**VIETNAM GENERAL CONFEDERATION OF LABOR**

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**FINAL REPORT**

**THE DESIGN AND ANALYSIS OF ALGORITHMS**

***Instructor***: Nguyen Chi Thien

***Student***: Hoang Dinh Quy Vu - 521H0517

Tran Nhut Anh – 521H0491

Nguyen Hoang Phuc – 521H0509

**HO CHI MINH CITY, 2023**

**VIETNAM GENERAL CONFEDERATION OF LABOR**

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**FINAL REPORT**

**THE DESIGN AND ANALYSIS OF ALGORITHMS**

***Instructor***: Nguyen Chi Thien

***Student***: Hoang Dinh Quy Vu - 521H0517

Tran Nhut Anh – 521H0491

Nguyen Hoang Phuc – 521H0509

**HO CHI MINH CITY, 2023**

# ACKNOWLEDGEMENT

I would like to thank the teachers of Ton Duc Thang University in Ho Chi Minh City, as well as the teachers of the Department of the design and analysis of algorithms for IT for their enthusiastic guidance during my time studying at the school; Special thanks to Mr. Thien who directly guided me to complete my graduation thesis.

I sincerely thank!

**PROJECT COMPLETED**

**AT TON DUC THONG UNIVERSITY**

I hereby certify that this is my thesis project and was conducted under the guidance of Mr Thien. The research content and results in this topic are honest and have not been previously published in any form. The data in the tables are collected by the author from various sources clearly stated in the reference section for analysis, comments, and evaluation.

Furthermore, this thesis includes comments, evaluations, and data from other authors and organizations, all of which are cited and annotated with their sources.

If any fraud is detected, I will fully take responsibility for the content of my thesis. Ton Duc Thang University is not liable for any copyright infringement or violation caused by me during the implementation process (if any).

*Ho Chi Minh city, 9th December, 2023*

*Author*

*(Sign and write your full name)*

# TEACHER'S CONFIRMATION AND ASSESSMENT SECTION

**The confirmation part of the instructor**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Ho Chi Minh city, 11th April, 2023*

**The evaluation part of the teacher marks the test**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Ho Chi Minh city, 11th April, 2023*

# TABLE OF CONTENTS

[ACKNOWLEDGEMENT i](#_Toc153187990)

[TEACHER'S CONFIRMATION AND ASSESSMENT SECTION iii](#_Toc153187991)

[TABLE OF CONTENTS 1](#_Toc153187992)

[LIST OF SYMBOLS AND ABBREVIATIONS 3](#_Toc153187993)

[CHAPTER 1: AN IMPORVED CHEMICAL REACTION OPTIMIZATION ALGORITHM FOR SOLVING THE SHORTEST COMMON SUPERSEQUENCE PROBLEM 4](#_Toc153187994)

[1.1 Introduction 4](#_Toc153187995)

[1.1.1 Shorest common supersequence – SCS 4](#_Toc153187996)

[1.1.2 Chemical reaction optimization - CRO 4](#_Toc153187997)

[1.1.3 Improved chemical reaction optimization – IMCRO 4](#_Toc153187998)

[1.1.4 Different approaches 5](#_Toc153187999)

[1.2 IMCRO 5](#_Toc153188000)

[1.2.1 Framework 5](#_Toc153188001)

[1.2.2 Activity chain 6](#_Toc153188002)

[1.2.3 Operators 9](#_Toc153188003)

[1.2.3.1 On-wall ineffective collision 9](#_Toc153188004)

[1.2.3.2 Inter-molecular ineffective collision 10](#_Toc153188005)

[1.2.3.3 Decomposition 11](#_Toc153188006)

[1.2.3.4 Synthesis 12](#_Toc153188007)

[1. 3.2.5 Repair function 13](#_Toc153188008)

[1.3 Algorithms extends 13](#_Toc153188009)

[1.3.1 Artificial Bee Colony - ABC 13](#_Toc153188010)

[1.3.2 Ant colony optimization- ACO 16](#_Toc153188011)

[CHAPTER 2: ANALYZE THE COMPLEXITY 18](#_Toc153188012)

[2.1 Initialization\_IMCRO 18](#_Toc153188013)

[2.2 Iteration 19](#_Toc153188014)

[2.2.1 Operator 19](#_Toc153188015)

[2.2.1.1 On-wall ineffective collision 19](#_Toc153188016)

[2.2.1.2 Inter-molecular ineffective collision 20](#_Toc153188017)

[2.2.2.3 Decomposition 20](#_Toc153188018)

[2.2.1.4 Synthesis 21](#_Toc153188019)

[2.3 ACB 22](#_Toc153188020)

[2.4 ACO 23](#_Toc153188021)

[2.4.1 Initialization\_ACO 23](#_Toc153188022)

[2.4.2 Iteration 24](#_Toc153188023)

[CHAPTER 3: RESULT AND ANALYSIS 28](#_Toc153188024)

