

University of Oran 2

Institute of Maintenance and Industrial Security

MASTER'S THESIS

To Obtain a master's degree in

Industrial Engineering

Artificial Intelligence Technique for the Under Subset Coverage Problems (USCP)

Presented and published:

Tajudeen Abdullahi Adigun

The defense jury will be:

Mr. Rouan-Serik M	MCB - Université d'Oran 2	Supervisor
Mme. Boulekhras A	MCB - Université d'Oran 2	President
Mr. Moulai-khatir A	MCB - Université d'Oran 2	Examinator

Academic Year 2023/2024

Abstract

The thesis investigates the Under Set Cover Problem (USCP), a complex combinatorial optimization problem, and aims to develop efficient algorithms using AI techniques. The research uses Greedy, Random search, and Genetic algorithm techniques, and evaluates their performance through computational time experiments. The findings show potential for improving USCP efficiency and addressing complex optimization problems, contributing to theoretical advancements and practical applications.

keywords: Under Set Cover Problem (USCP), Combinatorial Optimization, Computational Time, Problem-Solving, Heuristic Methods, Algorithmic Performance, Artificial Intelligence (AI).

Résumé

Cette thèse examine le Problème de Couverture de Sous-Ensembles (PCSE), un problème complexe d'optimisation combinatoire, et vise à développer des algorithmes efficaces utilisant des techniques d'intelligence artificielle (IA). La recherche utilise des techniques telles que l'algorithme glouton, la recherche aléatoire et les algorithmes génétiques, et évalue leur performance à travers des expériences de temps de calcul. Les résultats montrent un potentiel d'amélioration de l'efficacité du PCSE et de résolution de problèmes d'optimisation complexes, contribuant ainsi aux avancées théoriques et aux applications pratiques.

Mots-Clés : Problème de Couverture de Sous-Ensembles (PCSE), Optimisation Combinatoire, Temps de Calcul, Résolution de Problèmes, Méthodes Heuristiques, Performance Algorithmique, Intelligence Artificielle (IA).

:الملخص

تبحث هذه الأطروحة في مشكلة التغطية الجزئية (USCP)، وهي مشكلة معقدة في التحسين التوافقي، وتهدف إلى تطوير خوارزميات فعالة باستخدام تقنيات الذكاء الاصطناعي. تستخدم الدراسة تقنيات مثل الخوارزمية الجشعة، البحث العشوائي، والخوارزميات الجينية، وتقيّم أدائها من خلال تجارب زمن الحوسبة. تظهر النتائج إمكانية تحسين كفاءة USCP وحل مشاكل التحسين المعقدة، مما يساهم في التقدم النظري والتطبيقات العملية.

مشكلة التغطية الجزئية (USCP)، التحسين التوافقي، زمن الحوسبة، حل المشاكل، الأساليب الاستدلالية، الأداء الخوارزمي، الذكاء الاصطناعي (AI).

Dedication

This thesis is dedicated to everyone who has prayed for me, inspired me, encouraged me, challenged me, guided me, helped me, taught me, and supported me throughout this journey. Your collective efforts have enabled me to reach this destination, and I hope this work serves as a humble inspiration for others in the future.

Acknowledgment

First and foremost, I would like to express my profound and sincere gratitude to the supreme power, Almighty Allah who is the one that has given this opportunity to me and has guided me through. I extend my heartfelt appreciation and gratitude to my dear parents (Tajudeen M.B Adigun and Qudirat Salau), my beloved sisters (Aisha and Zainab Tajudeen), and other family members for their unconditional love and support. Their encouragement kept me motivated during the challenging times of this journey. A special thanks to my friends (Farouk Sidi, Khadija Kabir, and others) for your unwavering support, and understanding, and for always being there to provide a listening ear and words of encouragement. Your friendship has been a source of strength and motivation throughout this process. I am deeply and genuinely grateful to my supervisor and head of department (Dr. ROUAN Serik Mehdi) for his unwavering support, patience, exceptional and insightful guidance, and continuous encouragement throughout. His expertise and constructive feedback were invaluable in shaping the direction and encouragement have been pivotal to the completion of this thesis. To my sensational and extraordinary teachers (Mr Achache Habib, Mr. Benfekir Abderrahmane, Mr. Kacimi Abderrahmane, Mr. Titah Mouloud, Mrs. Belkhadja Lilla, Mrs. Benaicha Halima, and others), You've etched your presence into my life's journey both academically and non-academically. It's my pleasure to be your student. To my amazing colleagues (Abdulrahman(s), Muhd Imed Merabet, Abdennour Belbachir, Amira Sahraoui, Souad Smaine, Shahrazad Touati, Wissem Chergui, and others), thank you for your unwavering support and understanding that enriched my experience. Your camaraderie and friendship have been a source of strength and motivation throughout. Your encouragement has made this journey memorable and rewarding. I am immensely appreciative of my institution (the University of Oran 2) and the entire administrative staff of the Institute of Maintenance and Security Industry (IMSI) for trusting and thus offering me this opportunity to be part of its family. I am also grateful to all those who contributed, directly or indirectly, from the beginning of my journey to Algeria to the successful completion of this thesis. Your support and encouragement have been instrumental in this achievement. Finally, I want to thank ME for not giving up, for persevering through every challenge, for believing in my strength, and for continuing to strive toward my goals even when times were tough.

List of Figures

Figure 1 Mathematical model of an Artificial Neural Network (ANN). 22
Figure 2 Structure of CNN. 25
Figure 3 Chromosomes B 46
Figure 4 Chromosomes A 46
Figure 5 GA selection operation 47
Figure 6 GA crossover operation 47
Figure 7 GA crossover operation 48
Figure 8 Flow chart of a typical GA 48
Figure 9 Screenshot from PC with AC_01_cover.txt file 58
Figure 10 Random search solutions for 10 attempts 68
Figure 11 Genetic Algorithm solutions for 10 attempts 69
Figure 12 Bar chart comparison for Random search vs Genetic Algorithm 71
Figure 13 Graphical comparison of Random search vs Genetic Algorithm 72

List of Tables

Table 1 The evolution of different strategies for Artificial Intelligence (AI) 19
Table 2 Result of Random Search Solutions and Runtimes 67
Table 3 Result of Genetic Algorithm Solutions and Runtimes 69
Table 4 Comparison of Greedy, Genetic Algorithm, and Random Search 70
Table 5 Comparison of 10 Attempts for Genetic Algorithm, and Random Search…..……71

List of Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
BMCP	Binary Multiple Choice Knapsack Problem
BTC	Bounded Tree Covering Problem
CNN	Convolutional Neural Network
COP	Combinatorial Optimization Problem
DAG	Directed Acyclic Graph
DL	Deep Learning
DNA	Deoxyribonucleic Acid
DNFW	Deep Neural Network Based on Fuzzy Wavelet
DNN	Deep Neural Network
DP	Dynamic Programming
FAA	Federal Aviation Administration
FPT	Fixed-Parameter Tractable algorithm
FSDGSP	Flow Shop Sequence-Dependent Group Scheduling Problem
GA	Genetic Algorithm
GA	Greedy Algorithm
GPU	Graphics Processing Units
IBM	International Business Machine
ILP	Integer Linear Programming
IP	Integer Programming
LB	Lower Bounding
LMI	Linear Matrix Inequality
LP	Linear programming
LPV	Linear Parameter-Varying
MCLP	Maximal Covering Location Problem
MLP	Multilayer Networks
MMKTC	Min-Max K-Tree Covering problem
OR	Operations Research
PSO	Particle Swarm Optimization
RBF	Radial Basis Function
RIITP	Rough Integer Interval Transportation Problem
RL	Reinforcement Learning
USCP	Under Set Cover Problem
SPP	Set Packing Problem
SPP	Set Partitioning Problem
SUS	Stochastic Universal Sampling
TSP	Traveling Salesperson Problem
SVR	Support Vector Regression

Table of Contents

Abstract ii
Résumé ii
الملخص: ii
Dedication iii
Acknowledgment iv
List of Figures v
List of Tables v
List of Abbreviations vi
Chapter 1: Introduction 1
Chapter 2: Under Subset Coverage Problems (USCP) 2
2.2 Mathematical Formula 2
2.3 Methods of Solving Under Subset Coverage Problem (USCP) 3
2.3.1 Exact Cover Problems 3
2.3.2 Maximum Coverage Problem 6
2.3.3 Weighted Set Cover Problems 9
2.3.4 Set Packing Problems. 11
2.3.5 Set Partitioning Problem 14
2.4 Conclusion 17
Chapter 3: Artificial Intelligence (AI) 18
3.1 Introduction 18
3.2 History of Artificial Intelligence (AI) 19
3.3 Artificial Intelligence Techniques 19
3.3.1 Artificial Neural Networks (ANNs) 19
3.3.2 Convolutional Neural Networks CNN 22
3.3.3 Fuzzy Logic 26
3.3.4 Machine Learning Technique 27
3.3.5 Deep Learning (DL) 29
3.4 Application of Artificial Intelligence (AI) 29
3.5 Conclusion 31
Chapter 4: AI Technique on Optimization 32

4.1 Introduction 32
4.2 Basic Concepts of Optimization, ... 33
4.3 Method of Solving Under Subset Coverage Problem USCP Using AI 33
4.3.1 Greedy Algorithms (GA) 33
4.3.2 Dynamic Programming (DL) 36
4.3.3 Linear Programming (LP) 40
4.3.4 Genetic Algorithms (GA) 43
4.3.5 Random Search Algorithms 49
4.3.6 Particle Swarm Optimization (PSO) 53
4.4 Conclusion 55
Chapter 5: Conception and Implementation 57
5.1 Introduction 57
5.2 Implementation Strategies 57
5.2.1 Materials Used 57
a. Greedy Algorithms 59
b. Random search algorithms 60
c. Genetic Algorithms 62
5.3 Conclusion 64
Chapter 6: Result and Discussion 66
6.1 Introduction 66
6.2 Experimental Setup 66
6.3 Results and Discussion 66
6.4 Comparative Analysis 70
6.5 Conclusion 72
Chapter 7: General Conclusion and Perspectives 73
7.1 General Conclusion 73
7.2 Perspectives 73
7.3 Conclusion 75
References 76

Chapter 1: Introduction

In the contemporary world, optimization problems are pervasive across various domains, including logistics, telecommunications, manufacturing, and finance. Among these, the Under Set Cover Problem (USCP) holds a significant place due to its wide range of applications and its theoretical importance in computational complexity. The USCP is a classic question in combinatorial optimization, where the objective is to select the smallest subset of sets that cover all elements in each universe. This problem is not only fundamental in theory but also critical in practice, as many real-world issues can be modeled or approximated as a set cover problem.

The increasing complexity and scale of modern problems necessitate the development of efficient algorithms and heuristics to tackle USCP. Traditional exact methods often become impractical for large instances due to their computational intensity, leading to a growing interest in heuristic and metaheuristic approaches. This thesis aims to explore various strategies for solving the USCP with applications, focusing on greedy algorithms, random search, and genetic algorithms, and discuss artificial intelligence (AI)

The primary objective of this research is to use AI techniques to solve optimization problems under USCP, and to evaluate the performance of proposed methods by Conducting some computational experiments on instances to compare the performance of the developed algorithms against existing methods, optimizing their parameters, and assessing their performance.

The USCP remains a challenging and significant problem in combinatorial optimization. This thesis aims to advance the understanding and solution of the USCP through the development of innovative heuristic methods. By exploring genetic algorithms, and greedy algorithms techniques, this research contributes to both theoretical advancements and practical applications, providing a robust foundation for future studies in the field**.**

Chapter 2: Under Subset Coverage Problems (USCP)

2.1 Introduction

Under subset Coverage Problems (USCP) is also known as set covering problems (SCP). Our interest in (USCP) emerges from its use in minimizing the number of patterns necessary to distinguish observations from a given population. An effective method, intended to handle substantial instances of USCP, is required to build a pattern creation and pattern minimizing strategy with high classification power.[1] Let *P* be a set of points, and *D* be a set of axis-parallel unit squares in the plane square. In a subset $C \subseteq D$ of unit squares, a point $p \in P$ is considered uniquely covered if included in precisely one square in *C*. These involve a pair (*P*, *D*) of points and axis-parallel unit squares. We are given input and instructed to discover a subset C ⊆ D that maximizes the number of points uniquely covered by *C*. [2]

2.2 Mathematical Formula

Mathematically, USCP is defined as Let A = (a_{ij}) be an *m*-row, *n*-column, zero-one matrix. We say a column j can span a row *i* if $a_{ij} = 1$. Each column *j* relates to a non-negative real cost (c_j). Let $I = \{1, ..., m\}$ and $J = \{1, ..., n\}$ represent the row and column sets, respectively. The USCP requires a minimum cost subset $S \subseteq J$, where each row $i \in I$ is covered by at least one column j ∈ S. The mathematical model for the USCP is expressed as follows:

Maximize.

$$minimize\ f(x) = \sum_{j=1}^{n} c_j x_j \quad (1)$$

Subject to

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1, \quad \forall i \in I \quad (2)$$

$$x_j \in \{0,1\}, \qquad \forall j \in J \quad (3)$$

If the costs cj are equal for each j ∈ J, this is referred to as the uni-cost SCP. Otherwise, the problem is termed the weighted or non-uni-cost SCP, where Ji = $\{j \in J: a_{ij} = 1\}$: the subset

of columns covering row i and Ij = {$i \in$ I: a_{ij} = 1}: the subset of rows covered by column j. The purpose is to minimize the total cost of the selected columns (x_j = 1 if column j is in the solution, 0 otherwise). Constraints guarantee that each row has at least one column. [3]

2.3 Methods of Solving Under Subset Coverage Problem (USCP)

There are various methods of solving the Under Subset Coverage Problem (USCP) with optimization techniques.

2.3.1 Exact Cover Problems

2.3.1.1 Introduction

Exact Cover is a nondeterministic polynomial-time (NP) complete problem with practical applications in resource scheduling, including aircraft fleet planning and cloud computing resource allocation. Solving NP-complete problems on typical sequentially functioning computers is challenging due to the exponential growth of the solution space. Exact coverage is formally defined as follows: Given a collection S of subsets, each having items from a target set X, Exact Cover asks if a subcollection S* of S exists, such that each element in X is contained in exactly one subset in S*.

In other words, the instance has an Exact Cover when all subsets in S*(i) are pairwise disjoint (i.e., ∀Si, Sj ∈ S*. Si ∩ Sj=φ) and (ii) give X when combined (i.e., S=∪ S di ∈S *Si). For a five-set example, given the goal set X={1,2,3,4,5} and the collection of subsets S= [{4,5},{2,3,5},{1,4}, {1,3}, {1,2}], the subcollection S*= [{2,3,5},{1,4}] is the answer. However, the subcollection S**= [{4,5}, {2,3,5}] is not a solution as "1" is absent and "5" occurs twice.[4]

Few studies have been conducted on the precise algorithm for the set covering problem. A weighted mutually exclusive maximum set covering issue was given an accurate method by Lu S and Lu X, with a time complexity of $O(1.325n)$, where m is the cardinality of the cover. Using the approach of measurement and treatment, Zhou Xiaoqing et al. analyzed the running time bound of the algorithm given by Lu S. They found that the algorithm's time complexity is $O(1.3132n)$, which is superior to the outcomes of conventional methods. For the 3-way point covering the issue, Kardoš provided an accurate method with a running time of $O(1.5171n)$. Tu and

Katrenič provided exact Fixed-Parameter Tractable algorithms (FPTs) with running times of O ($2k \times k3.376 + n4 \times m$) and O ($1.882k \times nO(1)$) for the 3-way point covering problem, respectively. Here, k is the parameter of the parameterized problem instance. Chen Kaiqi created an exact method for the Min-Max K-Tree Covering problem (MMKTC) using dynamic programming technology. The complexity of this algorithm is demonstrated to be $O(3n)$ using the binomial theorem. In parallel, he created two precise algorithms for the Bounded Tree Covering Problem (BTC) using dynamic programming technology, demonstrating that the method's respective complexity is $O(3n)$ and $O(2n)$. When analyzing the time complexity of the above exact algorithm, only the cover scale m or the base set scale n are considered. In contrast, the minimum set covering the problem's time complexity cannot be simply expressed in terms of n or m because it is related to both n and m. However, there is still an opportunity for improvement in terms of the overall number and length of search pathways, as the obtained time complexity is of the exponential order of n or m.[5]

In certain cases, a generalized variant of the exact cover issue is more effective for solving specific situations. Add more rows to convert a generalized problem to an exact cover problem. However, the other direction is not usually possible. The generalized issue separates the matrix into major and secondary columns, each with various restrictions. The answer requires at least one non-zero element in each primary column, as previously stated. The solution's secondary columns can have zero or one non-zero element, rather than just one. Let C_P represent the principal columns and C_S represent the secondary columns in matrices A and B. The subset of rows R_B is an exact cover if both of the following equations are met.

$$\sum_{i=1}^{k} b_{i,j} = 1 \; \forall j \; \in C_P \; \wedge \; \sum_{i=1}^{k} b_{i,j} \leq 1 \forall j \; \in C_S (4)$$

[6]

2.3.1.2 Applications of Exact Cover Problems

a. Maintenance constraints

Maintenance constraints are essential for airlines to ensure the location and timing of maintenance activities. These constraints can take various forms, such as guaranteeing a minimum number of aircraft overnighting at specific airports with maintenance stations for specific equipment types. The model enforces minimum and maximum numbers of aircraft in each equipment type overnighting at specified stations, incorporated as simple lower and upper bounds on the runways.

b. Fleet assignments.

Air coverage represents joint resource restrictions in the fleet assignment model, which is a multicommodity flow issue where equipment types are commodities. Decision variables are based on a time-staged air network model of the schedule, and they relate to flow along the arcs in the network. An aircraft of this type traveling along a certain air route is represented by a flight arc in the air network. If an aircraft type E is allocated to a particular air route, the flow on a flight arc equals one; otherwise, it equals zero. The types of aircraft that can be allocated to each air route are restricted by operations, which include constraints on aircraft type, noise levels, overwater flight patterns, and range. By excluding the relevant air equipment from the model's allowed list for a given flight path, these limitations are simple to apply. Ground timings are defined by operations as the necessary amount of time on the ground following a flight's arrival before the aircraft is prepared to take off on a leaving trip. Ground timings are determined by past performance and may vary according to the flight's origin and destination, for example, domestic, Caribbean, or European.

c. Noise restriction.

The Federal Aviation Administration (FAA) is considering adding noise restrictions to airports to limit the number of aircraft operations and noise per seat index level. These restrictions can be implemented as percentage constraint restrictions on less noisy planes, such as limiting operations of certain aircraft types at a specific airport to a given number or percentage of aircraft by equipment type over certain times of the day. The FAA could also impose additional charges based on noise levels at specific airports, adjusting the cost structure accordingly.

Additionally, constraints could be added to the number of parking positions and gates of each fleet type at each station to ensure every station can handle the aircraft assigned to it. The decision to add constraints to a model depends on the value of extra precision versus the effort required in model development and day-to-day solution processes.

d. Crew scheduling.

