao, L. Q., Li, S., & Kong, X. Y. (2015). Improved novel global harmony search with a new relaxation method for reliability optimization problems. *Information Sciences*, 305, 14-55.

Padhy, S., & Panda, S. (2017). A hybrid stochastic fractal search and pattern search technique based cascade PI-PD controller for automatic generation control of multi-source power systems in presence of plug in electric vehicles. *CAAI Transactions on Intelligence Technology*, 2(1), 12-25.

Park, J. B., Lee, K. S., Shin, J. R., & Lee, K. Y. (2005). A particle swarm optimization for economic dispatch with nonsmooth cost functions. *IEEE Transactions on Power systems*, 20(1), 34-42.

Passino, K. M. (2002). Biomimicry of bacterial foraging for distributed optimization and control. *IEEE control systems magazine*, 22(3), 52-67.

Pattnaik, S. S., Bakwad, K. M., Sohi, B. S., Ratho, R. K., & Devi, S. (2013). Swine influenza models based optimization (SIMBO). *Applied Soft Computing*, 13(1), 628-653.

Pedersen, M. E. H. (2010). Tuning & simplifying heuristical optimization (Doctoral dissertation, University of Southampton).

Pedersen, M. E. H., & Chipperfield, A. J. (2010). Simplifying particle swarm

optimization. *Applied Soft Computing*, 10(2), 618-628.

Pham, L. H., Duong, M. Q., Phan, V. D., Nguyen, T. T., & Nguyen, H. N. (2019b). A High-Performance Stochastic Fractal Search Algorithm for Optimal Generation Dispatch Problem. *Energies*, 12(9), 1796.

Pham, L. H., Nguyen, T. T., Pham, L. D., & Nguyen, N. H. (2019a). Stochastic fractal search based method for economic load dispatch. *TELKOMNIKA*, 17(5), 2535-2546.

Po, L. M., & Ma, W. C. (1996). A novel four-step search algorithm for fast block motion estimation. *IEEE transactions on circuits and systems for video technology*, 6(3), 313-317.

Qin, A. K., Huang, V. L., & Suganthan, P. N. (2008). Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE transactions on Evolutionary Computation*, 13(2), 398-417.

Radosavljević, J., & Jevtić, M. (2016). Hybrid GSA-SQP algorithm for optimal coordination of directional overcurrent relays. *IET Generation, Transmission & Distribution*, 10(8), 1928-1937.

Rahman, T. A., & Tokhi, M. O. (2016, August). Enhanced stochastic fractal search algorithm with chaos. In 2016 7th IEEE Control and System Graduate Research Colloquium (ICSGRC)(pp. 22-27). IEEE.

Rahman, T. A., As' Arry, A., Jalil, N. A. A., & Ahmad, R. M. K. R. (2017a, April). Chaotic fractal search algorithm for global optimization with application to control design. In 2017 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE) (pp. 111-116). IEEE.

Rahman, T. A., As'array, A., & Jalil, N. A. A. (2017b). Active vibration control of a flexible beam structure using chaotic fractal search algorithm. *Procedia engineering*, 170, 299-306.

Rashedi, E., Nezamabadi-Pour, H., & Saryazdi, S. (2009). GSA: a gravitational search algorithm. *Information sciences*, 179(13), 2232-2248.

Ravi, V., Murty, B. S. N., & Reddy, J. (1997). Nonequilibrium simulated-annealing algorithm applied to reliability optimization of complex systems. *IEEE Transactions on*

Reliability, 46(2), 233-239.

Ravi, V., Reddy, P. J., & Zimmermann, H. J. (2000). Fuzzy global optimization of complex system reliability. *IEEE Transactions on Fuzzy systems*, 8(3), 241-248.

Razmjooy, N., Khalilpour, M., & Ramezani, M. (2016). A new meta-heuristic optimization algorithm inspired by FIFA world cup competitions: theory and its application in PID designing for AVR system. *Journal of Control, Automation and Electrical Systems*, 27(4), 419-440.