[3.1 Review Paper 28](#_Toc153188025)

[3.2 Result 28](#_Toc153188026)

[REFERENCES 30](#_Toc153188027)

# LIST OF SYMBOLS AND ABBREVIATIONS

**SYMBOLS**

**ABBREVIATIONS**

# CHAPTER 1: AN IMPORVED CHEMICAL REACTION OPTIMIZATION ALGORITHM FOR SOLVING THE SHORTEST COMMON SUPERSEQUENCE PROBLEM

## 1.1 Introduction

### 1.1.1 Shorest common supersequence – SCS

* The SCS problem is a classical NP-hard problem that is normally solved by heuristic algorithms.
* It was first defined by David Maier in 1976, and it seeks to find the common supersequence with the minimum length.
* SCS has various applications in fields such as DNA sequencing, data compression, AI optimization, and multiple sequence alignment

### Chemical reaction optimization - CRO

* The CRO algorithm, initially introduced by Lam and Li in 2010, draws inspiration from the sequential steps involved in chemical reactions.
* In a chemical reaction, various sub-reactions occur, leading to a series of intermediate states.
* At each state, the energy of the molecule decreases, resulting in increased stability. This concept can be applied to the step-wise search approach used in optimization problems.

### Improved chemical reaction optimization – IMCRO

* A new algorithm called IMCRO is introduced for solving the SCS problem.
* The key contribution of this paper is the enhancement of the existing CRO\_SCS framework by introducing two new operators for decomposition and inter-molecular ineffective collisions in two of the four reactions of CRO.
* The results show that these new operators significantly improve the performance and efficiency of CRO in solving the SCS problem

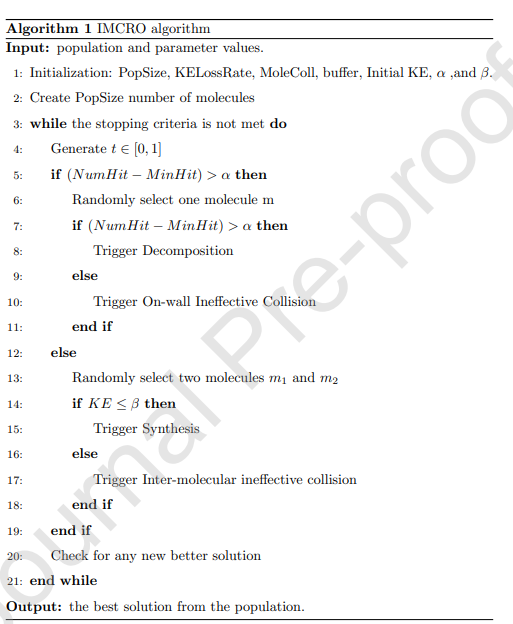
### Different approaches

* Various approaches have been proposed in order to address the SCS problem and find the optimal solution. Some significant proposals found in the literature include greedy methods, ant colony optimization (ACO), and artificial bee colony (ABC),…

## IMCRO

### 1.2.1 Framework

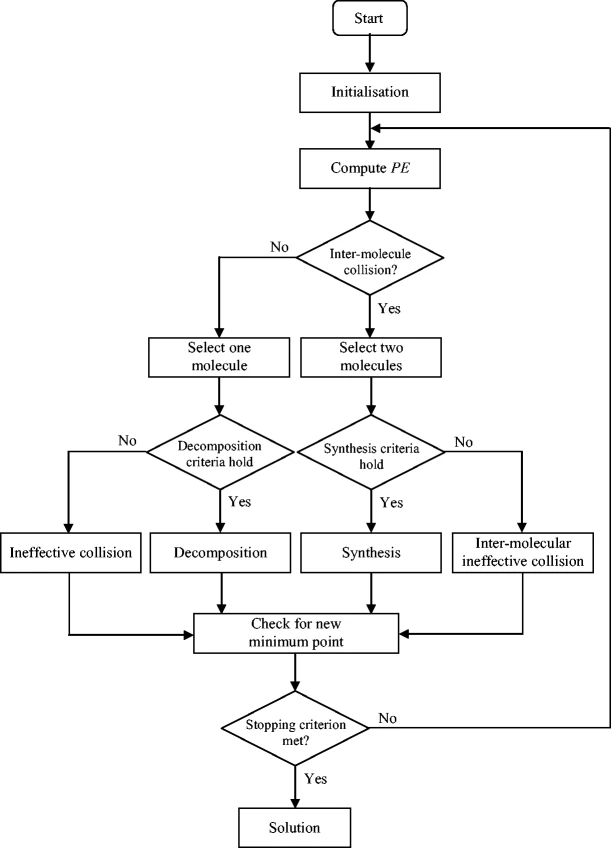
* The proposed framework in our proposal is an enhanced extension of the CRO-based algorithm, known as CRO SCS. It is composed of three stages, namely initialization, iteration, and finalization, which are described in detail in Algorithm 1.