Crew scheduling is a crucial aspect of airline scheduling, as it is difficult for schedulers to perform manually. Major U.S. and non-U.S. airlines have used a mathematical modeling technique to assign crews to flights. This involves generating many feasible rotations, which are sequences of flight legs owned by a single crew over a few days. Each rotation must satisfy a large set of work rules based on FAA regulations and labor contracts covering crew personnel. The crew scheduling problem is often referred to as determining a collection of rotations that covers all flights exactly once. The problem has two major components: generation of feasible rotations, also known as column generation or matrix generation, and optimization of the resulting matrix. The focus is on the computationally demanding aspects of the problem while reviewing the issues in turning results from this more restricted problem into monthly crew schedules.[7]

2.3.2 Maximum Coverage Problem

2.3.2.1 Introduction

The maximum coverage issue is a standard problem in computer science, computational complexity theory, and operations research. The original phrasing of the problem is as follows. There is a family of sets $\Omega = \{S1, \ldots, Sn\} = S1, \ldots, Sn$ and integer numbers $1 \leq k \leq n$. Sets may share certain components. The goal is to identify a family $\Omega \subseteq \Omega$ of kk sets from Ω that covers the maximum number of elements, i.e., the union $\cup Si \subseteq \Omega Si \cup Si \subseteq Si$ of the selected sets has the maximum size. Scientists are particularly interested in the subject of maximum coverage because of its numerous practical applications. Coverage is related to video surveillance cameras, emergency warning systems, mobile communications, the Internet, radio stations, environmental sensors, and service systems, among others.[8]

Using discrete maximum coverage models as a discrete approximation to the MCSCP, continuously distributed demand and candidate facility sites may be transformed into discrete spatial objects to find suitable locations for facility placement. The most popular Maximal Covering Location Problem (MCLP) was created by Church and ReVelle in 1974. discrete maximum coverage framework. It depends on the notation that follows:

$i =$ index of demand points to be served (entire set I, $|\mathrm{I}| =$ M);

$j =$ index of potential facility locations (entire set J, $|\mathrm{J}| =$ N);

$a_{ij} = \begin{cases} 1, & \text{if demand } i \text{ can be suitably served by facility } j; \\ 0, & otherwise; \end{cases}$

$\Omega_i = \{j | a_{ij} = 1\};$

$g_i = demand\ for\ service\ at\ i;$

$p = number\ of\ facilities\ to\ site;$

$X_j = \begin{cases} 1, & if\ a\ facility\ is\ sited\ at\ potential\ location\ j; \\ 0, & otherwise; \end{cases}$

$Y_i = \begin{cases} 1, & if\ a\ demand\ point\ i\ is\ covered\ by\ a\ sited\ facility; \\ 0, & otherwise; \end{cases}$

The a_{ij} components are determined by a predefined coverage requirement, often distance or transit time. The expected demand for services at i is g_i, which might be the whole population or anticipated customer activity. The choice factors X_j indicate where facilities will be located. Y_i are used to track if demand is being fulfilled by Facility locations. Given this notation, the MCLP formulation is as follows:

$$Z^{MCLP} = \max \sum_i g_i Y_i \quad (5)$$

$$subject\ to\ \textstyle\sum_{j \epsilon \Omega} X_j \geq Y_i, \forall i,$$

$$\sum_j X_j = p. \quad (6)$$

$$X_j = \{0,1\}, \forall j. \quad (7)$$

$$Y_i = \{0,1\}, \forall i. \quad (8)$$

The MCLP's goal (5) is to cover as much demand as possible. Constraint (6) determines whether demand is met by at least one facility. Constraint (7) states where p facilities must be located within the constraints of the budget. Constraint (8) places binary integer limits on decision variables.[9]

2.3.2.2 Applications of Maximum Coverage Problem

a. software package installation

In software package installation, maximizing profits on a server with limited storage means choosing packages with profitable applications and managing their memory requirements. Modeled as a Binary Multiple Choice Knapsack Problem (BMCP), each package represents a software application, with dependent packages as elements. The objective is to select the best package combination to maximize profits within the server's storage constraints.

b. Service provider operations.

The budget reflects a service provider's budget, with each piece representing a service available in the city. Each service has a profit, although it Prepares a facility, such as a base station, for the service. The service can be supported by any of the linked facilities, and each facility requires a certain building cost. To maximize profit from linked services, the service provider must prioritize building subsets of infrastructure within the budget.

c. Worker Employment

A corporation has an employment budget for its staff. Each employee's skills are prioritized for their importance to the company and must be covered by at least one paid worker. Sets represent labor, while components represent skills. The goal is to increase the company's overall skill score by employing employees within budget.

d. Financial decision-making

The budget represents a company's project investment budget. Each set corresponds to an investment leader, a specific employment cost, and a choice of projects in which to invest. Each piece represents a project, along with a potential return if invested.

The objective is to recruit a subset of project leaders within the budget to maximize profits across all connected projects. [10]

2.3.3 Weighted Set Cover Problems

2.3.3.1 Introduction

Although the WEIGHTED SET COVER problem is still NP-complete, we provide an effective fixed parameter exact method for a few practical use cases. We study a problem called TREE-LIKE WEIGHTED SET COVER (TWSC), where we are given a tree-like collection C of subsets (each having some positive real weight) of some base set S. The task is to "cover" S by a minimum weight choice of subsets from C. A subset collection C is called tree-like if the subsets in C can be organized in a tree T such that every subset one-to-one corresponds to a node of T and, for each element s in S, the nodes of T corresponding to the subsets containing s induce a subtree of T. In particular, the term "tree-like" refers to the requirement that the subset collection satisfy an acyclicity property (which will be referred to as the consistency criterion later) that is recognized from the well-known theory of tree decompositions of graphs.

TWSC is motivated by two practical applications: dynamic programming on tree decompositions and computational molecular biology. Dynamic programming on tree decompositions is an efficient solution for hard graph problems, but memory space consumption is an exponential bottleneck. To minimize redundancy, TWSC is formulated, where the tree decomposition serves as the subset collection and each tree node is assigned a positive weight equal to the size of the dynamic programming table associated with it. Experiments show memory savings of around 80 to 90% with TWSC formalization. In computational molecular biology, Page, and Cotton formulate the problem of locating gene duplications as TREE-LIKE UNWEIGHTED SET COVER. They assign each node a positive weight equal to the minimum number of distinct "episodes" of duplications at this node, and solving TREE-LIKE WEIGHTED SET COVER on this tree seems to be a prospective way to answer one of their open questions.[11]

We explore the Weighted Multiple Set Covering Problem (WMSCP), a variant of the traditional Weighted Set Covering Problem (WSCP) that requires components to be covered several times. A comprehensive statement of the WMSCP, the base set P {1, 2, ..., m} contains *n* subsets $P_1, \dots, P_n$ such that $\bigcup_{j=1}^{n} P_j = P$ each $P_j$with a positive real weight c_j, as shown below. A nonnegative integer-valued *n*-tuple $x = [\![(x]\!]_1, \dots, x_n)$ is a multiple cover for P. The number of times an element i is covered is defined as the sum of x_j's for those c_j's that include i, and the overall weight of the multiple cover is defined as $\sum_{j=1}^{n} c_j x_j$.

The minimal weight multiple cover for P that ensures each element i is covered for each given positive-integer-valued n -tuple $[\![(t]\!]_1, \dots, t_m)$ is found using WMSCP. if not more than two times. We may formulate the problem as follows by defining $a_{ij}$to be 1 when $i \in P_j$ and 0 otherwise.

$$\min \sum_{j=1}^{n} c_j x_j (9)$$

Subject to

$$\sum_{j=1}^{n} a_{ij} x_j \geq t_i \quad \forall i = 1, 2, \dots, m, (10)$$

$$x_j = 0, 1, 2, \dots. \quad \forall j = 1, \dots, n. (11)$$

The well-studied WSCP is WMSCP when all t_i's are 1's. Later, we shall consistently indicate $\sum_{j=1}^{n} t_i$ by T for simplicity.[12]

2.3.3.2 Applications of Weighted Set Cover Problems

a. Staff recruitment

The process of hiring staff entails choosing tasks from set *U* and agents from set *N*. A collection $s_i, i \in N$,of jobs that each agent *I* can do is represented as. To choose agents, a manager does interviews, making sure that every assignment from *U* is addressed. There is a fixed salary for each agent. Choosing a minimal cover has financial benefits as it guarantees that the fewest number of agents can do all activities, increasing profitability.

b. Information systems

Let a set of data points $N = U$. The data points that I can send a message to make up the set $s_i, i \in N$. It requires one step to obtain data from a point i to arrive at every point in s_i. Every point in N must be informed of an occurrence by the management. It can take a while to personally relay the information to every place in N. In this situation, the most effective course of action would be to notify just the points that make up a minimal cover. This would guarantee that the message reaches every data point in N in a single step.

c. Transport network.

Networks of transportation. Let $i \in N$ be the set of roads, s_i the set of roads passing through the intersection i, and N be the set of junctions. All roads will be watched over if security cameras are installed at crossroads with the least amount of cover possible. The number of cameras will be reduced.

d. Social network.

The set of agents N is defined as $U \subseteq \{\{i,j\} i,j \in N\}$. If $\{i,j\} \in U$, then there is a conflict between agents i and j. Agent i appears to be a party to the dispute as the set s_i is made up of agent pairs that are at odds with one another. reducing the agents to the barest Ultimately, the group that is free of conflicts and has the greatest number of members is what remains. [13]

2.3.4 Set Packing Problems.

2.3.4.1 Introduction

There are n objects in the Set Packing Problem (SPP), along with a set of exclusive constraints between some of the OB objects designated as O_1, ..., O_m. The goal of the set packing issue is to identify a packing that maximizes the total weight of objects while ensuring that no constraint is broken for each item j ∈ OB and its corresponding positive weight cj. The problem is extensively utilized in several domains, including but not limited to railway operations, scheduling, and routing trains at junctions, combinatorial auction winning bid selection, surgical procedures scheduling, and packet scheduling and transmission in communication networks.

Given a collection of sets of $I = \{1, \dots, n\}$and of valuated items, *and {T}*, $j \in J = \{1, \dots, m\}$, a collection of subsets of *I*, A packing is a subset $P \subseteq I$ such that |T∩P|

≤1, $\forall\, j \in J$. A collection of constraints between the elements of the set I can alternatively be thought of as the set J. Maximizing the overall value of the packing is the SPP's goal. [14] A way to formulate this issue is using a mathematical model:

$$\max \sum_{j=1}^{n} w_j x_j \ (12)$$

s.t

$$\sum_{j=1}^{n} a_{ij} x_j \leq 1 \quad \text{for} \quad i = 1, \dots, m, (13)$$

Where, a_{ij} are 0/1 coefficients and the w_j are weights[15]

$$\begin{bmatrix} \max z = \sum_{i \in I} c_i x_i \ (15) \\ \sum_{i=1} t_{ij} x_i \leq 1, \forall j \ \in J \ (16) \\ x_i \ \in \{0,1\}, \forall i \ \in I \ (17) \\ t_{ij} \ \in \{0,1\}, \forall i \ \in I, \forall j \ \in J \ (18) \end{bmatrix}$$

considering

a vector $x = (x_i)$ where $x_i = \begin{cases} 1 & \text{if } i \in P \\ 0 & otherwise \end{cases}$ vector $c = (c_i)$ where $c_i =$ value of item i

$$\text{a matrix } t = [\![(t)]\!]_{ij}) where\ t_{ij} = \begin{cases} 1 & \text{if } i \in T_j \\ 0 & otherwise \end{cases}$$

The node packing problem (2) is a specific case of the SPP in which the constraints are between pairs of items:

$$\begin{bmatrix} \max z = \sum_{i=I} c_i x_i \ (19) \\ x_i + x_j \leq 1, \forall \ pair\ (i,j) (20) \text{ of incompatible items} \\ x_i \ \in \{0,1\}, \quad \forall i \ \in I \ (21) \end{bmatrix}$$

A node packing problem may be used to design any SPP, with each constraint being divided into many constraints, one for each pair of items. The number of constraints

is increased by this reformulation, though, as each constraint involving n items is now expressed as C^2_n constraints on two items. A straightforward variable change ($x_i = 1 - x_i, \forall i \in I$) allows every node packing issue to also be represented as a set covering problem. [14].

2.3.4.2 Applications of Set Packing Problem

a. Material Properties.

The non-homogeneous characteristics of metal, such as grain orientation, restrict the range of orientations that may be used for objects in the sheet metal sector. The components may only be rotated at a certain angle if bending operations are performed afterward. Additionally, the fabric may have a pattern or certain directional qualities that limit the pieces' orientation to 0° and 180°. Mirroring the pieces is not always achievable since the other side of the cloth may have different qualities. Natural materials have different properties in different regions, like leather. Defects and color variations may be the cause of the quality discrepancy. The pieces' necessary quality must coincide with the corresponding quality zones on the object during the nesting process.

b. Cutting process.

The process of creating the pattern is greatly influenced by the cutting method employed to get the pieces. The minimal distance between the pieces depends on the cutting process (e.g., laser and plasma cutting, stamping). We call this parameter "bridge width." There is a certain breadth to the procedure when using a laser or plasma cutter. A certain space between adjacent forms is required to prevent damage to the pieces. If the bridge width is too tiny, the material tends to slide at the cutting edges during stamping procedures. The cutting length is a crucial factor that also influences the cutting process. It is possible to optimize layouts such that the entire cutting route is minimized when cutting every element.

c. Scheduling

The order in which the components are cut might have an impact on how they are manufactured later. In this instance, separate processing processes for the pieces are required. When designing huge layouts, a certain distribution of the components is made easier regarding the sheets. The order in which the pieces are assembled may

also be crucial for packaging and delivery. The pieces may need to be packaged in a certain order due to weight and geometrical restrictions. In certain cases, nested order lists are used to optimize material usage within a single arrangement. So, dispatching is likewise influenced by the pieces' order of succession.

d. Nesting process

It is possible for the pieces that need to be nested to have empty spaces in them. Some of these spaces could be big enough to consider for nesting smaller objects. Within the shipbuilding industry, this method is widely used and is called in-hole-nesting. To minimize waste, the tracking and nesting algorithm must be able to accommodate irregularly shaped empty spaces. There aren't always enough relatively tiny forms in the present nesting assignment. Since raw materials are frequently too valuable to be thrown away, certain filler elements might be chosen and utilized in their place. Although they are made for stock, they are not a part of the present order and might not be needed right away. Larger nesting tasks in the shipbuilding industry may involve materials of different types, such as sheets of different thicknesses. The nesting algorithm must decide on the best allocation based on availability, as several thicknesses may be suitable for a subsection of the order list.[16]

2.3.5 Set Partitioning Problem

2.3.5.1 Introduction

In the domain of operations research, a Set Partitioning Problem (SPP) is a well-known optimization problem that is defined as follows. Let *M* be a family of sets $\{M_1, M_2, \cdots, M_m\}$, where each M_j is a subset of *N*. Let *N* be a set with n members $\{N_1, N_2, \cdots, N_n\}$. A subset of *M*, indexes, X, is termed a cover if $U_{j\in X}\ M_j = N$ is fulfilled. If, for each $j_1, j_2 \in X$, $M_{j1} \cap M_{j2} = 0$ is satisfied, the cover X is called a partition. The issue of finding a partition X that minimizes the total of the costs $\sum_{j\in X} c_j$ is known as a set partitioning problem if a constant c_j, called a cost, is defined for each M_j. [17] Mathematically, the set partitioning problem (SPP) can be expressed as

$$Minimize\ z = \sum_{j=1}^{n} c_j x_j \quad (23)$$

subject to

$$\sum_{j=1}^{n} a_{ij}\, x_j = 1 \quad \text{for } i = 1, \dots, m (24)$$

where $a_{ij}$is binary for all i and j, and$a_{ij} > 0$. Finding the values of the binary variables x_j that minimize the objective function z is the aim.

The most well-known use of the SPP is airline crew scheduling. In this approach, each row $(i = 1, \dots, m)$ represents a required flight leg (takeoff and landing). The columns $(j = 1, \dots, n)$ reflect the permitted round-trip rotations (pairings) that an airline crew might take. Each crew assignment to a flight leg carries a cost (cs). The matrix elements (a_{ij}) are defined as

$$a_{ij} = \begin{cases} 1 & \text{if flight leg } i \text{ is on rotation } j \\ 0 & \text{otherwise.} \end{cases}$$

The scheduling of airline personnel is a prominent and economically relevant issue. The airline crew scheduling issue is often discussed in the literature on (OR). It has been estimated that the annual expenditures of pilots and flight attendants exceed one billion dollars. A minor enhancement compared to current systems might yield a substantial financial gain. Previously, the SPP's solutions were created by hand. However, the scale and complexity of airline crew scheduling issues have increased dramatically. 400 row and 30,000 column issues were considered "very large" in 1981. Currently, issues with hundreds of thousands of columns are considered "very large," and a benchmark problem including 837 rows and 12,753,313 columns has been created.[18]

2.3.5.2 Applications of set partition problems

a. Auction

The auctioneer's goal is to discover a way to divide up the assets among the bidders in a way that maximizes her earnings, given a set of assets (individuals) and a set of bids that assign a price to groupings of assets (admissible parts). Compared to the SPP, the winner determination issue is strictly broader since several bidders may place a price on the same collection of assets. However, it has been demonstrated by

Rothkopf et al. that the winner determination issue is like the Set Packing problem when the bidder objectives are expressed using the standard OR bidding language1. Furthermore, because there are always positive bids and null values are assigned to the lack of a bid on a particular asset, there is always an ideal packing that is also an optimal partition. Thus, the winner determination issue is the same as the SPP in this situation.

b. Date analysis.

The conventional clustering problem may also be broadly formulated as the SPP: data points must be divided into classes to maximize the intra-class similarity and/or the inter-class dissimilarity. Thus, the problem of finding the best partition emerges whenever someone wishes to categorize, organize, or abstract data. The SPP is related to data aggregation issues in this context. To preserve the structure of the system and optimize a specific compression rate, data points are divided into homogeneous classes. This process has applications in time series analysis, spatial analysis, multilevel community detection, database optimization, and image processing.

c. Game theory.

To complete an assigned goal, a population of self-motivated agents (individuals) must cooperate. To maximize social welfare (as indicated by the cost function) and produce a better outcome, coalition structure generation entails dividing the agents into viable teams (admissible parts). This area of study opens up a wide range of applications, including e-business (groups formed to meet specific market niches), e-commerce (buyers form coalitions to purchase products in bulk and take advantage of price discounts), distributed sensor networks (sensors collaborate to track targets), distributed vehicle routing (delivery companies band together to save on transportation costs), and information gathering (servers collaborate to respond to inquiries).

d. Optimizing problem.

The SPP's widespread applicability can be partially attributed to its similarities to the well-researched Set Packing (SP) and Set Covering (SC) problems, which relate to the relaxation of the "covering" and "pairwise disjoint" constraints concerning admissible partitioning, respectively. A plethora of other possible applications result

from the SPP's generalization of various combinatorial optimization problems, particularly in computational graph theory where individuals are graph vertices and admissible parts are defined based on a particular subgraph structure: for example, graph coloring, graph partitioning, dogmatic partitioning, weighted multiway cut, minimum set cover, partition into Hamiltonian subgraphs, forests, or perfect matching. However, this paragraph mainly discusses in further detail instances in which sets of acceptable parts and cost functions are supplied by expert knowledge about the analyzed population and its semantic or structural qualities, as this study concentrates on applications in specializations in operational research and artificial intelligence. [19]

2.4 Conclusion

Subset coverage problems, vital in computer science and mathematics, include the set cover and exact cover problems, both NP-complete or NP-hard. Despite their complexity, they have wide applications in real-world scenarios like resource allocation and facility location. Efficient solutions are elusive due to their NP-hardness, but approximation algorithms and heuristics offer practical near-optimal solutions. Ongoing advancements in optimization techniques and computational methods, including metaheuristics and integer programming, contribute to solving these challenges. Emerging technologies like quantum computing hold promise for tackling subset coverage problems. The field continues to evolve, driven by the need to optimize resource utilization and system performance in various domains. Subset coverage problems bridge theory and practice, emphasizing the significance of balancing computational complexity and solution quality. Researchers strive to refine existing algorithms and develop novel methodologies to address practical challenges efficiently. The quest for efficient solutions to subset coverage problems remains paramount in optimizing complex systems and addressing real-world challenges.

Chapter 3: Artificial Intelligence (AI)

3.1 Introduction

Artificial Intelligence (AI) refers to a machine's ability to perform functions identical to human thought. AI refers to computer systems and programs that can execute sophisticated tasks beyond simple programming, although it is still not comparable to human thought. According to Barr and Feigenbaum, AI is defined as the design of intelligent computer systems that display human-like behaviors such as understanding, language, learning, reasoning, and problem-solving. Intelligent computing technologies are increasingly valuable as alternatives to traditional methods or as components of integrated systems.[20]

Artificial Intelligence (AI) has grown more prevalent in our daily lives, with existing technologies.[21] AI is advancing rapidly. AI can range from e-commerce prediction algorithms to International Business Machine (IBM) Watson machines while being typically depicted in science fiction as human-like robots. AI focuses on specific tasks such as facial recognition, online searches, and driving. General AI outperforms humans in almost all cognitive tasks, as well as specific tasks like chess or equation solving. [22]

AI is a field of computer science aimed at designing intelligent computer systems with intelligent characteristics like human behavior. This is also aimed at emulating human cognition, creating intelligence without considering human characteristics or creating useful artifacts for human comforts and needs. However, a proper definition of AI is difficult to establish, as it requires a general understanding of input, output, and their relationship based on system structure. Without a general definition, complex systems may not be analyzed for performance and error analysis.

