December 2018

Combinatorial Ant Optimization and the Flowshop Problem

Tasmin Chowdhury

COMBINATORIAL ANT OPTIMIZATION AND THE FLOWSHOP PROBLEM

by

Tasmin Chowdhury

Bachelor of Science (B.Sc.)
Bangladesh University of Engineering and Technology
2016

A thesis submitted in partial fulfillment of
the requirements for the

Master of Science in Computer Science

Department of Computer Science
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
December 2018

UNLV | GRADUATE COLLEGE

Thesis Approval

The Graduate College
The University of Nevada, Las Vegas

November 15, 2018

This thesis prepared by

Tasmin Chowdhury

entitled

Combinatorial Ant Optimization and the Flowshop Problem

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science
Department of Computer Science

Wolfgang Bein, Ph.D.
Examination Committee Chair

Kathryn Hausbeck Korgan, Ph.D.
Graduate College Interim Dean

Ajay Datta, Ph.D.
Examination Committee Member

Justin Zhan, Ph.D.
Examination Committee Member

Pradip Bhowmik, Ph.D.
Graduate College Faculty Representative

Abstract

Researchers have developed efficient techniques, meta-heuristics to solve many Combinatorial Optimization (CO) problems, e.g., Flow shop Scheduling Problem, Travelling Salesman Problem (TSP) since the early 60s of the last century. **Ant Colony Optimization (ACO)** and its variants were introduced by Dorigo et al. [DBS06] in the early 1990's which is a technique to solve CO problems. In this thesis, we used the ACO technique to find solutions to the classic **Flow shop Scheduling Problem** and proposed a novel method for solution improvement. Our solution is composed of two phases; in the first phase, we solved TSP using ACO technique which gave us an initial permutation or tour. We used the same trip as an initial solution for our problem and then improved it by using 2-opt exchanges which yielded in a promising result. Furthermore, we introduced another improvement technique which gave us a more promising result. We have compared our results with the best (optimal) and worst solution known till date. A comprehensive experimental study using existing dataset proves that our approach remarkably gives good results.

Acknowledgements

At first, I express my most profound sense of gratefulness to Almighty Allah Who enables me to complete this research work.

I feel a proud privilege to express deepest gratitude and indebtedness to my research supervisor, Dr. Wolfgang Bein, Professor, Department of Computer Science, University of Nevada, Las Vegas for his scholastic guidance, valuable suggestions, constructive criticisms, outstanding assistance, instructions, and inspiration during the whole period of research work and preparing this dissertation.

Sincere gratitude and indebtedness are extended to Dr. Ajay Datta, Department of Computer Science, for serving on my thesis committee, for his valuable advice, needful suggestions, and heartfelt co-operation during the GA work and final preparation of the thesis.

I also feel proud to express my great respect and indebtedness to Dr. Justin Zhan, Department of Computer Science, for his encouragement and help from the very first semester of my study.

Heartiest gratitude to Dr. Pradip Bhowmik, Department of Chemistry, for serving in my thesis committee and also for his constant support and advice.

Countless thanks and gratefulness to the staffs of the Department of Computer Science, Graduate College, Graduate Financial Services, and Office of International Students and Scholars for their assistance throughout my study and research work.

Finally, I have no words to express gratefulness ever to my parents, husband Tasnim Imran Sunny, sister Tanzin Chowdhury and brother Abrar Chowdhury for their encouragement, patience, and co-operation during the period of study.

Tasmin Chowdhury

University of Nevada, Las Vegas

December 2018

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Ant Colony Optimization

1.1 What is Combinatorial Optimization (CO)

In many areas of Computer Science, Artificial Intelligence, Operations Research, Software Engineering, Machine Learning, and so on,**Combinatorial Optimization (CO)** [PS82] problems are critical to approach. A CO problem is an optimization problem, where an optimal solution has to be identified from a finite set of solutions. The solutions are generally discrete or can be formed into discrete. Some works have been done on continuous and mixed-variable optimization [Soc04]. So the simple concept is, we always try to optimize an objective function, it can be a minimization problem or a maximization problem depending on the problem statements. Solutions usually are expressed as sets, subsets, combinations or permutations. Here are some popular application area for CO problems.