Rechenberg, I. (1973). *Evolutionsstrategie—Optimierung technischer Systeme nach Prinzipien der biologischen Information*. Stuttgart-Bad Cannstatt: Friedrich Frommann Verlag.

Reddy, A. S., & Vaisakh, K. (2013). Shuffled differential evolution for large scale economic dispatch. *Electric Power Systems Research*, 96, 237-245.

Rego, C., Gamboa, D., Glover, F., & Osterman, C. (2011). Traveling salesman problem heuristics: Leading methods, implementations and latest advances. *European Journal of Operational Research*, 211(3), 427-441.

Reinelt, G. (1991). TSPLIB—A traveling salesman problem library. *ORSA journal on computing*, 3(4), 376-384.

Sadollah, A., Bahreininejad, A., Eskandar, H., & Hamdi, M. (2012). Mine blast algorithm for optimization of truss structures with discrete variables. *Computers & Structures*, 102, 49-63.

Saha, S., & Mukherjee, V. (2016). Optimal placement and sizing of DGs in RDS using chaos embedded SOS algorithm. *IET Generation, Transmission & Distribution*, 10(14), 3671-3680.

Sayah, S., & Zehar, K. (2008). Modified differential evolution algorithm for optimal power flow with non-smooth cost functions. *Energy conversion and Management*, 49(11), 3036-3042.

Secui, D. C. (2015). The chaotic global best artificial bee colony algorithm for the multi-area economic/emission dispatch. *Energy*, 93, 2518-2545.

Shah-Hosseini, H. (2009). The intelligent water drops algorithm: a nature-inspired

swarm-based optimization algorithm. *International Journal of Bio-inspired computation*, 1(1-2), 71-79.

Shareef, H., Ibrahim, A. A., & Mutlag, A. H. (2015). Lightning search algorithm. *Applied Soft Computing*, 36, 315-333.

Sharma, S., Bhattacharjee, S., & Bhattacharya, A. (2016). Quasi-Oppositional Swine Influenza Model Based Optimization with Quarantine for optimal allocation of DG in radial distribution network. *International Journal of Electrical Power & Energy Systems*, 74, 348-373.

Sheikhalishahi, M., Ebrahimipour, V., Shiri, H., Zaman, H., & Jeihoonian, M. (2013). A hybrid GA–PSO approach for reliability optimization in redundancy allocation problem. *The International Journal of Advanced Manufacturing Technology*, 68(1-4), 317-338.

Simon, D. (2008). Biogeography-based optimization. *IEEE transactions on evolutionary computation*, 12(6), 702-713.

Sivalingam, R., Chinnamuthu, S., & Dash, S. S. (2017). A hybrid stochastic fractal search and local unimodal sampling based multistage PDF plus (1+ PI) controller for automatic generation control of power systems. *Journal of the Franklin Institute*, 354(12), 4762-4783.

Sivanagaraju, S., Suresh, C. V., Srikumar, K., & Babu, A. N. (2014, February). Multi-objective optimization using NDSPSO with cost, emission and loss objectives. In *Proc. International conference on Power Systems, Energy and Environment (PSEE)*, Interlaken, Switzerland (pp. 57-62).

Slowik, A., & Slowik, J. (2008). Multi-objective optimization of surface grinding process with the use of evolutionary algorithm with remembered Pareto set. *The International Journal of Advanced Manufacturing Technology*, 37(7-8), 657-669.

Sohal, M., Singh, A., & Virk, R. S. (2015). A framework for optimizing distributed database queries based on stochastic fractal search. *Int. J. Comp. Sc. and Mob. Computing (JCSMC)*, 4(6), 544-551.

Sorkin, G. B. (1991). Efficient simulated annealing on fractal energy landscapes. *Algorithmica*, 6(1-6), 367.