The scientific goal of AI is to determine theories about knowledge representation, learning, rule-based systems, and search, while the engineering goal is to acquire machine abilities to solve real-life problems using techniques like knowledge representation, machine learning, rule systems, and state space search. While computer scientists and engineers have traditionally focused on engineering goals, psychologists, philosophers, and cognitive scientists are more interested in scientific goals. Both approaches can feed each other, ensuring a balanced approach to AI research.[23]

3.2 History of Artificial Intelligence (AI)

It is pertinent that we look at the origins of the notion of Artificial Intelligence (AI) before exploring the history of AI applications. AI was first conceptualized in the 1950s, according to i-Scoop (2016). Alegre (1991) claims that a convergence of concepts produced during a symposium that scientists held in the summer of 1956 on the campus of Dartmouth College, USA, led to the coining of the term Artificial Intelligence (AI). Table 1 displays the timelines for the growth of AI techniques and the people who developed them in chronological order. It is worth mentioning that artificial neural networks were the first AI techniques to be developed, while hybrid intelligent systems are the most recent.

AI technique	Year of development	Developer
Artificial neural network	1943	McCulloch and Pitts
Fuzzy logic	1965	Lofti A. Zadeh
Genetic algorithm	1970	John Holland
Case-based reasoning	1977	Schank and Abelson
Support vector machines	1995	Vapnik V
Particle swarm algorithm	1995	Eberhart and Kennedy
Hybrid intelligent systems	-	-

Table 1 The evolution of different strategies for Artificial Intelligence (AI)

[24]

3.3 Artificial Intelligence Techniques

Numerous Artificial Intelligence (AI) techniques are available, each suited for different applications and scenarios.

3.3.1 Artificial Neural Networks (ANNs)

3.3.1.1 Introduction

Artificial Neural Networks (ANNs) are computer systems that imitate human intelligence and pattern recognition. These systems, which are closely modeled after the brain, are intended to replicate these processes to the greatest extent feasible. ANNs are founded on the assumption that humans can complete these jobs with little effort, making them an excellent choice for computing systems.

An Artificial Neural Network (ANN) is a component of the nervous system made up of linked neurons. The term "neural" refers to neurons, whereas "network" refers to a graph-like structure. ANNs are sometimes referred to as "Neural Nets," "Parallel Distributed Processing Systems," and "Connectionist Systems." In an ANN, each node conducts basic computations, and each link transmits a signal from one node to another, identified by a quantity known as "Connection Strength" or "Weight." This approach offers a more efficient and accurate depiction of human intellect and knowledge. As a result, ANNs represent a viable option for developing computing systems capable of mimicking human intelligence and pattern recognition. A.O. Kurban investigated artificial neural networks (ANNs), which are non-linear mapping systems based on biological nervous system concepts.

A typical actual neuron consists of a dendritic tree that collects signals from several neurons, a cell body that integrates these signals, and a branching axon that distributes the response via connections with other neurons. Each neuron's response is a simple non-linear function of its inputs, which is heavily influenced by the strength of connections. Despite their simplicity, NNs with several neurons may produce complicated behaviors. NNs are made up of basic processors connected by weighted connections, or "neurons," with each node's output based on locally available data. Each unit takes input from a large number of other nodes and sends output to them.[25]

The ANN model has three primary layers: input, hidden, and output. These layers are made up of basic processing units, known as neurons or nodes, that are coupled by weighted connections. The study focuses on determining the number of hidden layers and nodes, which is a prominent research approach. Montesinos López et al. share further details about the ANN model. The one-hot encoding approach transforms categorical data into integers by expressing each input marker as a binary vector. This approach works effectively with category data that are unrelated. Numerical data inputs, on the other hand, are fed directly into models. Choosing a network architecture and parameters is usually done empirically, however commercial tools such as Qnet v2000 for Windows can assist in identifying the best ANN model. Silva et al. discuss this technique, which was created by Vesta Services.[26]

3.3.1.2 Features of Artificial Neural Networks (ANNs)

Neural networks are identified by three main features: Topology, Learning paradigm, and Data representation.

A. The topology of a Neural Network (ANN) refers to the arrangement and location of neurons in relation to input or output. A network's core characteristics are the number of layers, the number of neurons per layer, the connection grade, and the kind of connection between neurons. ANNs are classed as monolayer or Multilayer Networks (MLP), with the former having a single input layer and the latter including intermediate or hidden layers.

B. The learning paradigm entails adjusting the weights of the ANN based on the input information. Changes can be made by deleting, increasing, or changing the connections between neurons. The learning process is deemed complete when the values assigned to the weights remain constant.

C. Input/output data may be represented in ANNs in a variety of ways. Many networks employ analog input and output data, with either linear or sigmoidal activation functions. Some networks accept discrete or binary values for input data, with neurons triggered by an echelon function. Hybrid ANNs exist in which input data can take continuous values while output data offers discrete values, or vice versa.

To summarize, neural networks are distinguished by their topology, learning paradigm, and data representation. Understanding these traits is critical for developing effective and efficient neural networks.

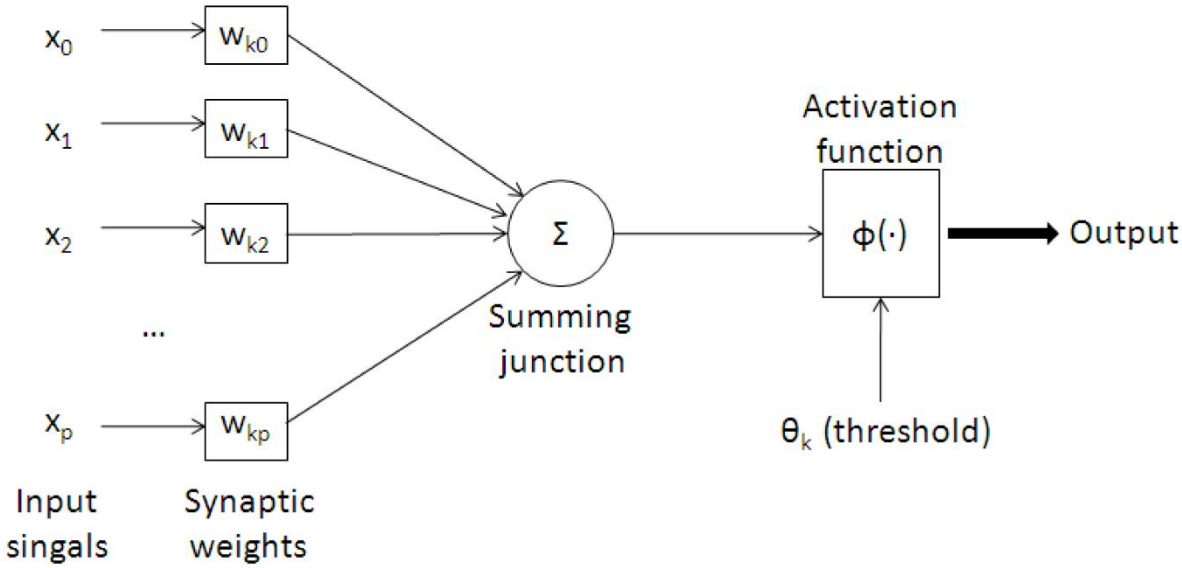


Figure 1 Mathematical model of an Artificial Neural Network (ANN).

[27]

3.3.2 Convolutional Neural Networks CNN

3.3.2.1 Introduction

To learn thousands of objects from millions of images, a model with a large learning capacity is needed. However, the object recognition task is complex and cannot be specified by a dataset as large as ImageNet. To compensate for this, a model with prior knowledge is needed. Convolutional Neural Networks (CNNs) are a class of models that can control their learning capacity by varying depth and breadth. They make strong assumptions about image nature, such as stationarity of statistics and locality of pixel dependencies. Compared to standard feedforward neural networks, CNNs have fewer connections and parameters, making them easier to train. However, they are still prohibitively expensive to apply on a large scale to high-resolution images. Current Graphics Processing Units (GPUs) and optimized 2D convolution implementations can facilitate the training of large CNNs, and recent datasets like ImageNet contain enough labeled examples to train these models without severe overfitting. [28]

CNN is a deep neural network with input, convolutional, sampling, full connection, and output layers. It uses input data like numbers, text, or images. The C layer extracts local features by connecting each neuron to the local sensory region. The S layer maps features,

with each layer consisting of multiple feature mappings. The activation function is often a sigmoid function with a smaller influence function core. The full connection layer expresses high-dimensional spatial data. The output layer outputs the test results of the neural network. CNN reduces the number of training parameters through local connection and weight sharing, making it a more efficient method compared to traditional statistical pattern recognition algorithms.

CNN's convolution and downsampling (pooling) layers are classified as special hidden layers, whereas the other layers (except the output layer) are considered ordinary hidden layers. The image is represented as P=f (x, y), where f (x, y) represents the grey value of column y in row x above the picture P. The image P has a pixel value of *M* X *N*. The convolution kernel, k (x, y), has a size of a X b. To produce the matrix C (s, t) of picture P and convolution kernel k, perform the following convolution process.

$$C(s,t) = f(x,y) \cdot k(x,y) \quad (25)$$

$$= \sum_{x=1}^{a} \sum_{y=1}^{b} (k(x,y) f(s+x-1, t+y-1) \quad (26)$$

$$\text{Where } 1 \le s \le M-a+1, 1 \le t \le N-b+1 (27)$$

The convolution operation is a process that uses a convolution core to walk up an image matrix, multiply its position elements, and add the results to create a new matrix. This process enhances the original signal features and reduces noise. The features are then sampled using a downsampling layer to reduce dimensions. For a matrix *A*, the size is *M* x *N*, and if each block has a size of λ×τ without overlapping, the ij block is expressed as

$$G_{\lambda,\tau}^{A}(i,j) = (a_{st})_{\lambda\tau} (28)$$

Where a_{st} represents the equivalent extended matrix,

$$(i-1)\cdot\lambda + 1 \le s \le i\cdot\lambda, (j-1)\cdot\tau + 1 \le t \le j\cdot\tau \quad (29)$$

The operation of averaging block $G\ A\ \lambda, \tau$ is called average pooling, and its expression is given as

$$down\left(G_{\lambda,\tau}^{A}(i,j)\right)=\frac{1}{\lambda\times\tau}\sum_{s=(i-1)\times\lambda+1}^{i\times\lambda}\sum_{t=(j-1)\times\tau+1}^{j\times\tau}a_{st}\,(30)$$

Pooling is essentially the expression of local characteristics, which can not only minimize the feature dimension, However, you may also extract secondary features. After downsampling, each pixel in the feature maps is extended to create a row-based feature vector. The feature vector is fully related to the output. The computation technique for C and S layers in forward propagation is as follows.

$$h_{k,j}^{l}=f\left(u_{k,j}^{l}\right)=f\left(\sum_{i}h_{k-1,j}^{l}*w_{ij}^{k}+b_{j}^{k}\right)(31)$$

$$h_{k+1,j}^{l}=g(\beta_{j}^{k+1}down\left(h_{k,j}^{l}\right)+b_{j}^{k+1})(32)$$

In this equation, *ul kj* is the convolution function's independent variable, *hl kj* is the *j*th convolution surface of the kth hidden layer H_k, *Wk ij* is the weight matrix from the *i*th input surface to the *k*th input surface, *bk j* is the offset of the *j*th output surface of the kth hidden layer H_k, and *bk + 1 j* is the multiplicative offset of the *(k + 1)*th hidden layer H_{k+1}. The hidden layer Hk + 1 is represented by g(.), a constant linear function with $g(x)=x.f(.)$The activation function, f(.), is typically a sigmoid function with the formula given below.

$$f(x)=\frac{1}{(1+e^{-x})}(33)$$

The backpropagation method calculates the cost function based on the error between the network and the ideal output, which is then propagated layer by layer using the chain rule. The random gradient descent technique adjusts the parameters of each layer of the network until it converges or meets the set iteration termination criteria. Backpropagation algorithms may be used to learn and train weights and biases for CNNs. The backpropagation error signal (or sensitivity) is as follows.

$$\begin{cases} \delta_{k,j}^{l} = \beta_{j}^{k+1}\left(f'(u_{k,j}^{l})^{\circ} up(\delta_{k+1,j}^{l})\right)(34) \\ \delta_{k+1,j}^{l} = g'(u_{k+1,j}^{l})^{\circ}(\delta_{k+1,j}^{l} * rot180(W_{j}^{k+2}))(35) \end{cases}$$

where the up-sampling function for blocks in a matrix is represented by up(.). rot180(.) is a function that depicts the Hadamard product of two vectors by first rotating the matrix vertically and then horizontally once. The partial derivatives of the weights and biases for the convolution layer are.

$$\begin{cases} \frac{\partial LN}{\partial W_{ij}^{k}} = \sum_{l=1}^{N} \delta_{k,j}^{l} * h_{k-1,i}^{l}(36) \\ \frac{\partial LN}{\partial b_{j}^{k}} = \sum_{l=1}^{N} \delta_{k,j}^{l}(37) \end{cases}$$

To converge the network model, the training set changes the weights and biases in both the forward and reverse propagation phases continually. Next, the network's training impact may be seen using the test set, and ultimately, a very effective CNN can be created. [29]

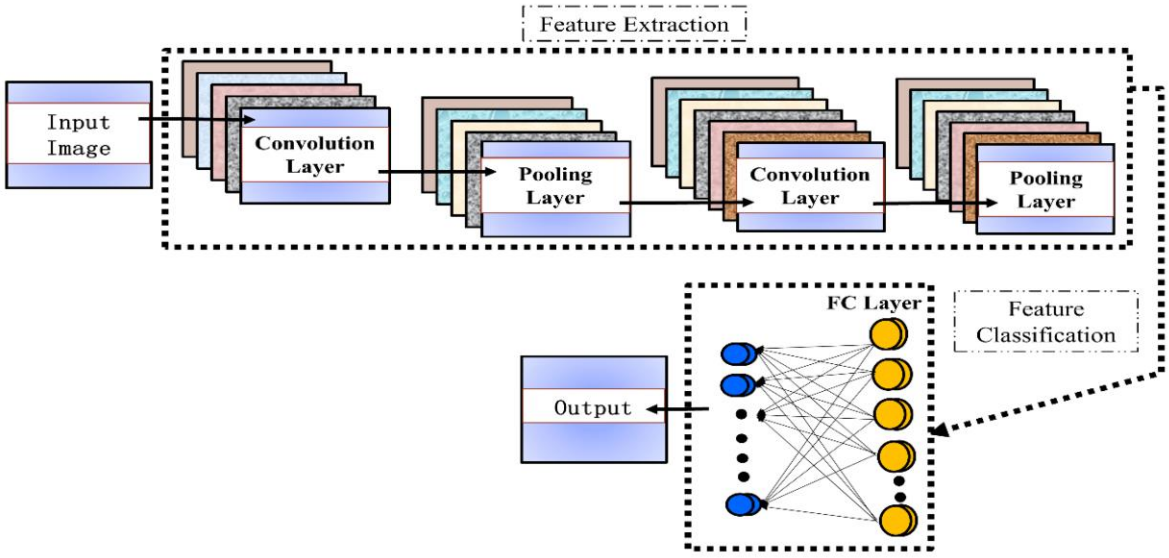


Figure 2 Structure of CNN.

[30]

3.3.3 Fuzzy Logic

3.3.3.1 Introduction

Fuzzy logic was inspired by the fuzzy sets that Zadeh (1965) created. The components of a fuzzy set are given a degree of membership, which is often a value between 0 and 1. Arguments are given varying degrees of validity to produce fuzzy logic. The normal set of real numbers (degrees) is [0, 1], where the remaining digits represent "partial truth," or transitional degrees of reality, while 00 represents "completely false," 11 represents "completely true," and so on. A wide range of conceptual frameworks and methodologies that concentrate on methodical management to some degree is commonly referred to as "fuzzy logic".

In engineering contexts (fuzzy control, fuzzy identification, fuzzy intelligence), the emphasis is on workable solutions that can withstand imprecision and suboptimality. As described in this article, fuzzy logic is defined narrowly, having emerged as a field of mathematics with the seminal work of Hájek et al. (2003). This idea has been called "fuzzy mathematical logic" in the past several years. In the spirit of classical, formal logic, it investigates logic via a realistic portrayal of partial truth.

The Deep Neural Network Based on Fuzzy Wavelet (DNFW) was characterized using TSK fuzzy frameworks that were based on different fuzzy rules. Dilation variables were employed as the result of each wavelet transform, which was made up of many wavelet systems with individually customizable interpretations.

In a TSK fuzzy model, each input is separated into fuzzy regions, each with its own degree of participation in the IF section. The rules' THEN section implements a fixed or linear function of inputs. According to Wang (1996), the IF-THEN criteria are as follows:

$$R_k\text{: If } x_1 \; is \; A_{k1} \dots \dots AND \; x_n \; is \; A_{kn} (38)$$

$$\text{THEN } y_k = \; a_{k0} + a_{k1}x_1 + \cdots + a_{kn}x_n, (39)$$

The kth rule of fuzzification is denoted by R_k whereas x_j and A_{kj} are used. There are fuzzy features and fuzzy sets. Fuzzy membership functions for A_{kj} are normally distributed and are as follows:

$$\mu_{kj}(x_1) = \exp\left(-\left(\frac{x_1 - c_{kj}}{\sigma_{kj}}\right)^2\right) (40)$$

Additionally, ckj shows the center points, whereas σkj displays the standard deviation of the membership function based on k. The TSK fuzzy technique with M rules yielded the following results:

$$y = \frac{\sum_{k=1}^{M} yk \prod_{j=1}^{n} \mu_{kj}(x_j)}{\sum_{K=1}^{M} \prod_{j=1}^{n} \mu_{kj}(x_j)} (41)$$

[31]

3.3.4 Machine Learning Technique

3.3.4.1 Introduction

The system learns from its experience because it is self-learning and is built on algorithms. For example, the kind of data that is sent into the system helps it identify patterns and respond to those patterns at the output. In this instance, the system learns over time and becomes smart without human assistance. It employs a statistical learning algorithm that, independent of human assistance, learns and becomes better automatically. Face recognition has gained popularity in the contact-free world due to its ability to accurately identify individuals based on their unique features. Machine learning technology is being used to improve systems in a variety of industries, including healthcare, banking, retail, travel, airports, Amazon Alexa, Google Home, YouTube, Netflix, and search engines. [32] Machine learning is categorized into four categories. This strategy is critical for predicting and making decisions in the face of unpredictability.

3.3.4.2 Categories of Machine Learning

A. Supervised Learning

In this method, we use tagged data to train our model. Supervised learning refers to learning with a supervisor present. The mapping function, which transfers the I/P variable (X) to the O/P variable (Y), is a key component of this sort of learning. Supervision for model training is the primary phase in supervised learning. Proper learning requires the presence of a teacher or mentor in school. This is divided into two problems:

1. Classification Models: Output variables that may be classed as "Yes" or "No" or "Pass" or "Fail" are considered classification models. We apply these models to predict the data category. These can be either binary or multiclass classification

models. Classification models are commonly used for spam filtering, churn prediction, and other applications.

2. Regression Models: Regression models are used to forecast outputs based on previous data, such as house rent. Popular regression algorithms include linear, polynomial, ridge, and logistic regression.