**Algorithm 1**

### 1.2.2 Activity chain

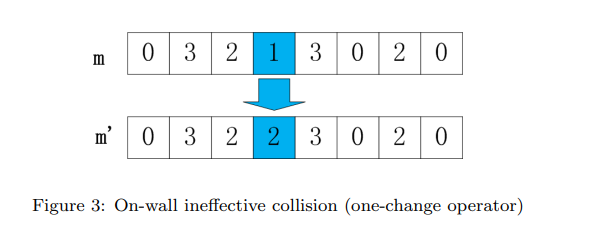
* Initialization (Stage 1):
  + Elements and molecules initialized, including PopSize, KELossRate, MoleColl, buffer, Initial KE, α, and β.
  + Molecule energy includes potential energy (PE) and kinetic energy (KE).
  + PE is related to the objective function (Formula 2), KE reflects tolerance.
  + Population generation and supersequence representation involve random insertion operations.
  + Integer encoding used to represent symbols from the alphabet.
* Iteration (Stage 2):
* Divided into reaction and repair subtasks.
* Reaction step involves on-wall ineffective collision, decomposition, inter-molecular ineffective collision, and synthesis.
* Uni-molecule and inter-molecule reactions determined by parameters α and β.
* Main iteration randomly generates parameter t to decide reaction type.
* Algorithm 1 IMCRO:
* Input: Population and parameter values.
* Initialization creates PopSize number of molecules.
* Main loop continues until stopping criteria are met.
* Reaction type determined by random parameter t.
* Different reactions (decomposition, on-wall ineffective collision, synthesis, inter-molecular ineffective collision) triggered based on conditions.
* Check for any new better solution in each iteration.
* Validation and Finalization (Stages 3 and 4):
* Validity check: If solution does not meet requirements, a repair algorithm is applied.
* Final stage checks stopping criteria (e.g., CPU time, function evaluations, objective function value).
* IMCRO outputs the best solution found with its objective value and terminates the procedure.

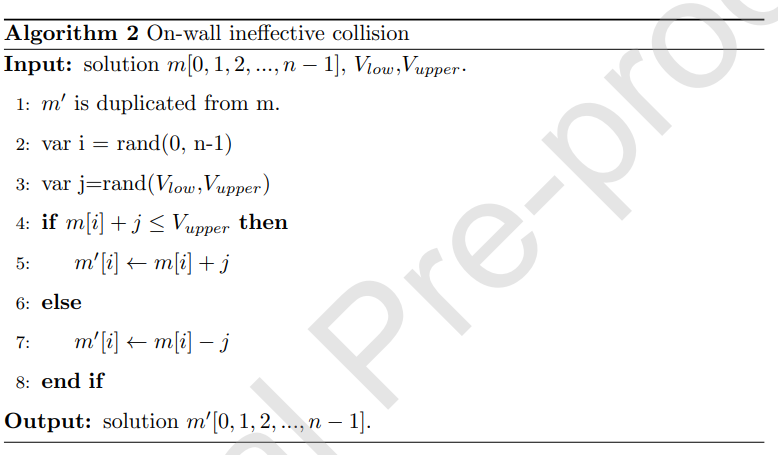


### 1.2.3 Operators

#### 1.2.3.1 On-wall ineffective collision

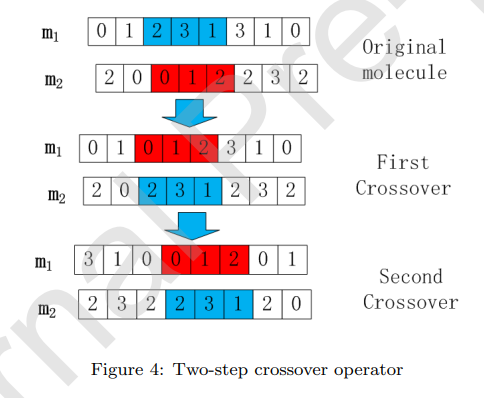
* The one-difference operator is an algorithm that modifies a character in a supersequence, which helps to explore nearby solutions without reducing the overall length. This operator expands the possibilities for finding the Shortest Common Supersequence (SCS).