- portfolio selection
- allocating register memory, picking up packages
- combinatorial auctions winner determination
- best airline network
- protein structure prediction
- allocations of jobs to people
- closure problem
- Internet data packet routing

- planning, scheduling, timetabling

There are a large amount of literature and approaches have been published since the early '60s to address CO problems. In this thesis, we concentrate on solving the well-known NP-complete discrete optimization problem named **Permutation Flow shop Scheduling Problem** in a different way than the state of the art approaches. There are already many simplified models used to address our concerned problem. However, Our approach gave a promising result. Some big real-life CO problems are shown below.

- finding the order of arcs with minimal backward cost and so on.
- finding shortest/cheapest round trips, known as Travelling Salesman Problem (TSP)
- finding models of propositional formulae, known as Boolean satisfiability problem (SAT)
- partitioning graphs or digraphs
- Graph Coloring Problem (GCP)
- finding variable assignment which satisfies constraints known as, Constraint Satisfaction Problem (CSP)
- partitioning, packing, covering sets

1.2 What is Continuous optimization problem?

In Continuous Optimization, the variable used in the objective function is required to be continuous, unlike Combinatorial Optimization. e.g., surface, curves, circle. Combinatorial Optimization, in contrast, has discrete variables in the objective function, a collection of things.

1.3 Ant System

ACO is based on ants systems and works with a set of ants. In this section, we will talk about virtual ant which is capable of producing many different types of complex behavior. In the book [Fla98] they have studied on ant system and put some complex scenarios an ant can behave.

In figure 1.1 an ant is located on a grid which is either painted black or white. This grid wraps around edges to edges. The ant is called Langton's ant [Lan95] as Langton is the one who first introduced it which follows the rules mentioned in the following:

- The ant takes a step forward.
- Langton's ant will always face in one of four directions (north, south, east, or west.)
- If the ant is now standing on a white grid, it will paint the grid black and turns 90 degrees to the right.
- or if it is standing on a black grid, then it paints it white and turns 90 degrees to the left.