B. Unsupervised Learning

Unsupervised learning occurs when there is no supervisor present, and no tagged data is available. This technique does not include any training data. A significant amount of data is fed into the machine to generate models and patterns, and then the model is fed with testing data to offer efficient predictions. Unsupervised learning does not have defined results and focuses on identifying unique patterns in each data set. The recommendation system uses unsupervised learning algorithms like k-means clustering and k-nearest neighbors. This category is categorized into:

1. Clustering: Clustering is an unsupervised learning technique that groups similar data points into clusters to organize unlabeled data based on similarities. Its primary purpose is to locate and group comparable data items into clusters, hence enabling data organization and analysis.
2. Anomaly detection is the process of recognizing unusual things or events that stand out from the rest of the data, typically signaling suspicious or anomalous behavior. It is frequently used in fields such as finance and healthcare to discover abnormalities such as fraud or medical errors.

C. Semi-supervised learning

Semi-supervised learning blends labeled and unlabeled data during training to increase accuracy. While tagged data requires professional human intervention, unlabeled data is more cost-effective. This method is used in text document classification since it is impractical for people to read each document in its entirety. Labeled text improves the efficiency of unlabeled text classification. Integrating both sources of input during training improves the model's performance and scalability. This strategy is especially useful when labeled data is limited or expensive to get because unlabeled data can considerably improve the learning experience.

D. Reinforcement Learning (RL)

Reinforcement Learning (RL) is a subset of machine learning involving interactive learning in an environment. Unlike supervised and unsupervised learning, RL agents learn by trial and error, performing actions and receiving feedback. In RL, feedback is provided in the form of positive rewards for desired results and negative punishments for bad ones. This feedback loop directs the agent's decision-making process and influences its behavior over time. RL focuses on action-based decision-making without prior knowledge of the data. The agent's activities have an impact on both current and future situations, as effects are delayed and learned via experience. RL combines feedback mechanisms in a unique way, allowing agents to navigate complex settings and make better decisions. This dynamic method of learning promotes exploration and consequence, differentiating RL as an effective technique for achieving goals in unpredictable contexts. [1], [32]

3.3.5 Deep Learning (DL)

This is a subset of machine learning that uses Deep Neural Networks (DNNs) with several layers between input and output. These layers are made up of neurons with a variety of parameters, which allows for hierarchical feature representation learning. This hierarchical technique, inherent in neural network topologies, allows for automatic feature learning, distinguishing deep learning from standard methods. DNNs use this method to extract and change features, with beginning layers handling simple input processing and succeeding layers learning more complicated features. As a result, deep learning excels at handling enormous datasets and complexity, providing reliable solutions for a variety of problems. High layers in the hierarchy of deep learning models build abstract representations during training, utilizing the compositional functions learned at lower levels, resulting in powerful feature extraction capabilities.[29], [32]

3.4 Application of Artificial Intelligence (AI)

1. Design optimization.

   Design optimization is used to determine the best product configuration, improve product qualities, increase functionality, minimize costs, and create lighter components, among other goals. The goal is to determine the best design while considering a variety of restrictions. In classical optimization processes, experts use extensive engineering knowledge to create a mathematical model of the problem and

identify the best design parameters. Designers are currently using artificial intelligence techniques such as evolutionary algorithms, swarm intelligence, artificial neural networks, and machine learning to address design optimization difficulties while having low engineering experience. (8)

2. AI in medicine

   Medical diagnosis requires both knowledge and experience, with small changes in medical images or signals often difficult for human detection, and particularly challenging for junior doctors. Advances in image and pattern recognition aid diagnosis across various medical data types like MRI and EEG. AI methods like fuzzy c-means and genetic algorithms for feature extraction from medical images, enhance quantitative and qualitative analysis. AI technologies are crucial in early cancer detection using MRI, distinguishing between benign and malignant tumors. Additionally, AI assists in risk stratification for cardiovascular diseases, with systems like ANN-based diagnosis of brain dysfunctions and fuzzy expert systems for heart and liver disease diagnosis based on multiple medical inputs. [33]

3. AI in aerospace

   Artificial intelligence (AI) is widely used in the aerospace sector to support various activities like aero-engine diagnosis, aircraft wear condition analysis, and aerospace alloy parameter optimization. AI used fuzzy logic and neural networks to include AI models in aircraft design, speeding up parameter selection and cutting down on design cycle time. Immune algorithms are among the methods used in aero engines to diagnose wear conditions and find different kinds of problems. Furthermore, to improve manufacturing productivity, artificial intelligence models, like artificial neural networks, are used to forecast surface roughness when milling aerospace alloys. [33]

4. AI in petroleum

   Artificial Intelligence (AI) has made tremendous strides in the petroleum industry, enabling better oil spill detection in oil fields, and optimizing drilling operations. To

maximize drilling penetration rate, a two-model approach that uses neural networks proved to be accurate and efficient. Singha et al. provided evidence of the efficacy of neural networks in feature- and image-based oil spill categorization systems. Additionally, artificial intelligence (AI) algorithms, like as artificial neural networks, are used in pipeline leak detection to identify patterns in pressure distribution, allowing precise leak location prediction following training.[33]

5. AI in the energy sector

   AI plays a crucial role in renewable energy, leveraging historical weather data to forecast energy generation from sources like wind and solar, aiding grid planning. It optimizes renewable energy systems by adjusting parameters for maximum efficiency, reducing costs. AI also optimizes energy storage systems by predicting demand fluctuations, ensuring efficient usage. Additionally, it manages the smart grid by balancing energy supply and demand, integrating renewable sources effectively. Furthermore, AI monitors energy consumption to promote conservation and reduce emissions, contributing to a sustainable energy future.[34]

3.5 Conclusion

The new generation of information technology, represented by AI, may spark a new phase of technological upheaval. As a typical enabling technology, the maturation and deployment of AI technology resulted in widespread penetration into other vital industries of the national economy, changing transportation, manufacturing, medical care, business, and other industries throughout time. Artificial intelligence approaches have previously been employed in a wide range of fields. Artificial intelligence applications in industries such as power, medical engineering, petroleum, and aerospace. Some of these apps have produced unanticipated effects. Furthermore, the current success of artificial intelligence can be attributed to the development of several supporting technologies, such as the Internet of Things, sensors, and big data, all of which have grown rapidly in recent years. We believe that, as AI and related technology clusters mature, the use of artificial intelligence will become increasingly widespread in the future.[33]

Chapter 4: AI Technique on Optimization

4.1 Introduction

The chapter covers the usual form of a general optimization problem, Optimization is the process of finding the optimal solution to a problem. The term optimal is derived from the Latin word "Optimus," which means "best." Since the beginning of his life, Man has strived for the best.[35] where the (problem) objective function f(x) must be minimized while taking equality and inequality constraints into account. Within predetermined bounds, the design variables are limited, and the objective and constraint functions can be either explicit or implicit, linear, or nonlinear. Discrete or integer variables may be used in optimization, which presents potential for global algorithms but difficulties for local ones. Bounds on variables, or side constraints, are taken into consideration individually. While restricted issues have side restrictions, unconstrained problems may have none, but they do have equality or inequality requirements. With the potential for many optima, optimization approaches seek to identify the optimal set of design variables that maximize the objective function while satisfying all constraints.[36]

History of optimization

Optimization theory began with the discovery of a geometric curve that would enclose the largest possible area, with Archimedes correctly conjecturing that the optimal curve was a semicircle. Early results included principles explaining natural phenomena, such as Heron of Alexandria's postulated light travels by the shortest path in 1657. Another branch of optimization focuses on choosing a function that minimizes certain functions. Two problems were known at the time of Newton: finding a curve with minimum resistance when rotating and the brachistochrone problem, where a frictionless bead traveling on a curve will cover the journey in the least time. Euler developed the calculus of variations in 1766, which led to the formulation of many laws of mechanics in terms of principles of optimality. Examples include the least action principle of Maupertuis, the principle of least restraint of Gauss, and Lagrange's kinetic principle. Lagrange invented a method for solving optimization problems with equality constraints using his Lagrange multipliers in 1760, while Gauss developed the least squares curve fitting method.

In 1834, W.R. Hamilton developed Hamiltonians, which united optics and mechanics at the time. In 1875, J.W. Gibbs presented a further principle of optimality concerning the

equilibrium of a thermodynamic system. Since then, there have been increasing contributions each year, with notable achievements from Dantzig and Bellman, as well as Pontryagin's work on the maximum principle used to solve problems in the theory of optimal control. [35]

4.2 Basic Concepts of Optimization,

These focus on the problem of maximizing a real-valued function of several real variables under constraints. The problem is illustrated using the example.

Maximize: $X_0 = f(X) = f(X_1, X_2)$ (42)

subject to $h_1(X) > 0$ (43)

$X_1 > 0$ (44)

$X_2 > 0$ (45)

The objective function is the function, and the constraint set is a set of inequalities. The problem is to find real values for X_1 and X_2 satisfying (43), (44), and (45), which when inserted in (42) will cause $f(X_1, X_2)$ to take on a value no less than that for any other $X_1 > X_2$ pair. Hence, X_1 and X_2 are called independent variables.[37]

Large-scale problem optimization faces challenges like multi-modality, dimensionality, and differentiability. Traditional techniques like steepest descent, linear programming, and dynamic programming struggle to solve these problems, especially with nonlinear objective functions. These techniques require gradient information, making them unsuitable for non-differentiable functions and problems with multiple local optima. To address these issues, more powerful optimization techniques are needed, and research has been ongoing for the past three decades to find effective solutions.[38]

4.3 Method of Solving Under Subset Coverage Problem USCP Using AI

There are several Artificial Intelligence (AI) techniques that can be applied to solve the Under Subset Coverage Problem (USCP) effectively

4.3.1 Greedy Algorithms (GA)

4.3.1.1 Introduction

When it comes to a variety of optimization problems including coverage problems specifically, greedy algorithms are frequently the first ones that are thought of. The principle is straightforward: attempt supplementing a partial solution to try building a solution progressively. Using a straightforward criterion, choose the "best"

augmentation for each iteration. Because choosing an augmentation that minimizes the ratio of "cost" to "advantage" is the most typical criterion, the term "greedy" is used to describe this behavior. We call an augmentation's cost-to-advantage ratio its "augmentation density." [39] Greedy algorithms have been developed to address a wide range of combinatorial optimization challenges. Many greedy algorithms have produced beautiful worst-case analysis conclusions. Greedy algorithms are simple to understand and code, with rapid execution speeds. In contrast, the accuracy of greedy algorithms is frequently poor. [40] The set covering problem (SCP) is NP-hard and requires heuristics to solve on large databases. Precise methods are studied, which build the cover step by step. The first sentence is selected based on a criterion, such as the number of units to cover. The sentence is added to the cover, and covered units are removed. The process starts again with the second sentence, containing a maximum of non-already-covered units. The performance of this method depends on the choice of sentences and the criterion guiding their selection. [41]

Greedy Algorithms (GA)are important in computer science because of their simplicity and effectiveness in handling certain issues. They are very useful for dealing with localized characteristics, as shown in the context of computing eigenpairs of huge sparse matrices.[42], [43] Despite potential theoretical limitations, greedy algorithms often perform well in practice, showcasing sublinear regret bounds in certain bandit models and competitive performance in various decision-making scenarios.[44] Furthermore, greedy algorithms can beat sophisticated neural network optimizers in combinatorial optimization. For example, greedy algorithms can solve the maximal independent set issue more quickly and effectively than graph neural networks. [45] Furthermore, by repeatedly identifying signal support and updating recovered signals, greedy algorithms play a significant role in hybrid compressed sensing in compressed sensing applications, proving effectiveness in noisy situations.[46]

An essential need for a problem to be suitable for a greedy algorithm is that it has the greedy choice property. According to this feature, the globally optimum solution is reached at every stage of the algorithm by choosing the locally optimal option. Greedy algorithms are used to provide optimum solutions in the context of

decision trees and linear optimization based on various uncertainty metrics. [47] [48] Furthermore, the geodesic characteristic of a greedy algorithm is investigated in the context of jump systems and convex function minimization, emphasizing the significance of trajectory optimality towards the closest optimal solution.[48] [49]

The optimal substructure property is necessary for a greedy algorithm to be able to solve a given issue. This characteristic guarantees that the best solution to each of the problem's subproblems may be combined to provide the best solution to the overall issue. Greedy algorithms try to find the best instant answer without going back and reevaluating previous judgments by making decisions based on the information provided at each stage. Numerous research publications emphasize the idea of optimum substructure, highlighting the fact that even while greedy algorithms don't always produce globally optimal solutions, they will always produce an optimal solution when the issue can be converted into a matroid structure.[50]

Greedy Algorithms (GA) are heuristic approaches that search for the overall best solution by selecting the option that is locally optimal at each stage. It is frequently applied to a variety of problem-solving situations, such as figuring out the shortest path in a weighted network.[47] Without going back and reevaluating earlier decisions, the algorithm chooses the best option that is accessible at each level. This straightforward yet efficient approach works well in reality, particularly in some bandit models where it may obtain near-optimal regret bounds. However, it can occasionally result in poor solutions.[51] Greedy algorithms may be tailored to many scenarios, such as spline interpolation, where adaptive point selection reduces overfitting and mistakes.[45]

4.3.1.2 Applications

1. Charging allocation:

The greedy algorithm is extended to respect non-ideal charging by using indicators to check if it can be incremented. The algorithm works similarly to the general greedy algorithm for the backpack problem, sorting vehicles by charging time, and then attempting to allocate a minimum current of $x$min to all cars in order. The allocated limits are incremented uniformly in one-amp steps twice to account for symmetry.

The algorithm may produce sub-optimal results, but it is efficient and provides a symmetric and fair distribution of currents.[52]

2. Scheduling methods:

The Flow Shop Sequence-Dependent Group Scheduling Problem (FSDGSP) is a challenging scheduling problem in cellular manufacturing systems, attracting significant attention from academics and practitioners. Methods for solving it are divided into three categories: exact algorithms, constructive heuristics, and metaheuristic algorithms. Exact algorithms use a Lower bound (LB) method with a flow time criterion, while constructive heuristics construct solutions based on scheduling rules. Metaheuristic algorithms, on the other hand, use various search frameworks to solve the FSDGSP, such as self-adaptive genetic algorithms, mathematical methods, hybrid harmony search algorithms, and scatter search algorithms. These methods aim to minimize tardiness, mean total flow time, and multicriteria. [53]

4.3.2 Dynamic Programming (DL)

4.3.2.1 Introduction

The study of transformation in mathematics is a fundamental concept that leads to investigations of the effects of repetition of a transformation, which is the study of multistage processes. This source of analysis includes iteration, ergodic theory, semigroups, branching processes, random transformations, and deterministic transformations at stochastic times. The focus is on multistage decision processes, which involve allowing a choice of transformation to be employed at each time. This direction of research raises new questions and develops new techniques, while also making repeated contacts with classical areas of analysis. The fundamental "as if" property of mathematics allows for the interpretation of many processes as such. The term "mathematical theory of multistage decision processes" is shortened to "dynamic programming,".[54]

Dynamic programming works by breaking down an issue into a series of smaller problems so that we can quickly calculate a solution to the larger problem given a solution to the smaller problems and the smaller problems can be effectively addressed recursively in terms of each other. One significant feature of dynamic

programming is the frequent reuse of a single subproblem's solution to address many, more complex subproblems. Therefore, to prevent recalculating answers later in a dynamic programming technique, we must "cache" the results. This is the way that dynamic programming algorithms vary from traditional recursive techniques. [55]

Sequential Decision Processes of DP

The optimization problem of the form involves choosing a decision from a set of eligible decisions Δ, with H being the objective function. The optimum is determined by d∗, which is the value of d ∈ Δ for which H(d) has the optimal value. In sequential decision processes, the decisions may be made in a specified sequence, such as $(d_1, d_2, \dots, d_n)$, where the ordered set belongs to a decision space $\Delta = D_1 \times D_2 \times \dots \times D_N$.

The optimizations are performed inside-out, each depending on outer decisions. Efficiency may depend upon the order in which decisions are made. For a given sequencing of decisions, the optimal choice depends on the ultimate choices d_i^* that are made for subsequent decisions. An algorithm is optimal if the locally optimal solution of $\text{optd}_1\{H\hat{}\ (d_1)\}$ yields the globally optimal solution H∗.

If the objective function h is (strongly) separable, the cost Cn of making a decision dnmaybe a function not only of the decision itself but also of the state $(d_1, d_2, \dots, d_{n-1})$, in which the decision is made. If h is (weakly) separable, the locally optimal solution of optd_1 {H^(d1)} yields the globally optimal solution H∗.[56]

The foundation of dynamic programming is the idea of optimal substructure, which states that a problem's best solution may be built from the optimal solutions of its subproblems. This idea is essential to effectively resolving challenging optimization issues. The optimum substructure property, for example, is obvious when dealing with the shortest path problem as the shortest path between two nodes may be found by considering the shortest pathways between intermediate nodes. Numerous research publications on dynamic programming emphasize this idea and stress how applicable it is in a variety of situations, including discounted dynamic programming.[57]

The concept of optimality, which posits that the best way to solve a dynamic programming issue is to combine the best answers to its subproblems, is a cornerstone

of dynamic programming. Even though this approach works for a wide range of situations, measure-theoretic difficulties are avoided by teaching it just for issues with finite or countable state spaces. Since dynamic pricing and inventory management are examples of traditional models with continuous state spaces, they cannot be applied to them, and students may not be aware of the potential difficulties while learning dynamic programming models with general state spaces. To tackle this, we give circumstances and a self-contained elementary argument that defines when the discounted dynamic programming principle of optimality holds true. These circumstances provide insight into the potential challenges in the generic state space situation. We give examples from the literature where the principle of optimality holds, including the rather complex situation of universally measurable dynamic programming and the straightforward case of finite dynamic programming, where our primary finding may be implemented. [58]

Optimization difficulties are addressed by Dynamic Programming (DP), which divides them into overlapping subproblems, solves each subproblem only once, and stores the answers to prevent repeating computations. According to the DP principle of optimality, an optimal solution to a dynamic programming problem may be obtained by integrating optimal solutions to each of its subproblems. [59] [60] DP effectively handles a variety of issues by storing intermediate outcomes, such as those in multiagent systems with mean-field approximation and learning. These problems may be framed as reinforcement learning problems for increased efficiency and the creation of new algorithms.[61]

To create a Dynamic Programming (DP) solution, first define the issue and divide it into more manageable subproblems that have optimum substructures that overlap. Next, create a recursive connection to represent the bigger problem's optimal solution in terms of its subproblems' ideal solutions. To prevent repeating calculations, create a memorization table or bottom-up method to save and reuse the answers to subproblems. Once the primary issue has been addressed as ideally as possible, implement the DP solution by addressing the subproblems repeatedly. Lastly, evaluate the DP solution's temporal and spatial complexity to make sure the problem is solved effectively and efficiently.

One way to solve problems with dynamic programming is to create recursive relations and identify states. As the many configurations or circumstances of the issue being solved, states are the fundamental building blocks of dynamic programming. A recursive relation that explains the relationship between the best solution in each state and the optimal solutions of its subproblems may be formulated using these states, which are specified according to the constraints and features of the problem. The definition of the recursive relation, which allows for the iterative computing of solutions, is essential to dynamic programming. It explains how the best value at a given state is obtained from the optimal values of its previous stages. Dynamic programming effectively determines solutions by decomposing the main problem into smaller problems and leveraging states and recursive connections to construct the links between them. [62]

4.3.2.2 Applications

1. Path optimization

The goal of path-finding issues is to locate the most effective paths via grids or graphs. The Floyd-Warshall method uses dynamic programming to determine the shortest pathways between every pair of nodes. Both techniques update distances in a systematic manner to guarantee optimum pathfinding. Dynamic programming is used to address longest route issues, which are frequently resolved in Directed Acyclic Graphs (DAGs) for critical path analysis and project scheduling. Urban planning, network routing, and navigation all make extensive use of these methods.