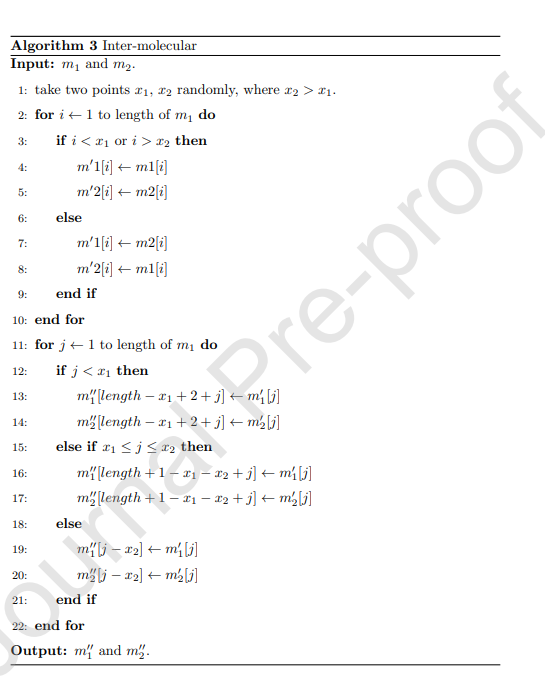




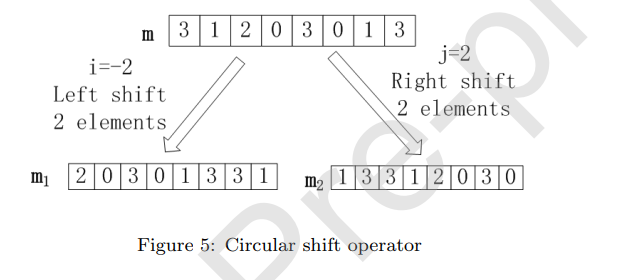
#### 1.2.3.2 Inter-molecular ineffective collision

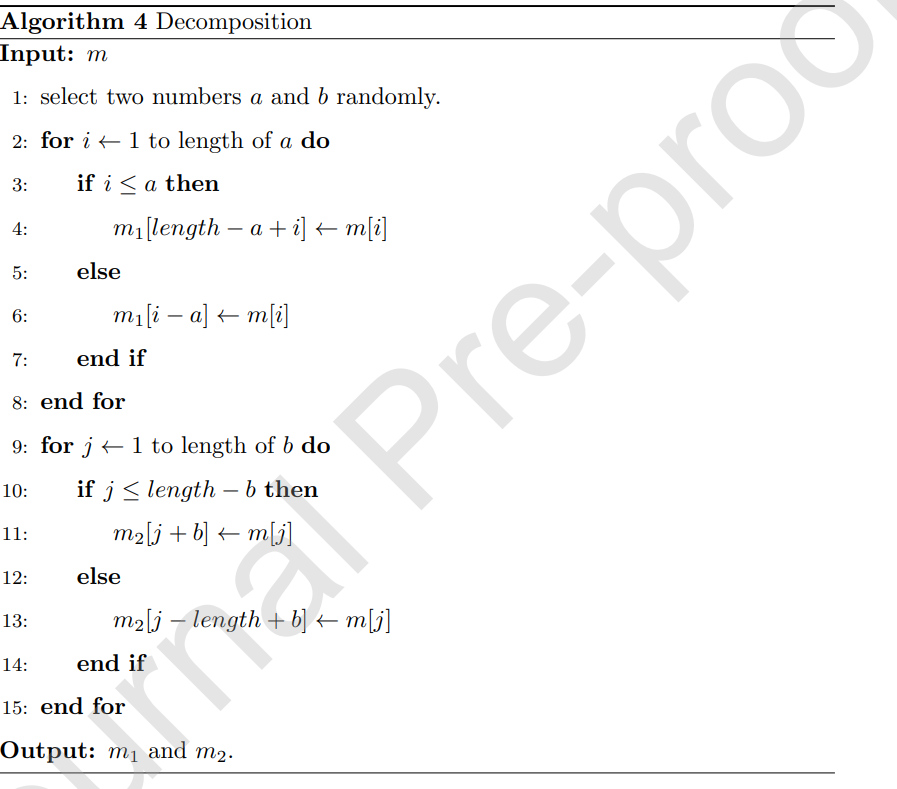
* This algorithm randomly selects two molecules, m1 and m2, and applies two crossover operators: the first between two different molecules, and the second within each individual molecule, resulting in two new solutions, m'1 and m'2.





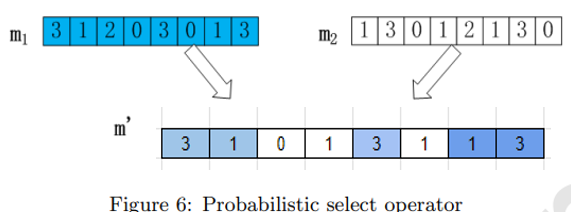
#### 1.2.3.3 Decomposition

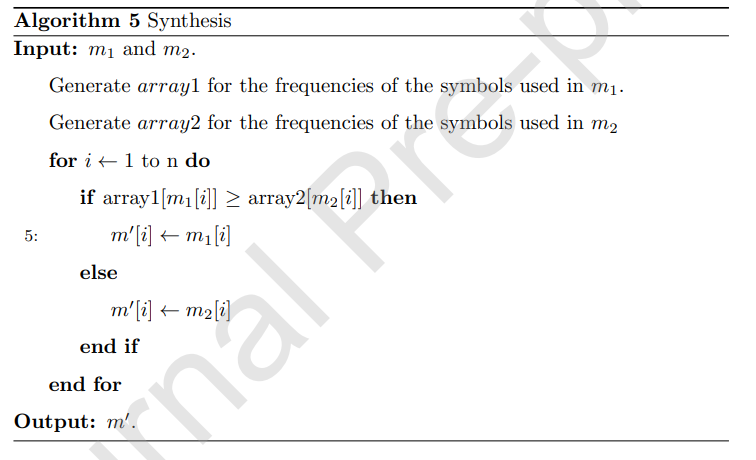
* The decomposition reaction enables exploration of a different search space, causing significant changes in molecular structures.
* In a circular shift operator, generated within the range [-n, n].
* Two randomly chosen integers,
* -i : shift left i times
* j : shift right j times