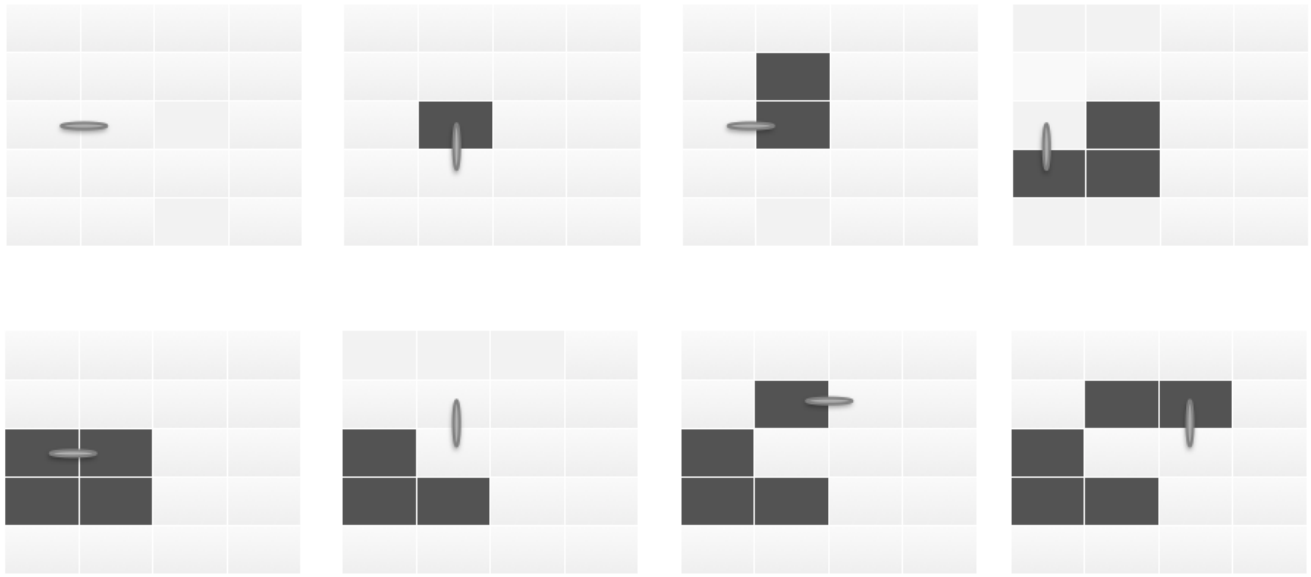

Figure 1.1: Eight steps of Langton's virtual ant

Figure 1.1 shows the steps of an ant would take. As the rules for Langton's ant are time reversible, they had the opportunity to undo the simulation in time. From the rule set of virtual ant we can imply, if an ant is standing on a black cell, the cell was white at the previous time step. After that, the ant just made a 90 degree turn to the right. They remarked in the book that the ant's entire future is deterministic, as well as the past. This would seem to imply that Langton's ants in every perspective are simple, even it has global simplicity. However, time reversibility does not intimate global integrity. They also tried to confirm that whether the ant is recursive or not in its prior actions. We can see from figure 1.1 that ant rapidly begins to cooperate with grid areas that it has been previously, which implies that the ant's future state and future conduct are a component of all its past activities. Thus if were to randomly flip the color of a cell at some point in the past, the ant's entire future could be dramatically altered. Thus, in a way, Langton's ants possess a form of sensitivity to initial conditions.

However, this not necessarily means that an ant will always behave in a way that appears random in the long term. They kept on reenacting the ant from figure 1.1 and after some time the ant started to build the highway which is a phenomenon discovered by James Propp.

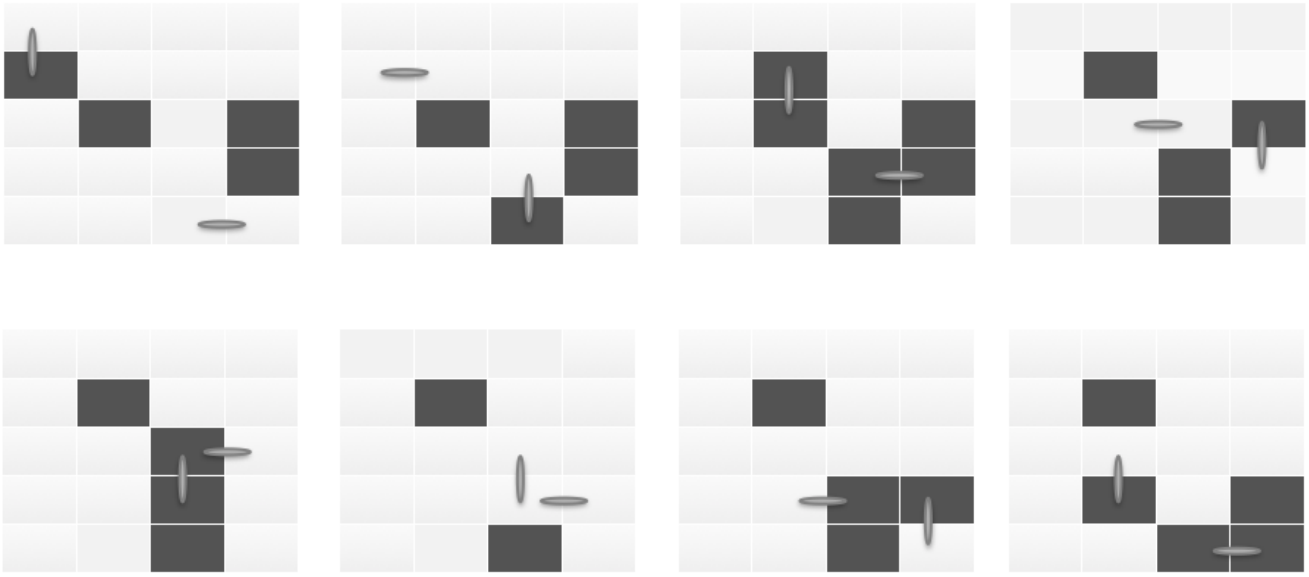

Figure 1.2: Two virtual ant reversing one another's work

If more than one ants are moving into a single cell space, they can interact and will collide into each other in a way that they will undo one another's work using the reversible nature of the ant rules. It is shown in figure 1.2. However, if two ants are allowed to move for a while before they intersect one another, then for the remaining of the simulation time they will often go into a stage of unbuilding a highway and removing random like patterns; therefore they will clean up the grid after some time. So there is a conclusion might happen, when two ants collide with one another, they will both start to undo one another's work which might make an empty grid.

When three or more ants start to roam around in a space more exciting things can happen. Suppose two ants are working and they intersected in a way they again reversing all their work, except for the third ant be roaming around and causing minor damage to the work of other two ants. Then if other two by any chance collide a cell the third one has more recently visited, they will start building a highway from there stopping the previous reversing project. Consequently, highways are very common to see being built then only partially unbuilt.