2. Game theory

In game theory, dynamic programming is used to solve sequential games. For example, the minimax method is used to solve two-player zero-sum games. The minimax method simulates every potential play to iteratively assess game trees and identify the best actions. Efficiency is increased by improvements like memorization and alpha-beta pruning, which save outcomes from subproblems and remove unnecessary branches. This method is frequently used in automated planning, economic models, and chess engines. Thus,

decision-making in intricate strategic relationships is optimized using dynamic programming.[59]

4.3.3 Linear Programming (LP)

4.3.3.1 Introduction

Linear programming (LP) is a prominent paradigm for making decisions and solving optimization issues. Taha (1997) defined Integer Programming (IP) issues as optimization problems that minimize or maximize the objective function while considering restrictions and integer variables. In general, integer programming applications may be used to accurately represent decision issues involving the efficient use of resources in engineering technology, corporate management, and a variety of other domains. According to Pawlak and Skowron (2007), for a vague idea rough (R), a lower approximation exists for all items that undoubtedly belong to the concept R, whereas an upper approximation exists for all objects that may belong to the concept R. In other words, the concept's lower approximation is the union of all the elementary concepts that are contained in it, while the higher approximation is the union of all the elementary concepts that have a non-empty intersection with the concept. Liu (2012) developed the idea of a rough variable, which is defined as a measurable function from a rough space to a set of real numbers. Garg (2015) also proposed an alternate method for tackling the multi-objective reliability optimization issue using uncertain, unclear, and imprecise data. Pandian et al. (2016) felt that all or some transportation issues' parameters were approximate integer intervals. The authors suggested a slice-sum technique to address the Rough Integer Interval Transportation Problem (RIITP), which involves rough integer intervals for transportation costs, supply, and demand. Rani et al. (2016) proposed a method for tackling multi-objective optimization problems from both optimistic and pessimistic perspectives. Garg and Arora (2018) created an approach to solve decision-making issues using partial ratings and preference information on qualities given assets. [63]

Linear programming (LP) formulations allow for the integration of instruction scheduling and register allocation into a single-issue description and solution. Furthermore, it is conceivable to get an ideal solution for the challenges at hand, but at a

cost. The expense of long computation times. We demonstrated that by utilizing Integer Linear Programming ILP-based approximations, calculation time may be lowered dramatically. The code quality is higher than that of traditional graph-based methods. Furthermore, integer linear programming allows for calculating lower limits on ideal schedule lengths. If an optimum solution is unavailable, the quality of an approximation solution can be assessed in this manner.

Basics of Integer Linear Programming (ILP)

In integer programming problems, an objective function is maximized or minimized while adhering to inequality, equality, and integrality constraints on some or all variables. Calculating an optimal solution for an integer linear program is NP-hard, however, there are numerous huge examples of such problems that can be solved. To achieve this, a systematic formulation is necessary, rather than an ad-hoc approach.[64]

There are three significant challenges that appear while addressing linear programming problems using the simplex approach.

1. Initialization

   Identifying a plausible initial solution is frequently easy for linear programming problems; in many cases, an all-zero solution is workable and can serve as the basis for a straightforward approach. Solving an auxiliary linear programming issue created by removing more variables from the original problem might yield a workable solution if the first one is not immediately apparent. Adjusting the cost function to minimize the total of these additional variables while keeping the original problem's limitations in mind. After solving the auxiliary problem, the values of the original variables indicate the first possible solution when all the new variables are zero. We next apply this first solution to the original linear programming issue. There is not a workable solution to the original problem if the auxiliary problem's answer does not produce a solution in which all of the additional variables are zero. The two-phase simplex approach is the name given to this tactic. The auxiliary problem is set up and solved in the first phase, and the main problem is solved in the second.
2. Iteration

The simplex method is a linear programming technique that involves selecting an entering or leaving variable for each iteration. There are often multiple candidates for these variables, but anyone who meets the requirements can be chosen. The only potential issue is un-roundedness, where the entering variable is not constrained by any of the candidate leaving variables, leading to an unbounded problem with no optimal solution. Another difficulty encountered during iteration is degeneracy, which occurs when one or more basic variables are zero. Degenerate solutions occur when a basic variable with zero value is replaced with an entering variable limited to zero value, resulting in a degenerate iteration. The method becomes "unstuck," and the iterations are non-degenerate. Degeneracy does not pose a threat to the simplex method, but it has consequences for termination. In this example, the next iteration is also degenerate, but the iteration after that is not and yields the optimal solution. Degeneracy does not endanger the simplex method, but it has consequences for termination.

3. Termination

Whether the issue is unbounded or impossible to solve, or if the best solution is found, the simplex technique ends. The simplex technique cycles, or it iterates through an infinite series of suboptimal solutions (all of which must have the same value for the cost function), as the only plausible alternative. Keep in mind that there can only be cycling when there is degeneracy since every cycle iteration needs to be degenerate. It is possible to prevent cycling by using a variety of tactics when choosing the entering, departing, or both variables. The most basic of these is known as the "smallest-subscript" rule, which stipulates that tied decisions about which variables to enter and which to leave are always resolved in favor of the variable with the smallest subscript.[65]

4.3.3.2 Applications

1. Logistics:

It investigates the feasibility of contract production planning in an enterprise using a linear programming method. It aims to address the issue of long-term delivery dates for automotive supply industry production companies due to insufficient production planning. The study proposes a production-to-stock

planning method to meet order demands on time and maximize profit for small-scale contract manufacturing companies. A model for contract manufacturing planning is developed, considering data and production constraints. an enterprise resource planning system. This is aimed at improving production planning in the automotive supply industry.[66]

2. Decision-making:

The optimization (maximization or minimization) of an objective function, which is a function of variables, is the focus of linear programming. A methodical process for selecting and justifying every decision made throughout the design and execution of the tactics that will be used to gather information and conduct analytical elements using the Revised Simplex approach. Constraints are a set of linear equalities and/or inequalities that apply to it. A mathematical approach known as "linear programming" deals with allocating scarce resources in the best feasible way based on a predetermined standard of optimality. The simplex technique, created by G. Dantzig in 1947 and made broadly accessible, is the most popular approach for resolving linear programming problems with any number of variables. [67]

4.3.4 Genetic Algorithms (GA)

4.3.4.1 Introduction

A quickly expanding topic of artificial intelligence is evolutionary algorithms, which include genetic algorithms (GAs). Darwin's idea of biological evolution serves as an inspiration for GAs. Through emulating this procedure, genetic algorithms can "evolve" answers to issues encountered in everyday life. The first to firmly establish computational evolution's theoretical foundation was Holland. Genetic algorithms and biological evolution provide the foundation of genetic algorithms or GAs. The design variables are represented by genes on a chromosome in this approach. A population, or set of potential people, is displayed on the response surface in GAs. Chromosomes with greater fitness are discovered by recombination, mutation, and environmental selection. The best chromosomes with higher fitness will inevitably arise due to natural selection.

The GA creates offspring (new chromosomes) with a high likelihood of being more fit than their parents by combining genes from two parent chromosomes using the recombination operator. However, mutation permits the creation of new regions to be investigated on the response surface. The fitness of the chromosomes can be improved by generations with GAs, and after several generations, chromosomes with the optimal variable settings will be produced. "John Holland" developed the genetic algorithm in the 1960s and 1970s at the University of Michigan with the help of his students and colleagues. Holland created the genetic algorithm in the 1960s. The genetic algorithms were introduced as an abstraction of biological evolution in Holland's 1975 book "Adaptation in Natural and Artificial Systems," which also provided a theoretical foundation for adaptation under the genetic algorithms. Holland's first objective was to formally research the phenomena of adaptation as it occurs in nature and to develop methods for importing the processes of natural adaptation into computer systems, rather than to create an algorithm to tackle issues. [68]

The basic concept of GA

The Genetic Algorithm (GA) approach is a method used to solve discrete or Combinatorial Optimization Problems (COPs). It allows the separation of the representation of the problem from the actual variables in which it was originally formulated. In this case, the vector x is represented by a string s, of length l, made up of symbols drawn from an alphabet A, using a mapping. c: A→ X. This is often designated as genotype-phenotype mapping.

The optimization problem becomes one of finding$\arg\min g_{s\epsilon f}$ where the function g(s) = f(c(s)). The bijection c is usually desirable, as it has an inverse, i.e., there is a unique vector x for every string s and a unique string s for every vector x. The GA approach was initially motivated by a biological analogy, such as the selective breeding of plants or animals. In GAs, a population of strings is used, often referred to in the GA literature as chromosomes. The recombination of strings is conducted using simple analogies of genetic crossover and mutation, and the search

is guided by the results of evaluating the objective function f for each string in the population.

De Jong's work established that population-based GAs using crossover and mutation operators could successfully deal with optimization problems of several diverse types. Crossover is a matter of replacing some of the genes in one parent with corresponding genes of the other. Mutation is another common operator, where a gene (or subset of genes) is chosen randomly, and the allele value of the chosen genes is changed.[69]

In combinatorial problems when the search space is of factorial order, GA is one of the most effective and widely used methods for locating the optimal or nearly optimal solution for challenging issues with a large search space particularity. Generating and manipulating several people using appropriate genetic operators is the main purpose of GA to get the answers. As a result, GA is categorized as one of the global search strategies that rely on the idea of gathering solutions rather than selecting one. In general, classical GA requires a significant amount of calculation time to find the best solution. Nonetheless, it may be fixed by using heuristics in a certain manner. Heuristic applications may shorten calculation times and enhance GA-evolved solutions.

1. Encoding: The proper encoding approach must be used to represent the people who are connected to the issue domain as genes with a certain length before beginning to solve the problem with GA. The encoding method chosen depends on the nature of the problem. Below, several encoding methods are shown.

- Binary encoding: every person is represented as a sequence of bits, either 0 or 1, where each bit corresponds to a chromosomal gene. For instance, binary encoding is used in the Knapsack problem:

Chromosome B: 111111100010110001011111

Chromosome A: 111100101100111011101101

Example of binary encoding

- Permutation encoding: each person is represented as a series of integers that indicates their place in the data. Ordering issues and the Traveling Salesperson Problem (TSP), for instance, make use of the permutation encoding technique:

  Chromosome A 1 3 5 6 2 4 7 8 9

  Chromosome B 6 5 8 7 2 2 1 3 9

  Example of permutation encoding

- Value encoding: with this kind of encoding, the person is represented as a string of values. Any character or real integer can be used for these values. For instance, the value encoding approach is used in neural network weight determination:

  Chromosome A DNA 1.2324 5.3243 0.4556 2.3293 2.4545

  Chromosome B ACDJEIFJDHDIERJFFDDFLFEGT

  Chromosome C (forward), (left), (right), (back), and (forward)

  Example of value encoding.

- Tree encoding: The chromosomes are arranged in a tree-like structure consisting of many objects, like commands in programming language.

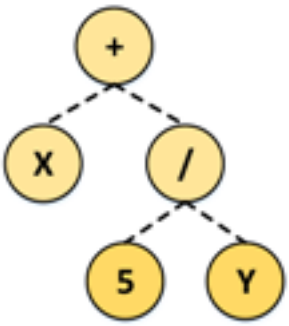


Figure 4 Chromosomes A

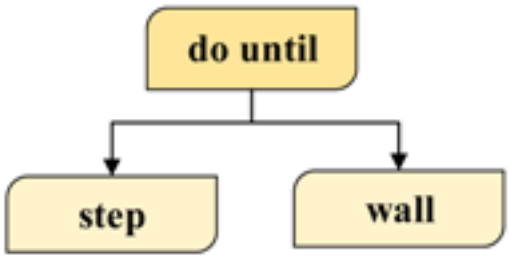


Figure 3 Chromosomes B

2. Fitness (evaluation): determines each member of the population's fitness value, f(x).
3. New Generation: starts with a blank population and proceeds to construct the new generation in stages 4 through 7.
4. Selection (Reproduction): The goal of the selection phase is to provide the most fit individuals of the population with more opportunities to procreate. Determining the best selection strategy is an essential step in this process. Numerous approaches for

selection are presented in the literature, such as Stochastic Universal Sampling (SUS), elitism, Rank, roulette wheel, tournament, and so on.

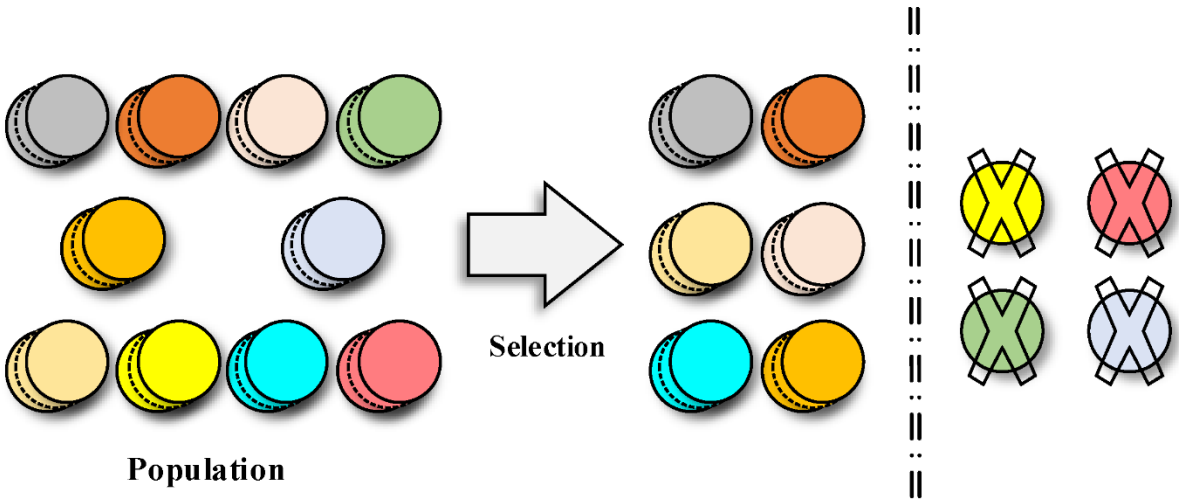


Figure 5 GA selection operation

For simplicity, we will now quickly discuss crossover and mutation using an example of binary encoding. Other encoding calls for other kinds of crossover and mutation.

5. Crossover (Recombination): this process transfers a portion of the parent Deoxyribonucleic Acid (DNA) from two people to a new child. Consequently, there is a possibility of having more fit progeny. GAs can be used with a variety of crossover operators. Let us start with single- and two-point crossovers and then use an additional strategy to meet certain scenarios.

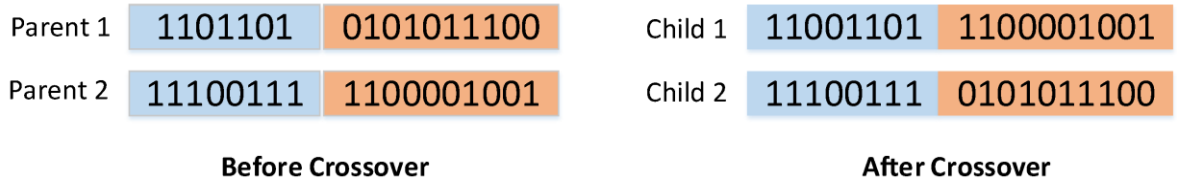


Figure 6 GA crossover operation

6. Mutation: The act of transferring or alternating certain genes inside a chromosome to produce new chromosomes that will serve as inputs for the following generation. To achieve a high positive degree of population variety, mutation seeks to find the finest practical solutions. It also helps prevent release in the local optimum. There are numerous ways to make mutations; the bit conversion mutation type is the first and progresses to other types that fit in various places.



Figure 7 GA crossover operation

7. Termination (stopping) criteria: The basic GA has been subjected to several terminating circumstances, including reaching the peak number of generations. Since the individuals' ability to provide updates has diminished, a low convergence diversity rate is anticipated. Increasing fitness yet staying below the cutoff point. GAs goes through several phases in their life cycle, including population seeding, selection, crossover, mutation, and halt constraint.

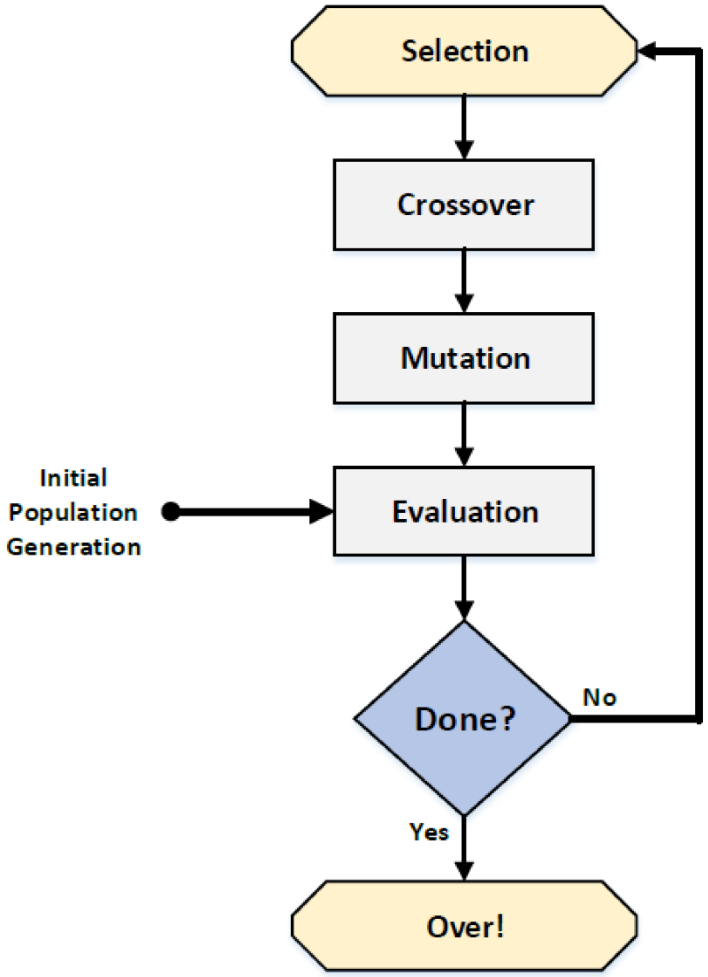


Figure 8 Flow chart of a typical GA

[70]

4.3.4.2 Applications

1. Reliability Optimization:

The likelihood that a system will function properly over a certain period and under given conditions is known as the system's dependability. Numerous systems are essential to many distinct functions, and their failure can have serious repercussions. fairly harsh. It is desirable to have reliability metrics for systems, such as communication switches, to assess existing dependability and identify potential improvement areas. In this area of study, optimization frequently entails determining how effectively to distribute redundant components across systems. To efficiently assess the dependability of the components, probabilities are assigned to them.

2. Job-Shop Scheduling:

Envision a manufacturing line with a series of machines, each one doing a single, little task. The labels on these machines range from 1 to m. Work must be done on machine 1, machine 2, etc., all the way up to machine m, for a single project to be finished. There is n in all. tasks that need to be completed, and each task calls for a specific amount of time on each machine (keep in mind that the time needed on a single computer may change depending on the task at hand). A machine can only be working on one task at a time, and once it gets going, it will not stop until the task is finished. The goal is to determine the best possible timetable. The objective is to find the ideal schedule so that the total time to complete all jobs is minimized. [71]

4.3.5 Random Search Algorithms

4.3.5.1 Introduction

Random search algorithms are methods that use randomness or probability, often in the form of a pseudo-random number generator. They are commonly used in continuous and discrete global optimization problems, particularly for ill-structured problems where the objective function may be nonconvex, nondifferentiable, and discontinuous over a continuous, discrete, or mixed continuous-discrete domain. These algorithms are useful for obtaining global optimal solutions, which can be difficult when there is no overriding structure that indicates whether a local solution

is indeed a global solution. Random search algorithms have been shown to solve large-scale problems efficiently in a way that is not possible for deterministic algorithms. While deterministic methods are known to be NP-hard, stochastic algorithms can be executed in polynomial time on average. For example, Dyer and Frieze showed that estimating the volume of a convex body takes an exponential number of function evaluations for any deterministic algorithm, but if one is willing to accept a weaker claim of being correct with an estimate that has a high probability of being correct, then a stochastic algorithm can provide such an estimate in polynomial time.

Random search methods are easy to implement on complex problems with "black box" function evaluations, as they rely on function evaluations rather than gradient and Hessian information. They can be coded quickly and applied to a broad class of global optimization problems. However, they are currently customized to each specific problem through trial and error. A common experience is that random search algorithms perform well and are "robust" in the sense that they give useful information quickly for ill-structured global optimization problems. Several types of categorizations have been suggested for random search algorithms, including two-phase methods, instance-based and model-based. Instance-based methods generate new candidate points based on the current point or population of points, while model-based methods rely on an explicit sampling distribution and update parameters of the probability distribution. Examples of instance-based algorithms include simulated annealing, genetic algorithms, tabu search, and ant colony optimization, while examples of model-based algorithms include stochastic gradient search and the cross-entropy method. Abstracting random search algorithms by a sequence of sampling distributions provides a means to understand their performance.