#### 1.2.3.4 Synthesis

* This algorithm combines two molecules, m1 and m2, to form a new molecule, m', through a process opposite to a decomposition reaction. An enhanced probabilistic selection operator variant expedites convergence by exploring diverse molecular structures and frequencies.





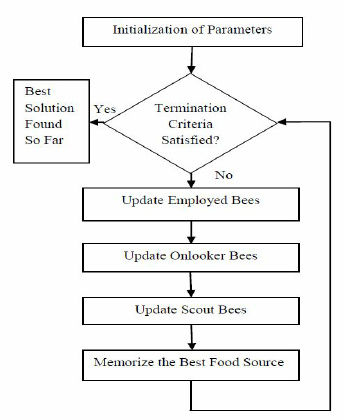
#### 1. 3.2.5 Repair function

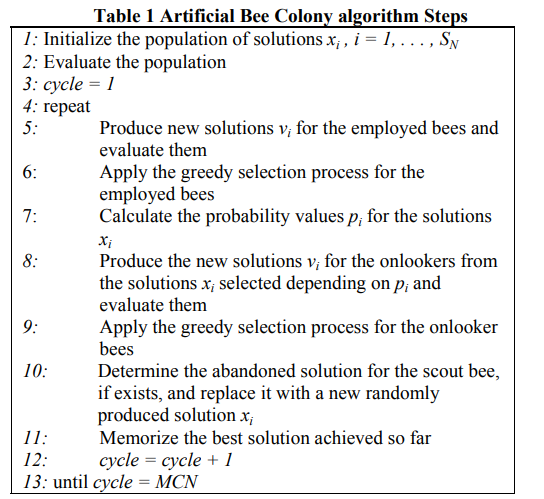
* This algorithm includes a two-phase process to ensure the validity of the new supersequence. The first phase involves validation, where the algorithm checks for any violations.
* If no violations are found during validation, the new molecule is seamlessly inserted into the population. However, if a violation is detected, the algorithm initiates the repair phase.
* During repair, the new molecule undergoes scrutiny against a predefined violation threshold (VT).
* If the extent of violation surpasses the threshold, the algorithm undertakes corrective measures to bring the molecule into compliance, ensuring the integrity of the resulting supersequence.

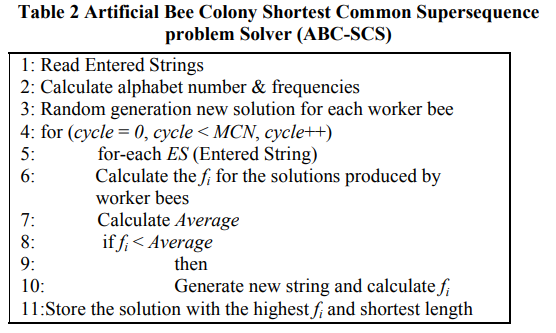
## Algorithms extends

### 1.3.1 Artificial Bee Colony - ABC

* The Artificial Bee Colony (ABC) is an optimization algorithm inspired by the foraging behavior of honeybees. Developed by Dervis Karaboga in 2005, the ABC algorithm is a technique based on swarm intelligence.
* It mimics the collaboration and communication observed in real honeybee colonies to solve complex optimization problems. The algorithm consists of three components: employed bees, onlooker bees, and scout bees, which work together to explore and exploit the solution space.
* The algorithm starts with a population of artificial bees that represent potential solutions.
* The employed bees evaluate the fitness of these solutions and share their findings with the onlooker bees.
* The onlooker bees then choose solutions based on the fitness information provided. Additionally, the scout bees contribute to the diversity of the population by exploring new and unexplored solutions.