Highway building nature is found on many other virtual ants other than Langton's ant. Some examples of generalized virtual ants will behave chaotically for hundreds of thousands of step and only then start making highways. For other examples, no one knows if they will be able to build a

highway ever because of their chaotic nature persistent.

Another interesting observation made by Stewart is that [Ste94] Virtual ant's long-term behavior is unnoticed by us. Even though we knew and studied the initial setting of the ant's world, we do not know if the ant will be able to build a highway. With this, he made a statement that even though physicists uncover theory of everything of our universe, also the initial state of this yet we will be unable to know the long-term behavior of our universe. so he said, "Theory of everything predicts everything yet explains nothing."

1.4 What is Ant Colony Optimization (ACO)

Section 1.3 was to familiarize ourselves with the Ant system and their complex behavior. We will now study how we can utilize ant's foraging behavior to solve NP-hard problems. Ant colony optimization (ACO) [DS04] is an idea of swarm intelligence [DBB$^+$04] introduced by Dorigo et al. [DBS06] in the early 1990's which has been later used to solve many CO problems. In swarm intelligence, each ant is a particle, and swarm means the collection of particles. ACO is one of the famous meta-heuristics [DC99, BR03, Rai06, HS04, Ree93b] to solve CO problems e.g. Travelling Salesman Problem [SH96, DG97, JP], routing problem in computer network [CD11], protein folding problem [UM93]. There are other famous metaheuristic algorithms such as tabu search, simulated annealing which can solve and improve the performance of CO problems. Optimization is a function to make the best result by doing effective use of resources. For example, to find the shortest path from one city to another we need to find some optimization technique by minimizing the time or money. So, by combining the definitions, we can say, Particle Swarm Optimization (PSO) is a group-based method of solving any CO problems.

Many complete and approximate algorithms can solve CO problems. However, complete solution means we need to have the exact optimized solution whatever the time limit [WN99] is which is practically impossible sometime to give up the time limit that is why approximation algorithms got popular in recent 30 years to solve NP-hard problems [GJ79]. ACO is one of them which is inspired by the ants foraging behavior.

1.5 What is Travelling Salesman Problem

Traveling salesman problem(TSP) is classified as an NP-hard problem which states that there given a set of cities and a salesman needs to travel all the cities and come back to the city he started from

by minimizing distance or cost or both. Also, he can not visit a city twice except the starting city. We know the complexity of the TSP problem is NP-hard, for example, if a salesman is visiting n cities, there are $(n-1)!$ possible solutions which make it Np-hard. However, $P = NP$ is yet the biggest open problem to be solved in computer science.

TSP has several applications, such as,

- planning
- scheduling
- packing, logistics, and manufacturing
- integrated Circuits
- distributing goods