The general global optimization problem (P) is defined as

$$\text{(P) } min\ _{x \epsilon S} f(x)$$

f(x) where x is a vector of n decision variables, S is an n-dimensional feasible region,

and f is a real-valued function defined over S. The objective function may be an oracle or black-box function, requiring a numerical value of f for an input value of x. The feasible set S can be represented by a system of nonlinear equations or a membership function that returns whether a given x is in S. The global optimal solution to (P) is denoted by (x∗, y∗), where.

$$x* = argmin_{x \epsilon S} f(x) \text{ and } y^* = f(x*) = min_{x \epsilon S} f(x)$$

Random search algorithms are often applied to stochastic optimization problems, but in this article, they assume deterministic objectives and membership functions. To ensure a global optimum exists, regularity conditions must be assumed. If (P) is a continuous problem and the feasible set S is nonempty and compact, the Weierstrass theorem from classical analysis guarantees the existence of a global solution. If there are multiple optimal minima, let x∗ be an arbitrary fixed global minimum. A generic random search algorithm is described by a sequence of iterates X_K on iteration k = 0, 1, ..., which may depend on previous points and algorithmic parameters.

Algorithm for Generic Random Search

Step 0: Set up the iteration index k = 0, initial points $X_O \subset$ S, and algorithm parameters K=0.

Step 1: Create a list of potential points. $V_{K+1} \subset$ S in accordance with a certain generator as well as the corresponding sample distribution.

Step 2: Update X_{K+1}using the algorithmic parameters, past iterations, and candidate points V_{K+1}. Update the algorithm parameters θ_{K+1} as well.

Step 3: Stop when a requirement for halting is satisfied. If not, increase k and go back to Step 1. [72]

4.3.5.2 Applications

1. Aerospace Control

This is a promising area of research using randomized methods to describe changes in flight conditions, uncertainties in the aerodynamic model, and inaccuracies in geometric and inertial data. Due to the lack of analytic relationships between state space matrices and uncertain parameters, simulation-based methods

and lookup tables are often used. The first paper studied the impact of parameter variations on aircraft lateral directional stability using a randomized approach. Techniques based on Monte Carlo simulations have been successfully used for computing the probability of instability and related performance criteria. Other research focuses on unmanned aerial vehicles, unmanned aerial vehicles for surveillance and fire detection missions, and Linear Parameter-Varying (LPV) methods for F-16 aircraft control. Monte Carlo methods have been applied for conflict resolution in air traffic control, with a scenario approach developed for the controller design of a multivariable LV100 gas turbine engine.

2. Multi-agent Systems and PageRank Computation

Distributed control over networks focuses on traffic management of multiple vehicle systems, such as autonomous vehicles. Multi-agent systems offer advantages such as speedup in task execution and robustness against failure. However, a multi-agent approach is conceptually and algorithmically more involved, particularly in dealing with distributed information, communication protocols, verification and validation of decentralized control laws, and coordination between agents. Several studies have explored decentralized approaches for studying collision avoidance of large numbers of autonomous vehicles in uncertain environments, distributed abstract optimization via constraint consensus, partitioning, and coverage control for gossiping robots, quantized consensus and averaging, and distributed filtering with H∞ consensus of estimates subject to Linear Matrix Inequality (LMI) constraints. Another problem deals with ranking web pages for search engines like Google or Yahoo!, which requires the computation of the eigenvector (PageRank) corresponding to the largest eigenvalue of a positive stochastic matrix representing the link structure of the web. A decentralized randomized algorithm has been proposed to facilitate the computation of PageRank. Recent contributions include a technique for aggregating web pages into groups and a general class of PageRank optimization problems involving finding an optimal outline strategy for a website subject to design constraints.[73]

4.3.6 Particle Swarm Optimization (PSO)

4.3.6.1 Introduction

Eberhart and Kennedy (1995) and Kennedy and Eberhart (1995) introduced the Particle Swarm Optimization (PSO) algorithm, a stochastic optimization method based on swarm. 1995. Fish, birds, insects, and herds are among the animals whose social behavior is simulated by the PSO algorithm. Each member of the swarm continuously modifies the search pattern in response to its own and other members' learning experiences. These swarms adhere to a cooperative method of finding food. The primary concept behind the PSO algorithm's construction is intimately linked to two studies: First is the evolutionary algorithm; similarly to evolutionary algorithms, PSO searches a sizable part of the optimized objective function's solution space concurrently by utilizing a swarm mode. The other is artificial life, which focuses on the study of artificial systems that have life characteristics.

Following his research on social animal behavior and the artificial life theory, Milanas presented five fundamental principles (van den Bergh 2001) for building computer-generated swarm artificial life systems with cooperative behavior.
(1) Proximity: the swarm ought to be capable of performing basic time and space calculations.
(2) Quality: The swarm must be able to detect changes in the environment's quality and react accordingly.
(3) Diverse response: The swarm should not restrict how it obtains resources to a small area.
(4) Stability: The swarm's behavior mode should not alter in response to alterations in the surrounding environment.
(5) Adaptability: the swarm ought to modify its manner of behavior when it is appropriate for this modification to occur.

The fifth and fourth principles are the opposing sides of the same coin, as you may have seen. These five guiding concepts include the essential traits of artificial life systems, and they have become the tenets upon which the swarm artificial life system is built. Particles in PSO can adjust their locations and velocities in response to changes in the surrounding environment if they satisfy the quality and proximity

requirements. Furthermore, with PSO, the swarm does not restrict its mobility; instead, it keeps looking for the best answer throughout the whole solution space. Particles in PSO can maintain a steady motion inside the search space, while also modifying their movement patterns in response to environmental changes. Particle swarm systems therefore adhere to the five criteria. [74]

Particle swarm optimization (PSO) is a technique that simulates colonial behavior in collective animals, including insects, cattle, birds, and fish. The early model, the Boid model, was designed to simulate bird behavior, with each bird having an initial velocity and position. The algorithm was later defined by Russell and Kennedy, who reduced everyone to a particle with no mass or volume, only velocity, and location. The swarm follows three major rules: avoid collisions with nearby entities, match its own velocity with the vicinity, and align with the folk's epicenter.

The PSO algorithm is a swarm-based method of searching, where each component is represented as a reasonable solution to the optimization problem in a D-dimensional examination space. It can memorize both the multitude's ideal position and its own, in addition to the velocity. Particle data are combined in each iteration to change each dimension's velocity, which is then used to determine the particle's new position. Particles in the multi-dimensional search space continually alter their states until they attain a balance or ideal state or exceed the calculating boundaries. The goal functions to introduce the unique relationship between several aspects of the issue space, and empirical evidence has proven this method to be an efficient optimization technique.[75]

4.3.6.2 Applications

1. Engineering – civil engineering

Particle swarm optimization (PSO) algorithm with inertia weight factor for structural damage detection in civil engineering construction management. The algorithm has unique technical advantages and is easier to fall into local minimums than traditional methods. The improved algorithm has more advantages than the original standard algorithm. It suggests that the grid model should be dynamically assessed under real damage conditions to clarify modal information. Dynamic tests on the foundation model should be conducted to obtain structural vibration frequency and formation for

comparison. The PSO algorithm should be used to optimize the judgment, focusing on improving existing model problems. Each model modification will obtain a "pseudo condition," accurately identifying the modal information. The PSO algorithm should be used in the damage identification of civil engineering structures to ensure that the modified structure's modal information is consistent with the damage condition. The improved PSO algorithm should be used in the overall operation to ensure consistency in the modified structure's stiffness reduction coefficient. [76]

2. Prediction analysis for time series

A time series is a sequentially measured and ordered set of observations that are uniformly spaced at regular periods of time. A sequence of past values is used in time series prediction to create a model that predicts future values. PSO was integrated with regression analysis and Radial Basis Function (RBF) neural networks, among other methods. To achieve the time series prediction in a workable power system, Lee combined RBF neural networks with a nonlinear time-varying evolution PSO algorithm. To model and forecast the permeability of hydrocarbon reservoirs, Akande suggested a hybrid PSO and support vector regression model. For the first time, while simulating the hydrocarbon reservoir, PSO is examined to determine the best choice of Support Vector Regression (SVR) hyper-parameters. To tackle the power dispatch problem, Zou integrated PSO with least square support vector regression for short-term load forecasting in the power system.[77]

4.4 Conclusion

Every optimization method has certain advantages that fit all kinds of problems. For structured issues with distinct subproblems, greedy algorithms, and dynamic programming work especially well, providing the best and most efficient solutions for problems such as minimal spanning trees and shortest pathways. They are perfect in situations when prompt, best-practice choices are required because of their simplicity and quickness.

When it comes to global optimization challenges, particle swarm optimization and genetic algorithms excel. They are appropriate for issues like traveling salesman, function optimization, and neural network training because of their extreme versatility and ability to traverse intricate, multimodal environments in search of global optima. Their resilience for a

variety of applications is improved by their capacity to escape local optima through stochastic processes.

For problems with linear constraints and objectives, linear programming is the recommended approach. When it comes to offering accurate, globally ideal solutions, it is dependable and efficient in handling complex industrial challenges like production scheduling, resource allocation, and transportation logistics.

Simulated annealing is one of the simple and straightforward random search techniques. They may be used as a first investigation or baseline and are helpful for examining poorly understood search fields. They are frequently less effective, though, and need a lot of assessments to get decent results.

The choice of optimization technique depends on the specific problem characteristics, available computational resources, and the desired solution quality. Hybrid approaches, which combine different techniques, often yield superior performance for complex, real-world problems by leveraging the strengths of each method. This can lead to more robust, efficient, and high-quality solutions, making hybrid methods a powerful tool in the optimization toolbox.

Chapter 5: Conception and Implementation

5.1 Introduction

The Under Set Cover Problem (USCP) is a classical question in combinatorial optimization and computer science. It is a fundamental problem with broad applications in various fields. The problem is formally defined as follows: given a universe U of elements and a collection S of subsets of U, the objective is to find the smallest sub-collection of S whose union covers all elements in U.

Mathematically, the Set Cover Problem can be expressed as:

- Input: A finite set $U = \{e_1, e_2, \dots, e_n\}$ a collection of subsets $S = \{s_1, s_2, \dots, s_n\}$ where $S_i \subseteq U$
- Output: A minimum-size sub-collection $C \subseteq U$ such that $\cup_{C \epsilon C} C = U$.

The USCP is known to be NP-hard, implying that no polynomial-time algorithm is likely to exist for solving all instances of this problem optimally. As a result, a significant amount of research has focused on developing approximation algorithms and heuristics to find near-optimal solutions efficiently.

Given its widespread applications, solving the Set Cover Problem efficiently is crucial in practical settings.

5.2 Implementation Strategies

5.2.1 Materials Used

Google Colab was used as the software. Python was used as the programming language, and the `random` library was used for random number generation-related functions and Matplotlib for plotting

.

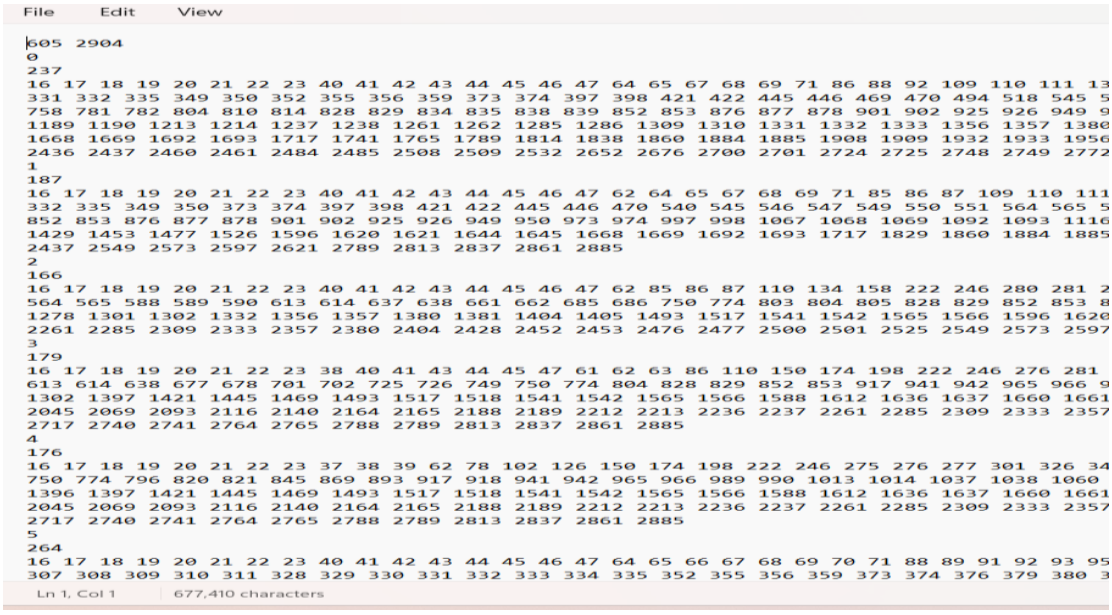


Figure 9 Screenshot from PC with AC_01_cover.txt file

Several strategies have been developed to tackle the Set Cover Problem; we used three heuristic techniques:

```
# Instance file example `AC_00_cover.txt`
10 7 # 10 samples and 7 candidates
0 # First sample
4 # 4 candidates cover sample 0
0 1 2 3 # the subset of these 4 candidates
1 # Second sample
3 # 3 candidates cover sample 0
0 1 4 # the subset of these 3 candidates
#...
```

This function reads a filename instance `AC_01_cover.txt` and returns the number of samples, the number of candidates, and the set of all subsets

```
# Read Instances file function
def read_file(filename):
  with open(filename) as fin:
        samples,candidates = map(int,fin.readline().split(' '))
        #print(f'{s = },{c = }')
        Js = []
        next(fin)
```

```
        next(fin)
        for j in range(samples):
            line = fin.readline().strip()
            Js.append(list(map(int,line.split(' '))))
            fin.readline()
            fin.readline()
 return samples,candidates,Js
```

this function returns the Is set from the Js set

```
# Is set former function
def Is_form(Js):
    Is = []
    for i in range(c):
      a = []
      for j,el in enumerate(Js):
        if i in el:
          a.append(j)
          continue
      Is.append(a)
    return Is
```

a. Greedy Algorithms

The greedy function we defined takes the `Is` set, which is the set of samples covered by each candidate, formed by the `Is from (Js)` function from the set of subsets `Js`; this algorithm simply sorts the `Is` set from the highest to lowest number of covered samples, removes each covered sample, and moves to the next candidates in the `Is` set until there are no samples left to cover.

```
def greedy_fit(s,Is,order):
    sol = []
    l = set(range(s))
    for i in order:
        if l:
            sol.append(i)
            l -= set(Is[i])
        else:
            break
    return sol
```

```
# Read the instance file and form the Is subset
filename = '/content/AC_01_cover.txt'
s,c,Js = read_file()
Is = Is_form(Js)
```

Run to get the result with runtime

```
    # import time libray for runtime calculation
    import time
# Run the greedy fit function
order = [i for i in range(c)]
order = sorted(order,key=lambda x:len(Is[x]),reverse=True) # sort Is
start = time.time()
solution = greedy_fit(s,Is,order)
end = time.time()
print(solution)
print(len(solution))
print(f'runtime of greedy fit approach: {end - start :.3f}')
```

Selected candidates: [1, 247, 2643, 2885, 128, 1330, 1574, 2772, 104, 152, 1066, 1310, 1594]

Number of candidates selected: 13

Runtime of greedy fit approach: 0.000

b. Random search algorithms

The function calculates the solution by summing the selected subsets and checking if the solution covers all `s` elements, returning if all `s` elements are covered and infinite otherwise.

```
import random

def evaluate_solution(s, Is, solution, costs):
    covered = set()
    total_cost = 0
    for i in solution:
        covered.update(Is[i])
        total_cost += costs[i]
    if len(covered) == s:
        return total_cost
    return float('inf')  # Return an infinite cost if not all elements are covered
```

This function runs for many trials (1000) and returns the best solution found.

```
def random_search(s, Is, costs, trials=1000):
    best_solution = None
    best_cost = float('inf')

    for _ in range(trials):
        solution = []
```

```
    uncovered = set(range(s))
    indices = list(range(len(Is)))
    random.shuffle(indices)

    for i in indices:
        if uncovered:
            solution.append(i)
            uncovered -= set(Is[i])
        else:
            break

    if not uncovered:  # Check if all elements are covered
        current_cost = evaluate_solution(s, Is, solution, costs)
        if current_cost < best_cost:
            best_cost = current_cost
            best_solution = solution

  if best_solution is not None:
    best_solution.sort()  # Sort the best solution in ascending order
  return best_solution, best_cost
```

Run to get the result with runtime for 10 different attempts(trials)

```
# Run the random search
# Assuming costs are provided as a separate list or can be inferred,
here's an example cost list
costs = [1] * len(Is)  # Example costs, replace with actual costs if
provided
rs_runtime = []
rs_solutions = []
for i in range(10):
  print('Trial Number ', i+1)
  start = time.time()
  best_solution, best_cost = random_search(s, Is, costs, trials=1000)
# call the random search function
  end = time.time()
  print("Selected subset indices:", best_solution)
  print("Number of subsets selected:", len(best_solution))
  print("Total cost of selected subsets:", best_cost)
  print(f'runtime of greedy fit approach: {end - start :.3f}')
  rs_runtime.append(end - start)
  rs_solutions.append(len(best_solution))
```

Trial Number 1

Number of candidates selected: 33

Runtime of greedy fit approach: 1.994

Trial Number 2

Number of candidates selected: 31

Runtime of greedy fit approach: 2.612

……

Trial Number 10

Number of candidates selected: 32

Runtime of greedy fit approach: 1.954

c. Genetic Algorithms

The components initialize a population of chromosomes representing subsets of Is, evaluate their fitness based on how well they cover the target set s and their associated costs, select parents based on fitness, perform crossover and mutation on the parents to create a new population, and iterate through generations to find the optimal subset of Is that covers all elements in Js with minimal cost

```
# Genetic Algorithm Components
def initialize_population(pop_size, chromosome_length):
    return [[random.randint(0, 1) for _ in range(chromosome_length)] for _ in
range(pop_size)]

def fitness_function(chromosome, s, Is, costs):
    solution = [i for i in range(len(chromosome)) if chromosome[i] == 1]
    covered = set()
    for i in solution:
        covered.update(Is[i])
    if len(covered) == s:
        return evaluate_solution(s, Is, solution, costs)
    else:
        return 1000000000  # Return a large but finite cost for solutions that do not cover all
elements

def selection(population, fitnesses):
    total_fitness = sum(fitnesses)
    selection_probs = [f / total_fitness for f in fitnesses]
```

```
    return population[random.choices(range(len(population)), weights=selection_probs, k=1)[0]]

def crossover(parent1, parent2, crossover_rate=0.8):
    if random.random() < crossover_rate:
        point = random.randint(1, len(parent1) - 1)
        return parent1[:point] + parent2[point:], parent2[:point] + parent1[point:]
    return parent1, parent2

def mutate(chromosome, mutation_rate=0.01):
    return [gene if random.random() > mutation_rate else 1 - gene for gene in chromosome]

def genetic_algorithm(s, Is, costs, pop_size=100, generations=100, crossover_rate=0.8, mutation_rate=0.01):
    chromosome_length = len(Is)
    population = initialize_population(pop_size, chromosome_length)
    best_solution = None
    best_cost = float('inf')

    for generation in range(generations):
        fitnesses = [fitness_function(chromosome, s, Is, costs) for chromosome in population]
        new_population = []
        for _ in range(pop_size // 2):
            parent1 = selection(population, fitnesses)
            parent2 = selection(population, fitnesses)
            child1, child2 = crossover(parent1, parent2, crossover_rate)
            child1 = mutate(child1, mutation_rate)
            new_population.extend([child1, child2])

        population = new_population

    for chromosome in population:
        cost = fitness_function(chromosome, s, Is, costs)
        if cost < best_cost:
            best_cost = cost
            best_solution = chromosome

    return [i for i in range(len(best_solution)) if best_solution[i] == 1], best_cost
```

Run to get the result with runtime for 10 different attempts (trials)

```
# Run the Genetic Algorithm
costs = [1] * len(Is)
ga_runtime = []
ga_solutions = []
```

```
for i in range(10):
  print('Trial Number ', i+1)
  start = time.time()
  best_solution, best_cost = genetic_algorithm(s, Is, costs,
pop_size=100, generations=100, crossover_rate=0.8, mutation_rate=0.01)
  end = time.time()
  print("Selected subset indices:", best_solution)
  print("Number of subsets selected:", len(best_solution))
  print("Total cost of selected subsets:", best_cost)
  print(f'runtime of greedy fit approach: {end - start :.3f}')
  ga_runtime.append(end - start)
  ga_solutions.append(len(best_solution))
```

Trial Number 1

Number of candidates selected: 1415

Runtime of greedy fit approach: 43.859

Trial Number 2

Number of candidates selected: 1411

Runtime of greedy fit approach: 41.234

……

Trial Number 10

Number of candidates selected: 1449

Runtime of greedy fit approach:41.138

5.3 Conclusion

In this thesis, we have explored the Set Cover Problem (SCP) using three distinct techniques: greedy algorithms, genetic algorithms, and random search. Each method offers unique strengths and trade-offs, contributing valuable insights into the practical resolution of this NP-hard problem

Greedy algorithm - The most used approximation algorithm for SCP. The greedy approach iteratively selects the subset covering the largest number of uncovered elements. Although it does not guarantee an optimal solution, it provides a logarithmic approximation ratio, which is often sufficient in practice

Random search - Designed to solve problems faster when classic methods are moderately slow or to find an approximate solution when classic methods fail to see any exact solution. They often involve making educated guesses, simplifying the problem, or using trial and error to explore the solution space.