- Spellucci, P. (1998). A new technique for inconsistent QP problems in the SQP method. *Mathematical Methods of Operations Research*, 47(3), 355-400.
- Stützle, T. (2006). Iterated local search for the quadratic assignment problem. *European Journal of Operational Research*, 174(3), 1519-1539.
- Subbaraj, P., Rengaraj, R., & Salivahanan, S. (2009). Enhancement of combined heat and power economic dispatch using self adaptive real-coded genetic algorithm. *Applied energy*, 86(6), 915-921.
- Sultana, S., & Roy, P. K. (2014). Multi-objective quasi-oppositional teaching learning based optimization for optimal location of distributed generator in radial distribution systems. *International Journal of Electrical Power & Energy Systems*, 63, 534-545.
- Taillard, É. (1991). Robust taboo search for the quadratic assignment problem. *Parallel computing*, 17(4-5), 443-455.
- Talbi, E. G. (2009). *Metaheuristics: from design to implementation* (Vol. 74). John Wiley & Sons.
- Tanabe, R., & Fukunaga, A. (2013, June). Success-history based parameter adaptation for differential evolution. In *2013 IEEE congress on evolutionary computation* (pp. 71-78). IEEE.
- Tanabe, R., & Fukunaga, A. S. (2014, July). Improving the search performance of SHADE using linear population size reduction. In *2014 IEEE congress on evolutionary computation (CEC)* (pp. 1658-1665). IEEE.
- Tejani, G. G., Bhensdadia, V. H., & Bureerat, S. (2016). Examination of three meta-heuristic algorithms for optimal design of planar steel frames. *Advances in Computational Design*, 1(1), 79-86
- Tian, P., Ma, J., & Zhang, D. M. (1999). Application of the simulated annealing algorithm to the combinatorial optimisation problem with permutation property: An investigation of generation mechanism. *European Journal of Operational Research*, 118(1), 81-94.
- Tizhoosh, H. R. (2005, November). Opposition-based learning: a new scheme for machine intelligence. In *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents*,

Web Technologies and Internet Commerce (CIMCA-IAWTIC'06) (Vol. 1, pp. 695-701). IEEE.

Toğan, V. (2012). Design of planar steel frames using teaching–learning based optimization. *Engineering Structures*, 34, 225-232.

Tran, C. D., Nguyen, T. T., Hoang, H. M., & Nguyen, B. Q. (2016). One rank cuckoo search algorithm for bi-objective load dispatch problem. *International Journal of Grid and Distributed Computing*, 9(4), 13-26.

Triki, E., Collette, Y., & Siarry, P. (2005). A theoretical study on the behavior of simulated annealing leading to a new cooling schedule. *European Journal of Operational Research*, 166(1), 77-92.

Tyagi, T., Dubey, H. M., & Pandit, M. (2016). Multi-objective optimal dispatch solution of solar-wind-thermal system using improved stochastic fractal search algorithm. *International Journal of Information Technology and Computer Science (IJITCS)*, 8(11), 61-73.

Vaisakh, K., & Srinivas, L. R. (2011a). Evolving ant direction differential evolution for OPF with non-smooth cost functions. *Engineering Applications of Artificial Intelligence*, 24(3), 426-436.

Vaisakh, K., & Srinivas, L. R. (2011b). Genetic evolving ant direction HDE for OPF with non-smooth cost functions and statistical analysis. *Expert Systems with Applications*, 38(3), 2046-2062.

Valian, E., Tavakoli, S., Mohanna, S., & Haghi, A. (2013). Improved cuckoo search for reliability optimization problems. *Computers & Industrial Engineering*, 64(1), 459-468.

Van Tran, H., Van Pham, T., Pham, L. H., Le, N. T., & Nguyen, T. T. (2019). Finding optimal reactive power dispatch solutions by using a novel improved stochastic fractal search optimization algorithm. *TELKOMNIKA*, 17(5), 2517-2526.

Voss R. F., (1988), “Fractals in nature: From characterization to simulation”, Edited by Peitgen, H. O., & Saupe, D. (1988). *The Science of Fractal Images*, pg.21

Vrugt, J. A., Robinson, B. A., & Hyman, J. M. (2008). Self-adaptive multimethod search for global optimization in real-parameter spaces. *IEEE Transactions on Evolutionary*

Computation, 13(2), 243-259.

Wang, Y., Cai, Z., & Zhang, Q. (2011). Differential evolution with composite trial vector generation strategies and control parameters. *IEEE Transactions on Evolutionary Computation*, 15(1), 55-66.