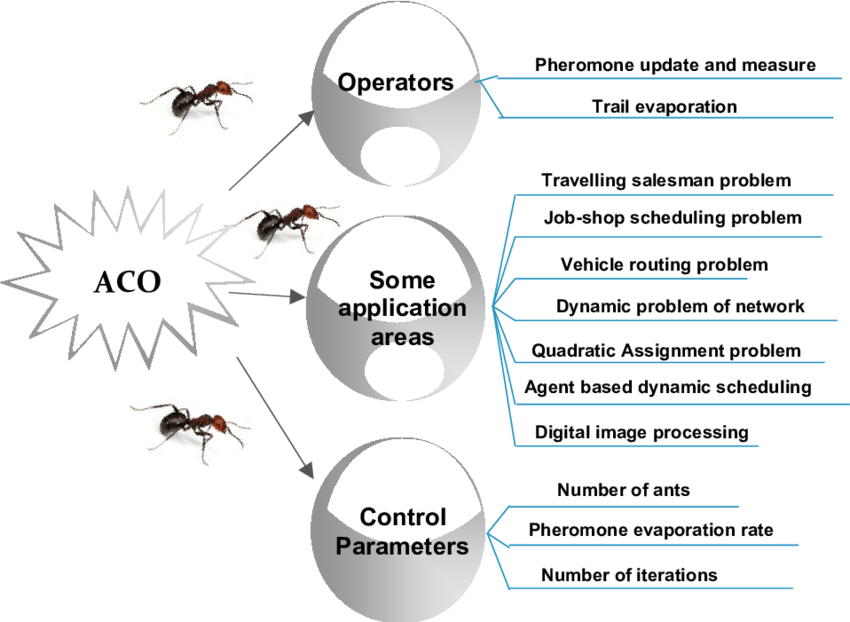




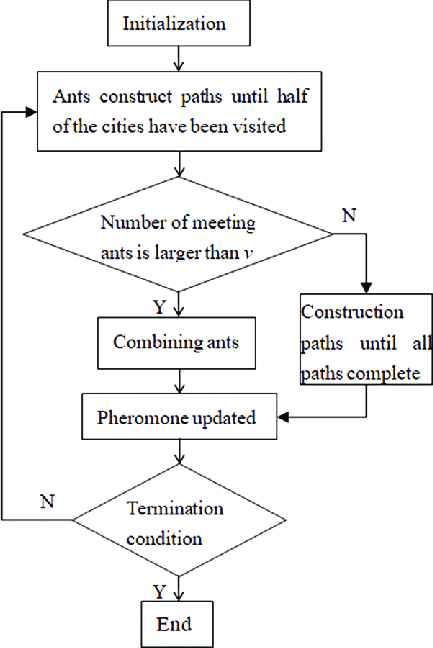


### 1.3.2 Ant colony optimization- ACO

* ACO algorithm mimics ant foraging to find an optimal path. Ants explore paths, leaving pheromones. Paths with more pheromones are favored. For the SCS problem, ants represent paths in a sequence, and pheromones indicate promising subsequences. The algorithm leverages cumulative pheromone information to guide optimal path selection.



* This is activity chain of algorithm:
* 1. Initialize ants:
  + Create ants with random paths.
  + Calculate ant path distances.
* 2. Main loop:
  + 2.1 Select best ant.
  + 2.2 Perform local pheromone updates:
    - For each ant except the best 10%:
      * Swap random cities.
      * Update ant path if improved.
  + 2.3 Perform global pheromone updates:
    - Swap three random cities for the best ant.
    - Update ant path if improved.
  + 2.4 Update pheromone levels.
  + 2.5 Sort ants by distance.
* 3. Output shortest path:
  + Return path of the ant with the shortest distance.



# CHAPTER 2: ANALYZE THE COMPLEXITY

## 2.1 Initialization\_IMCRO

* In this class Initialization\_IMCRO, it have 4 functions:
* supersequenceGenerate: The method creates a supersequence by randomly adding characters from a group of strings. It iterates through each string in the group and, for every character in the string, adds it to the supersequence at a random position.
* populationGeneration: Calls supersequenceGenerate for each individual in the population.
* encodingPopulation : Maps characters to integers based on a predefined dictionary.
* createMolecule : Converts a string of digits into a list of integers and creates a Molecule object with random doubles and the list of integers
* Basic operation: The initialization process includes population generation, encoding population, and molecule creation. These operations involve string manipulation, character mapping, and list conversion.

And finally, function initialization:

* Objective: Create number of popsizes base on set of strings input, it called supersequences. Calls population Generation encodingPopulation and create Molecule
* Input:
  + popsize: popsize the list of molecule instance
  + setOfStrings: list of strings
* Worst case: when the population size (popSize), the length of the longest string (N), and the number of strings in the set (M) are maximized.
* Time complexity: O(popSize \* N \* M)
  + N is the length of the longest string in the set of strings.
  + M is the number of strings in the set.
* Bestcase: The best-case scenario is characterized by a small population size, short string lengths, and a minimal number of strings in the set

## 2.2 Iteration

### 2.2.1 Operator

#### 2.2.1.1 On-wall ineffective collision

* Objective:
  + This focuses on instances of molecules colliding with container walls, resulting in structural transformations.
  + A "one-difference operator" is used to make a single change in the molecule's composition to achieve this.
* Input:
  + molecule (list): the input molecule and it represent by list
* Output:
  + The method returns a new list.
* The best case and the worst case are the same; they occur when the array access and operations are constant.
* Time Complexity: O(1), constant time complexity due to simple array access, comparisons, and assignments.