Genetic algorithm - Employing evolutionary strategies to explore a broad search space and identify optimal solutions.

Chapter 6: Result and Discussion

6.1 Introduction

This chapter presents the results and comparative analysis of three heuristic techniques—greedy algorithm, genetic algorithm, and random search—applied to the Set Cover Problem (SCP). The goal is to evaluate and compare the effectiveness, efficiency, and practicality of each method in solving SCP instances. We start by outlining the experimental setup, including the datasets, parameter settings, and evaluation metrics. The subsequent sections provide a detailed discussion of the performance of each algorithm, followed by a comprehensive comparison to highlight their strengths and weaknesses.

6.2 Experimental Setup

The experiments were conducted using a standard SCP dataset, which includes a predefined universe of elements and a collection of subsets. Each subset is associated with a cost. The following metrics were used to evaluate the algorithms:

- Solution Quality: Measured by the total cost of the selected subsets.
- Computational Time: Time taken to reach the solution.

Parameters for the genetic algorithm were set based on preliminary tuning, with a population size of 100, 100 generations, a crossover rate of 0.8, and a mutation rate of 0.01. The random search was performed over 1000 trials to ensure a thorough exploration of the solution space. The genetic and random include 10 attempts (trials).

6.3 Results and Discussion

Greedy Algorithm

The greedy algorithm iteratively selects the subset that provides the most cost-efficient coverage of the uncovered elements. This method is computationally efficient and simple to implement.

- Performance: The greedy algorithm quickly produced solutions that were near-optimal, covering the entire universe with a relatively low total cost.
- Computational Time: It consistently requires less computational time compared to the other methods due to its straightforward approach.

Example Result:

- Selected subset: [1, 247, 2643, 2885, 128, 1330, 1574, 2772, 104, 152, 1066, 1310, 1594]
- Number of subsets selected: 13
- Runtime of greedy fit approach: 0.000

Random Search

Random search involves randomly selecting subsets and evaluating their coverage and cost, repeated over many trials to find a satisfactory solution.

- Performance: Achieved an average solution, with some trials yielding near-optimal results and others falling short.
- Computational Time: It was generally faster than the genetic algorithm but less efficient than the greedy algorithm due to the high number of iterations required.

Example Result:

Solution	33	31	32	32	30	29	27	33	23	32
Runtime(s)	1.994	2.612	3.014	1.942	1.952	1.975	1.937	2.284	3.394	1.954

Table 2 Result of Random Search Solutions and Runtimes

The results data represents random search solutions and their corresponding runtimes in seconds. The solutions range from 23 to 33, with runtimes varying between 1.937 and 3.394 seconds.

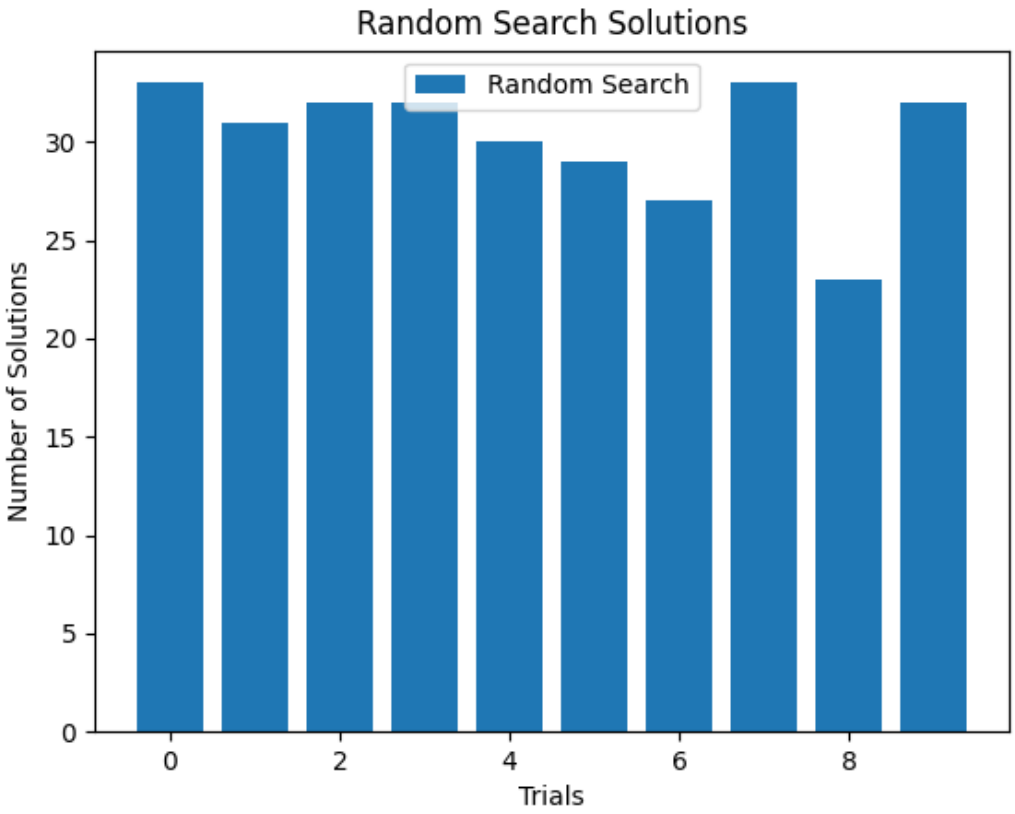


Figure 10 Random search solutions for 10 attempts

Genetic Algorithm

The genetic algorithm leverages evolutionary principles to explore the solution space, balancing exploration and exploitation through selection, crossover, and mutation operations.

- Performance: It demonstrated the ability to find high-quality solutions (Achieved a very High solution), but in this case, it didn't outperform the greedy algorithm and Random search in terms of the selected subsets.
- Computational Time: It required significantly more time due to the iterative nature of population evolution and the need for multiple fitness evaluations.

Example Result:

Solution	1415	1411	1469	1404	1413	1417	1453	1405	1410	1449
Runtime(s)	43.859	41.234	42.44	42.145	41.871	42.685	41.775	41.198	40.631	41.138

Table 3 Result of Genetic Algorithm Solutions and Runtimes

The results data represents a genetic algorithm's solutions and corresponding runtimes in seconds. The solutions range from 1404 to 1469, with runtimes varying between 40.631 and 43.859 seconds.

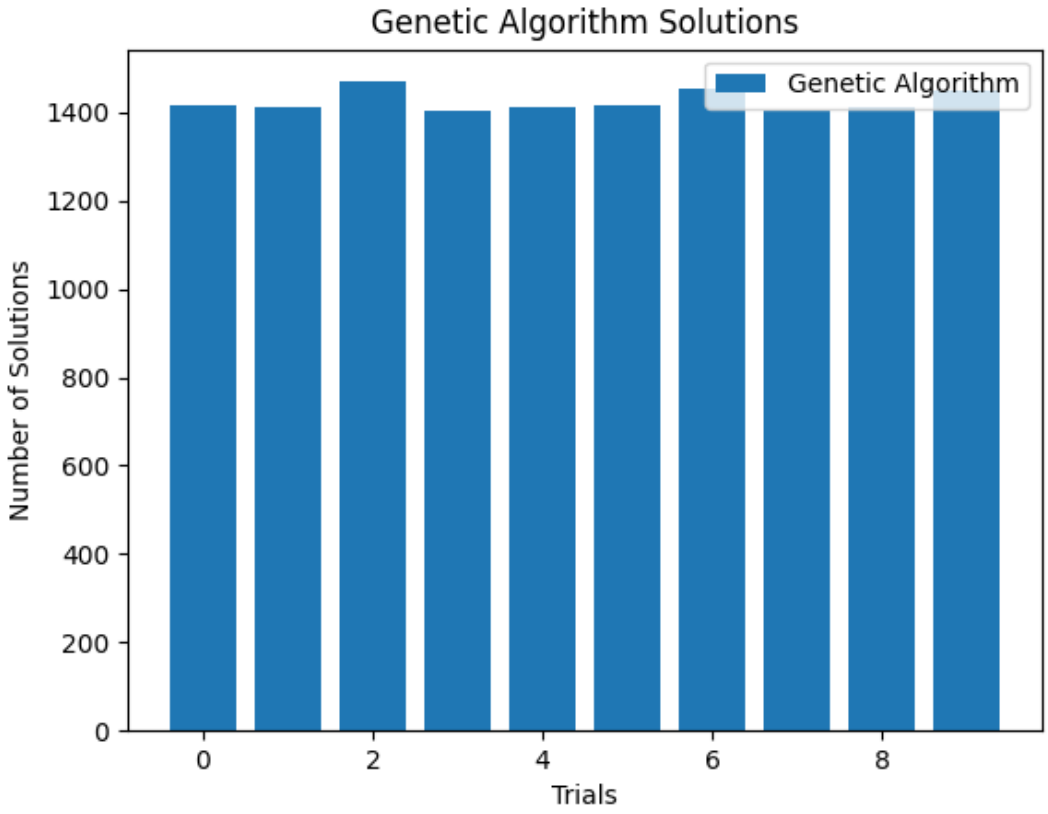


Figure 11 Genetic Algorithm solutions for 10 attempts

6.4 Comparative Analysis

The comparative analysis highlights the distinctive advantages and limitations of each algorithm:

- Greedy Algorithm: Best suited for quick, near-optimal solutions with low computational overhead. It is particularly effective for large-scale problems where a reasonably good solution is required swiftly.
- Genetic Algorithm: Achieved very high solutions, with high computational time. Thus, given the highest subset in the solution.
- Random Search: Serves as a useful baseline and can complement other methods by preventing local optima. It is beneficial in situations where simplicity and broad exploration are prioritized.

Approach	Solution	Runtime(s)
Greedy Algorithm	**13**	**0.000**
Random Search	Min 23, Max 33	Min 1.937, Max 3.394
Genetic Algorithm	Min 1405, Max 1469	Min 40.631, Max 43.859

Table 4 Comparison of Greedy, Genetic Algorithm, and Random Search

Table 4 compares three approaches: the Greedy Algorithm, Random Search, and Genetic Algorithm. The Greedy Algorithm found a solution of 13 in 0.000 seconds, Random Search solutions ranged from 23 to 33 with runtimes between 1.937 and 3.394 seconds, and Genetic Algorithm solutions ranged from 1405 to 1469 with runtimes between 40.631 and 43.859 seconds.

s/n	RS solution	GA solution	RS Runtime (s)	GA Runtime (s)
1	33	1414	1.994	43.859
2	31	1411	2.612	41.234
3	32	1469	3.014	42.444
4	32	1404	1.942	42.145
5	30	1413	1.952	41.871
6	29	1417	1.975	42.685
7	27	1453	1.937	41.775
8	33	1405	2.284	41.198
9	23	1410	3.394	40.631
10	32	1449	1.954	41.138

Table 5 Comparison of 10 attempts for Genetic Algorithm, and Random Search

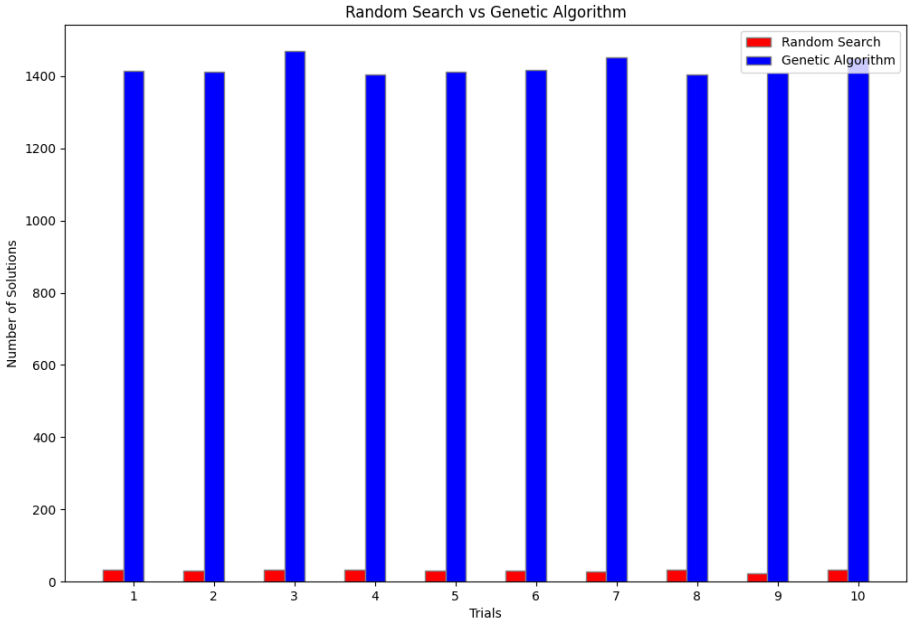


Figure 12 Bar chart comparison of solutions for Random Search vs Genetic Algorithm

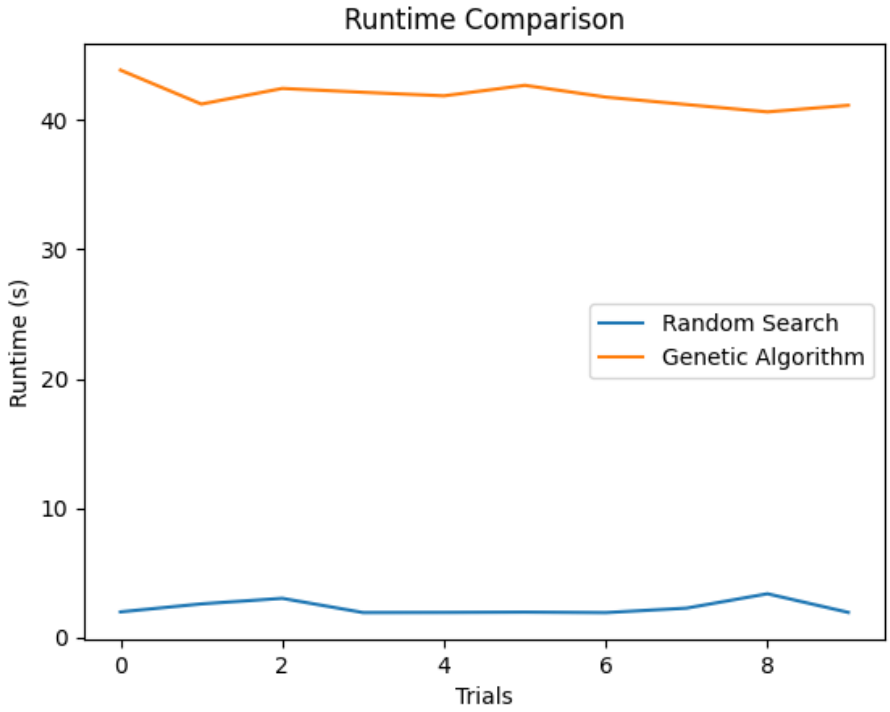


Figure 13 Graphical comparison of Runtime (s) for Random search vs Genetic Algorithm

6.5 Conclusion

The results demonstrate that while the greedy algorithm is highly efficient and often adequate, genetic algorithms did achieve high solutions with high computational time. Random search, though less consistent, offers a straightforward and flexible approach that can yield good results with sufficient iterations.

Each heuristic technique brings unique strengths to solving the Set Cover Problem. The choice of method depends on the specific requirements of the problem instance, such as the need for speed, solution quality, or computational resources.

Chapter 7: General Conclusion and Perspectives

7.1 General Conclusion

The Set Cover Problem (SCP) is a fundamental issue in computer science and mathematics, classified as NP-hard due to its computational complexity. Despite the inherent difficulty in finding optimal solutions, the SCP has numerous practical applications in fields such as resource allocation, facility location, and various optimization tasks in operational research and artificial intelligence. The importance of SCP stems from its ability to model a wide range of real-world problems, making it a critical area of study.

Throughout this thesis, we explored various approaches to solving the SCP, including classical algorithms like the Greedy Algorithm, advanced methods such as Genetic Algorithms, and simple but effective techniques like Random Search. Each method brings its unique advantages and limitations to the table:

- Greedy Algorithm: This method is highly efficient in terms of computational time and often produces reasonably good solutions. It is particularly useful for large-scale problems where a quick, near-optimal solution is sufficient.
- Random Search: Serving as a baseline, this method offers simplicity and broad exploration, preventing the algorithm from getting stuck in local optima.
- Genetic Algorithm: Known for its robustness in exploring the solution space, this algorithm is complex, and costs computational time.

The comparative analysis of these algorithms demonstrated that while the Greedy Algorithm provides quick and often adequate solutions, the Genetic Algorithm did achieve large solutions and took more computational resources. The Random Search offers a flexible approach that can yield good results with sufficient iterations.

7.2 Perspectives

The quest for efficient solutions to the Set Cover Problem remains an ongoing challenge, driven by the need to optimize resource utilization and enhance system performance across various domains. Several perspectives can be drawn for future research and advancements in this field:

1. Advancements in Heuristics and Metaheuristics: Continued development and refinement of heuristic methods, including hybrid approaches that combine the strengths of different algorithms, could lead to more efficient and effective solutions. Techniques like Simulated Annealing, Tabu Search, and Ant Colony Optimization hold promise for improving the solution quality and computational efficiency.
2. Application of Machine Learning: Integrating machine learning techniques can offer new ways to tackle SCP by learning from previous solutions and predicting better starting points for heuristic algorithms. Reinforcement learning could be used to dynamically adjust strategies based on real-time performance feedback.
3. Quantum Computing: Emerging technologies such as quantum computing present exciting opportunities for solving NP-hard problems like SCP. Quantum algorithms, such as Quantum Annealing, have the potential to outperform classical methods by exploiting quantum parallelism and tunneling effects.
4. Approximation Algorithms: Developing new approximation algorithms that guarantee solutions within a certain factor of the optimal can provide practical solutions for large-scale instances where exact solutions are computationally infeasible. Theoretical advancements in this area can significantly impact practical applications.
5. Interdisciplinary Approaches: Leveraging insights from other fields, such as biology (e.g., genetic algorithms) and physics (e.g., simulated annealing), can inspire novel algorithms and solution methods. Collaborative research that integrates knowledge from diverse disciplines can lead to innovative approaches to SCP.
6. Real-World Applications and Case Studies: Applying these algorithms to a broader range of real-world problems can provide valuable feedback and drive further improvements. Case studies in various sectors, such as logistics, telecommunications, and healthcare, can demonstrate the practical utility and identify areas for enhancement.
7. Enhanced Computational Techniques: Utilizing advanced computational techniques, such as parallel processing and distributed computing, can significantly reduce the time required to solve large instances of SCP. Implementing these techniques in existing algorithms can enhance their scalability and efficiency.