Wang, Y., Cai, Z., & Zhang, Q. (2012). Enhancing the search ability of differential evolution through orthogonal crossover. *Information Sciences*, 185(1), 153-177.

Wang, Y., Li, H. X., Huang, T., & Li, L. (2014). Differential evolution based on covariance matrix learning and bimodal distribution parameter setting. *Applied Soft Computing*, 18, 232-247.

Wen, X. M., Tay, A. A. O., & Nee, A. Y. C. (1992). Micro-computer-based optimization of the surface grinding process. *Journal of Materials Processing Technology*, 29(1-3), 75-90.

Wu, G., Mallipeddi, R., Suganthan, P. N., Wang, R., & Chen, H. (2016). Differential evolution with multi-population based ensemble of mutation strategies. *Information Sciences*, 329, 329-345.

Wu, P., Gao, L., Zou, D., & Li, S. (2011). An improved particle swarm optimization algorithm for reliability problems. *ISA transactions*, 50(1), 71-81.

Xu, Z., Kuo, W., & Lin, H. H. (1990). Optimization limits in improving system reliability. *IEEE Transactions on reliability*, 39(1), 51-60.

Yang, X. S. (2012, September). Flower pollination algorithm for global optimization. In *International conference on unconventional computing and natural computation* (pp. 240-249). Springer, Berlin, Heidelberg.

Yang, X. S., & Deb, S. (2009, December). Cuckoo search via Lévy flights. In *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)* (pp. 210-214). IEEE.

Yang, Xin-She, and *Nature-Inspired Metaheuristic Algorithms*. "Luniver press." Beckington, UK (2008): 242-246.

Yeh, W. C. (2009). A two-stage discrete particle swarm optimization for the problem of multiple multi-level redundancy allocation in series systems. *Expert Systems with Applications*, 36(5), 9192-9200.

- Yuan, X., & Shen, X. (2008, July). Block matching algorithm based on particle swarm optimization for motion estimation. In 2008 International Conference on Embedded Software and Systems (pp. 191-195). IEEE.
- Zhang, D., Liu, Y., M'Hallah, R., & Leung, S. C. (2010). A simulated annealing with a new neighborhood structure based algorithm for high school timetabling problems. *European Journal of Operational Research*, 203(3), 550-558.
- Zhang, G., Liu, M., Li, J., Ming, W., Shao, X., & Huang, Y. (2014). Multi-objective optimization for surface grinding process using a hybrid particle swarm optimization algorithm. *The International Journal of Advanced Manufacturing Technology*, 71(9-12), 1861-1872.
- Zhang, H., Hu, X., Shao, X., Li, Z., & Wang, Y. (2013). IPSO-based hybrid approaches for reliability-redundancy allocation problems. *Science China Technological Sciences*, 56(11), 2854-2864.
- Zhang, J., & Sanderson, A. C. (2009). JADE: adaptive differential evolution with optional external archive. *IEEE Transactions on evolutionary computation*, 13(5), 945-958.
- Zhang, Q., & Li, H. (2007). MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, 11(6), 712-731.
- Zhou, C., Sun, C., Wang, B., & Wang, X. (2018). An improved stochastic fractal search algorithm for 3D protein structure prediction. *Journal of molecular modeling*, 24(6), 125.
- Zhu, S., & Ma, K. K. (2000). A new diamond search algorithm for fast block-matching motion estimation. *IEEE transactions on Image Processing*, 9(2), 287-290.
- Zou, D., Liu, H., Gao, L., & Li, S. (2011). A novel modified differential evolution algorithm for constrained optimization problems. *Computers & Mathematics with Applications*, 61(6), 1608-1623.

CURRICULUM VITAE

Melike ÖZTÜRK has graduated from Sakarya University, Department of Industrial Engineering in 2011 and received her MSc. from the same department in 2013. She worked as a manufacturing engineer and a manufacturing manager between 2013-2015. She began working as a research assistant at Doğuř University in 2017. She studies heuristic algorithms and optimization.