#### 2.2.1.2 Inter-molecular ineffective collision

* Objective:
  + The purpose is to introduce significant changes to enhance local search capabilities and prevent getting stuck in local optimization by promoting diversity.
  + A crossover operator is used in genetic or evolutionary algorithms for optimization. It selects two molecules from the population and uses a two-step mechanism to generate two new solutions.
  + It is a two step process: the first step is to crossover between two molecules, and the second step is to crossover inside the molecule itself
* Input:
  + molecule (list): the input molecule and it represent by list
* Output:
  + The method returns a tuple (m1, m2), where m1 and m2 are the two molecules and m1, m2 are also list.
* Best case: O(1), where the lengths of the input molecules are minimal.
* Worst-case: O(N), where N is the maximum length of the input molecules.
* Time complexity of O(N), where N is the length of the input molecules. The method iterates over the molecules and performs crossover operations up to the minimum length of the two input molecules.

#### 2.2.2.3 Decomposition

* Objective:
  + The decomposition involves randomly selecting two numbers 'a' and 'b', and then splitting the input molecule into two new molecules, 'm1' and 'm2', based on the selected numbers.
    - The negative number −a is used for shifting to the left a steps.
    - The positive number j is used for shifting to the right j steps.
* Input:
  + molecule (list): the input molecule and it represent by list
* Output:
  + The method returns a tuple (m1, m2), where m1 and m2 are the two molecules and m1, m2 are also list.
* Best case: O(1), where the lengths of the input molecules are minimal.
* Worst-case: O(N), where N is the maximum length of the input molecules.
* Time Complexity: O(N), where N is the length of the input molecules. The method involves iterating over the molecules to create two new molecules based on random numbers 'a' and 'b

#### 2.2.1.4 Synthesis

* Objective:
  + Generates a new list by combining two input lists in a way that preserves the frequency of the symbols used in each input list.
* Input:
  + molecule1 (list): The first input list.
  + molecule2 (list): The second input list.
* Output:
  + The method returns a new list.
* Best case: O(1), where the lengths of the input molecules are minimal.
* Worst-case: O(N), where N is the maximum length of the input molecules.
* Time Complexity: O(N), where N is the length of the input molecules. The method involves iterating over the molecules to create a new list by preserving the frequency of symbols in the input lists.

## 2.3 ACB

* OBject: Represents an Artificial Bee Colony optimization algorithm for molecule structure.
* Input:
  + initial\_pop: Initial population for the optimization.
  + molecule (list): List of atoms forming the molecule. (Note: molecule is not defined in this scope.)
  + frequencies (list): List of frequencies corresponding to each atom in the molecule.
  + population\_size (int): Size of the population in the optimization.
  + max\_cycles (int): Maximum number of cycles or iterations.
  + mo: Initialization (Assumed to be a function or data structure) result for the population.
  + n (int): Number of cycles or iterations.
  + molecules (list): List of Molecule\_Bee instances representing the population of molecules.
* Output:
  + List of integers representing the best structure found by the Artificial Bee Colony algorithm.
* Best Case: O(1), the best case occurs when the algorithm quickly converges to the optimal solution, resulting in a low number of iterations.
* Worst Case: O(maxCycles \* populationSize). The worst case occurs when the algorithm takes the maximum allowed number of cycles to converge or does not converge at all.
* Time Complexity: O(maxCycles \* populationSize. The dominant factor in the time complexity is the loop that iterates for maxCycles, and within that loop, there are operations that involve the population size.

## 2.4 ACO

### 2.4.1 Initialization\_ACO

* In the Ant Colony Optimization (ACO) algorithm's Initialization class, the method initialize oversees two key functions: populationGeneration and encodingPopulation from Initialization\_IMCRO. These functions collaboratively generate an initial population of supersequences and encode them into integers.

**populationGeneration:**

* Objective:
  + Generate a diverse initial supersequence population iteratively based on provided strings.
* Time Complexity:
  + O(popSize \* N \* M), where popSize is the population size, N is the length of the longest string, and M is the number of strings.

**encodingPopulation:**

* Objective:
  + Encode initial supersequences into integers using a predefined dictionary.
* Time Complexity:
  + O(P \* L), where P is the population size, and L is the average supersequence length.
* The initialize method synchronizes these functions, laying the foundation for the IMCRO algorithm. It assembles molecules from generated and encoded supersequences, playing a pivotal role in subsequent stages. Time complexity is primarily governed by populationGeneration, emphasizing its significance in IMCRO.
* **Inputs:**
  + popSize: Signifying the population size.
  + setOfStrings: Representing the list of strings.
* **Worst Case:**
  + Manifests when popSize, N (length of longest string), and M (number of strings) reach maximum. Time Complexity: O(popSize \* N \* M).
* **Best Case:**
  + Envisions a scenario with a small population, short strings, and a sparse set. An aspirational benchmark for optimal performance.
  + The method returns a tuple (m1, m2), where m1 and m2 are the two molecules

### 2.4.2 Iteration

**calculateDistance:**

* Objective:
  + Computes the total distance of a given ant path.
* Input:
  + antPath (List of Integers): The sequence representing the ant path.
* Output:
  + distance (double): The total distance of the ant path.
* Best Case:
  + O(1), when the ant path is short.
* Worst Case:
  + O(N), where N is the length of the ant path.
* Time Complexity:
  + O(N), as it involves iterating over the ant path to calculate the distance.