7.3 Conclusion

In conclusion, while the Set Cover Problem presents significant challenges due to its computational complexity, ongoing research, and technological advancements offer promising avenues for developing more efficient and effective solutions. The interplay between theory and practice will continue to drive progress in this field, emphasizing the importance of balancing computational complexity with solution quality. Researchers and practitioners must remain adaptive, continually refining algorithms and exploring new methodologies to meet the evolving demands of real-world applications.

References

Reference

[1] A. Mishra, N. T. D. Linh, M. Bhardwaj, and C. M. A. Pinto, *Multi-Criteria Decision Models in Software Reliability: Methods and Applications*. CRC Press, 2022.

[2] T. Ito *et al.*, "A polynomial-time approximation scheme for the geometric unique coverage problem on unit squares," *Computational Geometry*, vol. 51, pp. 25–39, Jan. 2016, doi: 10.1016/j.comgeo.2015.10.004.

[3] B. Crawford, R. Soto, E. Monfroy, G. Astorga, J. García, and E. Cortes, "A Meta-Optimization Approach to Solve the Set Covering Problem," *Ingeniería*, vol. 23, no. 3, pp. 274–288, 2018.

[4] P. Surendiran *et al.*, "Solving Exact Cover Instances with Molecular-Motor-Powered Network-Based Biocomputation," *ACS Nanosci. Au*, vol. 2, no. 5, pp. 396–403, Oct. 2022, doi: 10.1021/acsnanoscienceau.2c00013.

[5] E. Benoist, G. Fertin, and G. Jean, "The Exact Subset MultiCover Problem," in *Theory and Applications of Models of Computation*, D.-Z. Du, D. Du, C. Wu, and D. Xu, Eds., Cham: Springer International Publishing, 2022, pp. 189–200.

[6] J. M. Tjensvold, "Generic Distributed Exact Cover Solver".

[7] R. Rushmeier, K. Hoffman, and M. Padberg, "Recent Advances in Exact Optimization of Airline Scheduling Problems," Jun. 1997.

[8] S. Yakovlev, O. Kartashov, and D. Podzeha, "Mathematical Models and Nonlinear Optimization in Continuous Maximum Coverage Location Problem," *Computation*, vol. 10, no. 7, Art. no. 7, Jul. 2022, doi: 10.3390/computation10070119.

[9] R. Wei, "Coverage Location Models: Alternatives, Approximation, and Uncertainty," *International Regional Science Review*, vol. 39, no. 1, pp. 48–76, Jan. 2016, doi: 10.1177/0160017615571588.

[10] L. Li, Z. Wei, J.-K. Hao, and K. He, "Probability learning based tabu search for the budgeted maximum coverage problem," *Expert Systems with Applications*, vol. 183, p. 115310, Nov. 2021, doi: 10.1016/j.eswa.2021.115310.

[11] J. Guo and R. Niedermeier, "Exact algorithms and applications for Tree-like Weighted Set Cover," *Journal of Discrete Algorithms*, vol. 4, no. 4, pp. 608–622, Dec. 2006, doi: 10.1016/j.jda.2005.07.005.

[12] J. Yang and J. Y. Leung, "A generalization of the weighted set covering problem," *Naval Research Logistics*, vol. 52, no. 2, pp. 142–149, Mar. 2005, doi: 10.1002/nav.10093.

[13] V. V. Gusev, "Set-weighted games and their application to the cover problem," *European Journal of Operational Research*, vol. 305, no. 1, pp. 438–450, Feb. 2023, doi: 10.1016/j.ejor.2022.05.026.

[14] X. Delorme, X. Gandibleux, and J. Rodriguez, "GRASP for set packing problems," *European Journal of Operational Research*, vol. 153, no. 3, pp. 564–580, Mar. 2004, doi: 10.1016/S0377-2217(03)00263-7.

[15] B. Alidaee, G. Kochenberger, K. Lewis, M. Lewis, and H. Wang, "A new approach for modeling and solving set packing problems," *European Journal of Operational Research*, vol. 186, pp. 504–512, Feb. 2008, doi: 10.1016/j.ejor.2006.12.068.

[16] E. Hopper and B. C. H. Turton, "A Review of the Application of Meta-Heuristic Algorithms to 2D Strip Packing Problems".

[17] T. Tanaka and K. Fujii, “Melodic Pattern Segmentation of Polyphonic Music as a Set Partitioning Problem,” in *International Congress on Music and Mathematics*, Puerto Vallarta, Mexico, Nov. 2014. Accessed: Apr. 24, 2024. [Online]. Available: https://hal.science/hal-01467889
[18] “Levine - A Parallel Genetic Algorithm for the Set Partition.pdf.” Accessed: Apr. 24, 2024. [Online]. Available: https://ftp.mcs.anl.gov/pub/tech_reports/reports/P458.pdf
[19] R. Lamarche-Perrin, Y. Demazeau, and J.-M. Vincent, “A Generic Algorithmic Framework to Solve Special Versions of the Set Partitioning Problem,” in *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*, Limassol, Cyprus: IEEE, Nov. 2014, pp. 891–897. doi: 10.1109/ICTAI.2014.136.
[20] A. Mellit and S. A. Kalogirou, “Artificial intelligence techniques for photovoltaic applications: A review,” *Progress in Energy and Combustion Science*, vol. 34, no. 5, pp. 574–632, Oct. 2008, doi: 10.1016/j.pecs.2008.01.001.
[21] K. Kraus, N. Kraus, M. Hryhorkiv, I. Kuzmuk, and O. Shtepa, “Artificial Intelligence in Established of Industry 4.0,” *WSEAS TRANSACTIONS ON BUSINESS AND ECONOMICS*, vol. 19, pp. 1884–1900, Nov. 2022, doi: 10.37394/23207.2022.19.170.
[22] H. Lu, Y. Li, M. Chen, H. Kim, and S. Serikawa, “Brain Intelligence: Go beyond Artificial Intelligence,” *Mobile Netw Appl*, vol. 23, no. 2, pp. 368–375, Apr. 2018, doi: 10.1007/s11036-017-0932-8.
[23] K. R. Chowdhary, “Introducing Artificial Intelligence,” in *Fundamentals of Artificial Intelligence*, New Delhi: Springer India, 2020, pp. 1–23. doi: 10.1007/978-81-322-3972-7_1.
[24] O. E. Agwu, J. U. Akpabio, S. B. Alabi, and A. Dosunmu, “Artificial intelligence techniques and their applications in drilling fluid engineering: A review,” *Journal of Petroleum Science and Engineering*, vol. 167, pp. 300–315, Aug. 2018, doi: 10.1016/j.petrol.2018.04.019.
[25] A. D. Dongare, R. R. Kharde, and A. D. Kachare, “Introduction to Artificial Neural Network,” vol. 2, no. 1, 2012.
[26] S. M. Al-Sager, S. S. Almady, S. A. Marey, S. A. Al-Hamed, and A. M. Aboukarima, “Prediction of Specific Fuel Consumption of a Tractor during the Tillage Process Using an Artificial Neural Network Method,” *Agronomy*, vol. 14, no. 3, Art. no. 3, Mar. 2024, doi: 10.3390/agronomy14030492.
[27] F. Martínez-Álvarez, A. Troncoso, G. Cortés, and J. Riquelme, “A Survey on Data Mining Techniques Applied to Electricity-Related Time Series Forecasting,” *Energies*, vol. 8, pp. 13162–13193, Nov. 2015, doi: 10.3390/en81112361.
[28] A. Krizhevsky, I. Sutskever, and G. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” *Neural Information Processing Systems*, vol. 25, Jan. 2012, doi: 10.1145/3065386.
[29] L. M. Rojas-Barahona, “Deep learning for sentiment analysis,” *Language and Linguist. Compass*, vol. 10, no. 12, pp. 701–719, Dec. 2016, doi: 10.1111/lnc3.12228.
[30] S. Ali, J. Li, Y. Pei, M. Aslam, Z. Shaukat, and M. Azeem, “An Effective and Improved CNN-ELM Classifier for Handwritten Digits Recognition and Classification,” *Symmetry*, vol. 12, p. 15, Oct. 2020, doi: 10.3390/sym12101742.
[31] M. Ahmadi, M. Soofiabadi, M. Nikpour, H. Naderi, L. Abdullah, and B. Arandian, “Developing a Deep Neural Network with Fuzzy Wavelets and Integrating an Inline PSO to Predict Energy Consumption Patterns in Urban Buildings,” *Mathematics*, vol. 10, no. 8, Art. no. 8, Jan. 2022, doi: 10.3390/math10081270.

[32] N. Sharma, R. Sharma, and N. Jindal, “Machine Learning and Deep Learning Applications-A Vision,” *Global Transitions Proceedings*, vol. 2, no. 1, pp. 24–28, Jun. 2021, doi: 10.1016/j.gltp.2021.01.004.
[33] “Artificial Intelligence Technology and Engineering Applications - ProQuest.” Accessed: Apr. 17, 2024. [Online]. Available: https://www.proquest.com/openview/15062833a48455ce554e76d3ecfa8c94/1?pq-origsite=gscholar&cbl=6630503
[34] M. Khaleel, A. A. Ahmed, and A. Alsharif, “Artificial Intelligence in Engineering,” *Brilliance: Research of Artificial Intelligence*, vol. 3, no. 1, Art. no. 1, Mar. 2023, doi: 10.47709/brilliance.v3i1.2170.
[35] “Foulds - 1981 - Optimization Techniques.pdf.” Accessed: Jun. 05, 2024. [Online]. Available: https://moscow.sci-hub.se/4004/91da22904102f7188638341b93246d01/foulds1981.pdf?
[36] G. Venter, “Review of Optimization Techniques,” in *Encyclopedia of Aerospace Engineering*, 1st ed., R. Blockley and W. Shyy, Eds., Wiley, 2010. doi: 10.1002/9780470686652.eae495.
[37] L. R. Foulds, *Optimization Techniques*. in Undergraduate Texts in Mathematics. New York, NY: Springer New York, 1981. doi: 10.1007/978-1-4613-9458-7.
[38] R. V. Rao and V. J. Savsani, “Advanced Optimization Techniques,” in *Mechanical Design Optimization Using Advanced Optimization Techniques*, in Springer Series in Advanced Manufacturing. , London: Springer London, 2012, pp. 5–34. doi: 10.1007/978-1-4471-2748-2_2.
[39] G. Even, “Recursive Greedy Methods”.
[40] C. Cerrone, R. Cerulli, and B. Golden, “Carousel greedy: A generalized greedy algorithm with applications in optimization,” *Computers & Operations Research*, vol. 85, pp. 97–112, Sep. 2017, doi: 10.1016/j.cor.2017.03.016.
[41] “Francois and Boeffard - The Greedy Algorithm and its Application to the Co.pdf.” Accessed: Apr. 27, 2024. [Online]. Available: http://www.lrec-conf.org/proceedings/lrec2002/pdf/265.pdf
[42] M. Jedor, J. Louëdec, and V. Perchet, “Be Greedy in Multi-Armed Bandits.” arXiv, Jan. 04, 2021. Accessed: May 28, 2024. [Online]. Available: http://arxiv.org/abs/2101.01086
[43] T. M. Hernandez, R. Van Beeumen, M. A. Caprio, and C. Yang, “A greedy algorithm for computing eigenvalues of a symmetric matrix with localized eigenvectors,” *Numerical Linear Algebra App*, vol. 28, no. 2, p. e2341, Mar. 2021, doi: 10.1002/nla.2341.
[44] M. C. Angelini and F. Ricci-Tersenghi, “Modern graph neural networks do worse than classical greedy algorithms in solving combinatorial optimization problems like maximum independent set,” *Nat Mach Intell*, vol. 5, no. 1, pp. 29–31, Dec. 2022, doi: 10.1038/s42256-022-00589-y.
[45] T. M. Hernandez, R. Van Beeumen, M. A. Caprio, and C. Yang, “A greedy algorithm for computing eigenvalues of a symmetric matrix,” *Numerical Linear Algebra App*, vol. 28, no. 2, p. e2341, Mar. 2021, doi: 10.1002/nla.2341.
[46] C.-L. Tai, S.-H. Hsieh, and C.-S. Lu, “Greedy Algorithms for Hybrid Compressed Sensing,” *IEEE Signal Processing Letters*, vol. 27, pp. 2059–2063, 2020, doi: 10.1109/LSP.2020.3037732.

[47] N. Marriwala, C. C. Tripathi, S. Jain, and D. Kumar, *Mobile Radio Communications and 5G Networks: Proceedings of Third MRCN 2022*. Springer Nature, 2023.
[48] N. Minamikawa, "Geodesic property of greedy algorithms for optimization problems on jump systems and delta-matroids," *Discrete Applied Mathematics*, vol. 344, pp. 43–67, Feb. 2024, doi: 10.1016/j.dam.2023.11.003.
[49] M. Azad, I. Chikalov, S. Hussain, M. Moshkov, and B. Zielosko, "Greedy Algorithms for Decision Trees with Hypotheses." arXiv, Mar. 16, 2022. Accessed: May 28, 2024. [Online]. Available: http://arxiv.org/abs/2203.08848
[50] Z. A. Siddiqui, R. Shukla, A. Jyoti, A. Prakash, and M. Rahul, "On the Connection of Matroids and Greedy Algorithms," *IJEER*, vol. 10, no. 2, pp. 126–130, Jun. 2022, doi: 10.37391/ijeer.100213.
[51] R. Campagna, S. De Marchi, E. Perracchione, and G. Santin, "Greedy algorithms for learning via exponential-polynomial splines," *Adv Comput Math*, vol. 48, no. 6, p. 69, Dec. 2022, doi: 10.1007/s10444-022-09986-8.
[52] "An Efficient Greedy Algorithm for Real-World Large-Scale Electric Vehicle Charging," presented at the Energy-Efficient Computing and Networking, Jun. 2023. doi: 10.1145/3575813.3597349.
[53] X. He, Q.-K. Pan, L. Gao, L. Wang, P. Suganthan, and P. Nagaratnam, "A Greedy Cooperative Co-Evolutionary Algorithm With Problem-Specific Knowledge for Multiobjective Flowshop Group Scheduling Problems," Jan. 2023.
[54] L. R. Foulds, *Optimization Techniques: An Introduction*. Springer Science & Business Media, 2012.
[55] P. F. Felzenszwalb and R. Zabih, "Dynamic Programming and Graph Algorithms in Computer Vision," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 4, pp. 721–740, Apr. 2011, doi: 10.1109/TPAMI.2010.135.
[56] A. Lew and H. Mauch, *Dynamic Programming: A Computational Tool*. Springer, 2006.
[57] H. Francois and O. Boeffard, "The Greedy Algorithm and its Application to the Construction of a Continuous Speech Database".
[58] J. Liu, C.-C. Zhao, and Z.-G. Ren, "The Application of Greedy Algorithm in Real Life," *DEStech Transactions on Engineering and Technology Research*, vol. 0, no. mcee, Art. no. mcee, 2016, doi: 10.12783/dtetr/mcee2016/6391.
[59] G. Gouse Mohammad, N. Kumar Arul, and S. V A., *Advanced Applications of Python Data Structures and Algorithms*. IGI Global, 2023.
[60] B. Light, "The Principle of Optimality in Dynamic Programming: A Pedagogical Note." arXiv, Feb. 18, 2024. Accessed: May 28, 2024. [Online]. Available: http://arxiv.org/abs/2302.08467
[61] H. Gu, X. Guo, X. Wei, and R. Xu, "Dynamic Programming Principles for Mean-Field Controls with Learning." arXiv, Apr. 15, 2022. Accessed: May 29, 2024. [Online]. Available: http://arxiv.org/abs/1911.07314
[62] M. Treiber and M. A. Treiber, *Optimization for computer vision: an introduction to core concepts and methods*. in Advances in computer vision and pattern recognition. London, Heidelberg: Springer, 2013.
[63] E. E. Ammar and A. A. Emsimir, "A mathematical model for solving integer linear programming problems," *Afr. J. Math. Comput. Sci. Res.*, vol. 13, no. 1, pp. 39–50, Feb. 2020, doi: 10.5897/AJMCSR2019.0804.

[64] D. Kästner and M. Langenbach, *Code Optimization by Integer Linear Programming*. 1999, p. 136. doi: 10.1007/978-3-540-49051-7_9.
[65] Dr. Amilal Kulhari, "Significance of Linear Programming for Optimization," *IJARSCT*, pp. 179–186, May 2023, doi: 10.48175/IJARSCT-10899.
[66] R. Başar and Y. Eyüpoğlu, "Fason Üretim Planlamada Doğrusal Programlamanın Kullanılmasına Yönelik Bir Çalışma: Otomotiv Yan Sanayi Örneği," *İktisadi İdari ve Siyasal Araştırmalar Dergisi*, vol. 8, no. 21, pp. 600–617, Jun. 2023, doi: 10.25204/iktisad.1293104.
[67] O. Olanrewaju Adedipupo, L. Taiwo Abideen, O. Tolulope Samson, F. Ayodeji, and O. Kayode Oladipupo, "Application of Linear Programming for Decision Making to Business in Whao Beverages Nig. Ltd," *AJAM*, vol. 10, no. 3, p. 86, 2022, doi: 10.11648/j.ajam.20221003.11.
[68] D. Beasley, "An Overview of Genetic Algorithms : Part 1, Fundamentals".
[69] "Gendreau and Potvin - 2010 - Handbook of Metaheuristics.pdf." Accessed: Jun. 01, 2024. [Online]. Available: https://homes.di.unimi.it/cordone/courses/2023-ae/Lez22-Materiali/Ch05-GA.pdf
[70] A. B. Hassanat, V. B. S. Prasath, M. A. Abbadi, S. A. Abu-Qdari, and H. Faris, "An Improved Genetic Algorithm with a New Initialization Mechanism Based on Regression Techniques," *Information*, vol. 9, no. 7, Art. no. 7, Jul. 2018, doi: 10.3390/info9070167.
[71] A. A. E.- Sawy, M. A. Hussein, E.-S. M. Zaki, and A. A. Mousa, "An Introduction to Genetic Algorithms: A survey A practical Issues," vol. 5, no. 1, 2014.
[72] Z. B. Zabinsky, "Random search algorithms," *Department of Industrial and Systems Engineering, University of Washington, USA*, 2009, Accessed: Jun. 01, 2024. [Online]. Available: https://courses.washington.edu/inde510/516/AdapRandomSearch4.05.2009.pdf
[73] R. Tempo, G. Calafiore, and F. Dabbene, "Applications of Randomized Algorithms," in *Randomized Algorithms for Analysis and Control of Uncertain Systems*, in Communications and Control Engineering. , London: Springer London, 2013, pp. 283–327. doi: 10.1007/978-1-4471-4610-0_19.
[74] D. Wang, D. Tan, and L. Liu, "Particle swarm optimization algorithm: an overview," *Soft Comput*, vol. 22, no. 2, pp. 387–408, Jan. 2018, doi: 10.1007/s00500-016-2474-6.
[75] L. Abualigah, D. Falcone, and A. Forestiero, "Swarm Intelligence to Face IoT Challenges," *Computational Intelligence and Neuroscience*, vol. 2023, no. 1, p. 4254194, 2023, doi: 10.1155/2023/4254194.
[76] M. Fang, "Optimization application of particle swarm algorithm in civil engineering," *AETR*, vol. 3, no. 1, p. 275, Dec. 2022, doi: 10.56028/aetr.3.1.275.
[77] S. Cheng, H. Lu, X. Lei, and Y. Shi, "A quarter century of particle swarm optimization," *Complex Intell. Syst.*, vol. 4, no. 3, pp. 227–239, Oct. 2018, doi: 10.1007/s40747-018-0071-2.

.