1.6 Artificial Ant's Foraging Behaviour

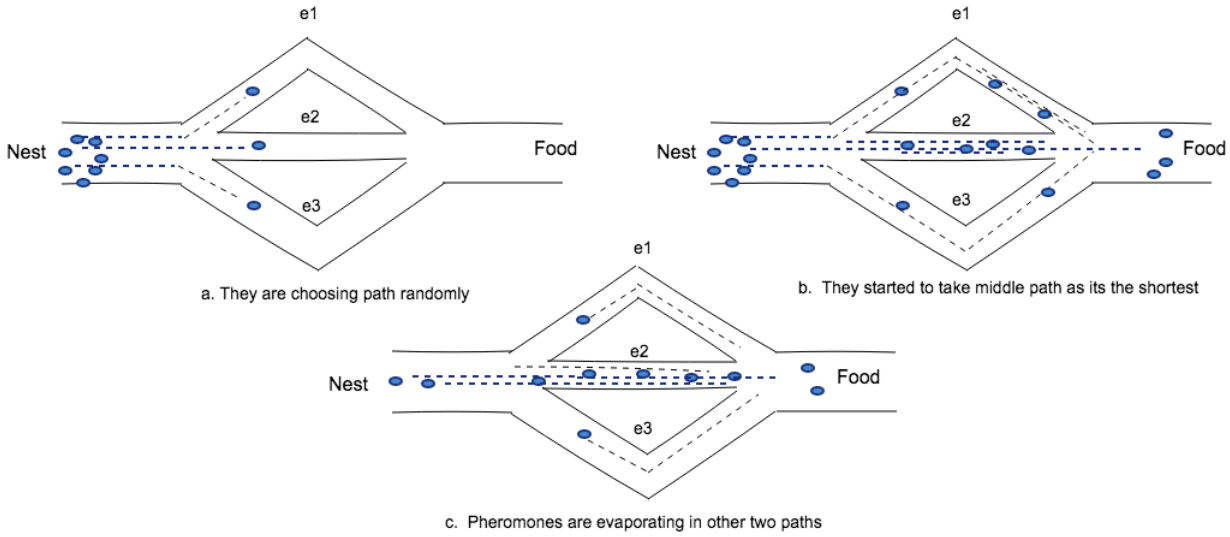

Figure 1.3: Ant's foraging behavior

ACO is a heuristics that help to solve many NP-hard problems as well as TSP. The whole analogy follows a simple graph G(V,E) model where V consists of two nodes (Nest, Food) from the figure 1.3 and E consists of three links between food and nest where $length(e_2) < length(e_1) < length(e_3)$ First, each ant randomly chooses a node as a start node. The goal is to bring the food to the nest. In figure 1.3(a), all ants start looking for food from their nest. As there are three paths and

yet no pheromone on the road each of them chooses path randomly. As they advance, they leave pheromone in the path. In the second section of figure 1.3(b) they have found the source of food and judged the quantity and quality of the food and started to return to their nest carrying food with them.

From the figure, edge e_2 is more intense as it is the shortest one. Even pheromone evaporation happens, still, it should have more pheromone than the other two edges. This time they will again leave pheromone on the road, the quantity of pheromone left on the road depends on the food. When they are going back, if an ant chooses an edge e_i, pheromone value on that edge $\tau_i, i = 1, 2, 3$ will change as follows, the amount that will be added depends on the length of the chosen path and Q. Q is a positive constant

$$\tau_i = \tau_i + \frac{Q}{length(e_i)}, i = 1, 2, 3 \tag{1.1}$$

Then they continue to build up their solution by adding edges which are not yet visited which

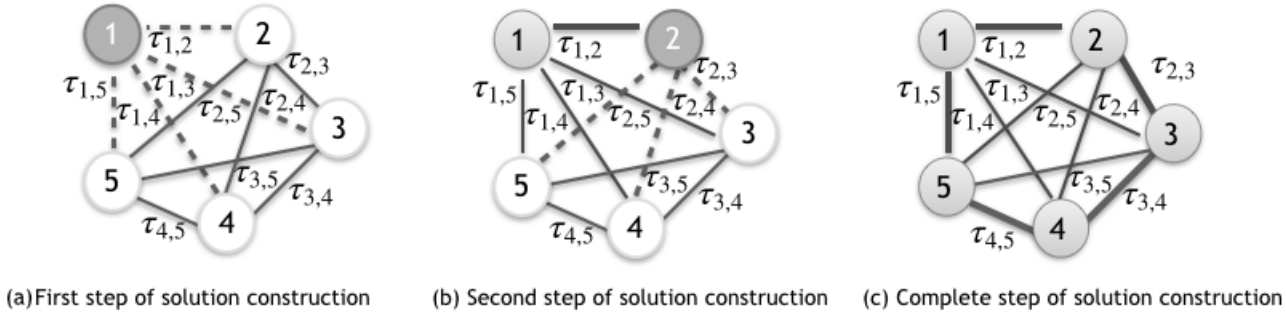

Figure 1.4: Solution Construction for TSP

is explained in another figure 1.4. For this, they have to keep a memory VISITED where all the visited edge information will be stored. Once an ant has visited all the nodes, it will return to the node it started. At each step, it will add the visited edge information to the solution. Whenever an ant is in any node, it will choose the next edge based on the probability of choosing a path, whichever path is rich in pheromone using the following formula

$$p_i = \frac{\tau_i}{\tau_1 + \tau_2 + \tau_3}, i = 1, 2, 3, \tag{1.2}$$

Other ants will track the pheromone smell and go to the food source which has been defined as 'stigmergy': indirect communication between ants [DAGP90] using pheromone trails. for this

particular example, $\tau_2 > \tau_1 > \tau_3$, so probability of choosing e_2 is higher. Pheromone also evaporates over time using the following formula,

$$\tau_i = (1 - \rho) * \tau_i, i = 1, 2, 3, \tag{1.3}$$

here, the parameter $\rho \in (0, 1]$ controls the pheromone evaporation. So whichever path is shortest tend to have more pheromone concentration left in the path. From the figure, we can see that the middle path is the shortest yet some ants chose the upper one, and few chose the lower one, but most of them chose the middle one following the pheromone smell. Eventually, upper and lower path's pheromone will fade away, and all ants will take the middle path which is the shortest one (c). The system will iterate, and per iteration same number of ants will be simulated.

differences between artificial ants and real ant:

From the above scenario, we will point out some major differences between real and artificial ant.