**swap:**

* Objective:
  + Swaps two elements in a sequence.
* Input:
  + sequence (List of Integers): The sequence to be modified.
  + i (int): Index of the first element.
  + j (int): Index of the second element.
* Output:
  + Modified sequence after swapping elements.
* Best Case:
  + O(1), for a quick swap.
* Worst Case:
  + O(1), when swapping two elements.
* Time Complexity:
  + O(1), as it directly performs the swap operation.

**globalPheromoneUpdate:**

* Objective:
  + Updates the pheromone globally by swapping three elements.
* Input:
  + antPath (Entry<List<Integer>, Double>): Ant path and its associated distance.
  + a (int): Index of the first element to swap.
  + b (int): Index of the second element to swap.
  + c (int): Index of the third element to swap.
* Output:
  + Updated antPath after global pheromone update.
* Best Case:
  + O(1), for a quick global update.
* Worst Case:
  + O(N), when updating pheromones in a longer ant path.
* Time Complexity:
  + O(N), as it involves iterating over the ant path to perform the update.

**solve:**

* Objective:
  + The main method implementing the ACO algorithm, aiming to find an optimal solution.
* Input :
  + No input this method is to run the ACO
* Output:
  + Ant paths and pheromone updates based on probabilities.
* Best Case:
  + O(1), when the optimal solution is found quickly.
* Worst Case:
  + Iterative process with O(iterationSize) complexity.
* Time Complexity:
  + O(iterationSize), where iterationSize is the number of iterations in the ACO algorithm.

# CHAPTER 3: RESULT AND ANALYSIS

## 3.1 Review Paper

* In the paper “An Improved Chemical Reaction Optimization Algorithm for Solving the Shortest Common Supersequence Problem ” by Fei Luo, Cheng Chen, Joel Fuentes. All algorithms were implemented in Java and executed in a computer machine with Intel Core i5-4210U CPU at 2.40GHz, 4.00GB RAM and Windows 7 (64 bits).
* We will repeat the process and execute all algorithms (MYIMCRO, ACO, ACB) that were implemented in Java. The execution will take place on a computer with 13th Gen Intel(R) Core(TM) i5-13420H 2.10 GHz processor, 16.00 GB RAM, and running on Windows 11 (64-bit). This will ensure that the performance of all algorithms is improved compared to what was mentioned in the paper.

## 3.2 Result

* In the beginning, we are conducting a test.
* **initialPop** = "acg", "cat", "gtt", "tgc","tcc"

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **popSize** | **iterations** | **IMCRO** | ABC | **ACO** |
| 20 | 10 | 0.015 | 0.017 | 0.015 |
| 50 | 100 | 0 | 0 | 0.039 |
| 100 | 200 | 0.002 | 0 | 0.017 |
| 500 | 500 | 0.006 | 0.035 | 0.031 |
| 1000 | 1000 | 0.015 | 0.047 | 0.094 |

* Second time, we are conducting tests using a larger population size and a greater number of iterations.
* **initialPop** = "acg", "cat", "gtt", "tgc","tcc", "acg"

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **popSize** | **iterations** | **IMCRO** | **ABC** | **ACO** |
| 100 | 500 | 0.001 | 0 | 0.017 |
| 500 | 500 | 0 | 0.037 | 0.008 |
| 1000 | 1000 | 0 | 0 | 0.031 |
| 5000 | 1000 | 0.016 | 0.708 | 0.047 |
| 100 | 100 | 0 | 0.017 | 0.003 |
| 500 | 100 | 0 | 0.031 | 0.017 |
| 100 | 500 | 0 | 0.109 | 0.101 |
| 500 | 500 | 0.001 | 0.006 | 0.004 |
| 1000 | 500 | 0.001 | 0.066 | 0.002 |

# REFERENCES

[1] Fei Luo, Cheng Chen, Joel Fuentes, An improved chemical reaction optimization algorithm for solving the shortest common supersequence problem (2020)   
[2] A. S. Jaradat, M. M. Noaman, Solving shortest common supersequence problem using artificial bee colony algorithm, International Journal of ACM Jordan 2 (1) (2011) 180–185.

[3] S. Rajendran, C. Rajendran, H. Ziegler, An ant-colony algorithm to transform jobshops into flowshops: A case of shortest-common-supersequence stringology problem, in: Bio-Inspired Models of Network, Information, and Computing Systems - 5th International ICST Conference, BIONETICS 2010, Boston, MA, USA, December 1-3, 2010, Revised Selected Papers, 2010, pp. 413–424