- Real ants move in an asynchronous way where artificial ants are synchronous.
- Artificial ants have memory, they knows the cities they have visited.
- Artificial ant deposits pheromone on its way back while real ants leave pheromone every time it moves.
- Foraging behavior of real ant is based on the fact that a shorter path will be completed soon and pheromone will be added quickly. As opposed to this artificial ants do some quality measure which determines the strength of pheromone during their return trip.
- Artificial ants are not blind; they prefer to choose the nearby city from their position as they know the distances between two cities

1.7 Parameters and Functions

ACO is a nice technique to solve many NP-hard problems but first it is necessary to find out the right parameters. ACO algorithm requires a long search. Initially edges contain different quantity of pheromone but after some time the edge which has taken more by the ants contains more pheromone. Thus, parameters setting have an important aspect to improve the performance. These parameters depend on the problem instance. In this paper [SS13], they gave detailed experiment

parameter name	definition	data type
iteration	how many times system will iterate	(int)
number of ants (n_a)	total number of ants participating	(int)
number of cities (n)	total number of cities to be visited	(int)
τ_{ij}	artificial pheromone value	(double)
p_{ij}	probability of choosing link i to j	(double)
η_{ij}	the visibility of city j from city i	(double)
d_{ij}	the distance between city i and j	(double)
f(s)	length of the solution (tour) s	(int)
α	the parameter to regulate the influence of τ_{ij}	(double)
β	the parameter to regulate the influence of η_{ij}	(double)
Q	positive constant	(int)
ρ	evaporation factor	(double)
initial city ($Nest$)	nest of ants	(int)

Table 1.1: Parameter's table

results for changes of parameter α, β, ρ, and Q. They mentioned as α increase, the more prominent plausibility of ants picking the path to have a place with the previous tour, on the new tour. In the ant system, α indicate the importance of collected pheromone on the edges. Again for increasing β, the possibility of ants choosing the local shortest path will increase the speed of convergence gets faster. Global optimization and fast convergence both are necessary to obtain an optimal solution. The algorithm should have a proper balance between them. According to them, The residue of pheromone left by ant on edge evaporates with time. The parameter ρ directly affect the global search ability and additionally arbitrariness. At the point when the number of nodes increases, the pheromones on the way which will never be crossed progressed toward becoming 0.

moreover, lastly, they concluded by saying the effect of amount Q to the performance depends on the three parameters (pheromone factor α, heuristic factor β and the residues coefficient of pheromone ρ)

1.8 Ant System for Solving TSP

We will explain the same scenario in 1.6 yet in a generalized manner to solve TSP. Given an n-city, complete undirected graph $G = (V, E)$ with distance matrix d_{ij}, the artificial ants are distributed to these n cities randomly. According the Ant System (AS) approach [DMC96], the edges of the TSP graph are the solution components, so each edge e_{ij} has a pheromone value τ_{ij}. First a node of TSP graph v_i is chosen randomly. Each ant will choose the next edge to visit according to the

nearby city and pheromone trail on the paths. Those edges which are being considered to visit an unvisitied city by ants will be added to the solution construction starting from the city it is located. when there is no unvisited city left the ant will go back to the city from where it started which implies that ants have a memory 'VISITED' where it stores the already visited nodes. so if an ant is in node v_i the next edge will be chosen based on the following probability in general:

$$p_{ij} = \frac{\tau_{ij}}{\sum_{k \in 1,...,|V| | v_k \notin VISITED} \tau_{ik}}, \forall j \in 1, ..., |V|, v_j \notin VISITED \tag{1.4}$$

Once all the ants in the colony completes their solutions, pheromone evaporation starts as follows:

$$\tau_{ij} = (1 - \rho) * \tau_{ij}, \forall \tau_{ij} \in P \tag{1.5}$$

here, P is the set of all pheromone values. Ants start their return trip and pheromone will be deposited by following formula:

$$\tau_{ij} = \tau_{ij} + Q/f(s) \tag{1.6}$$

where s is the solution set and f(s) is the objective function value of it. Simulation will go until a stopping condition is satisfied, applying n_a ants per iteration.

1.9 The ACO metaheuristic

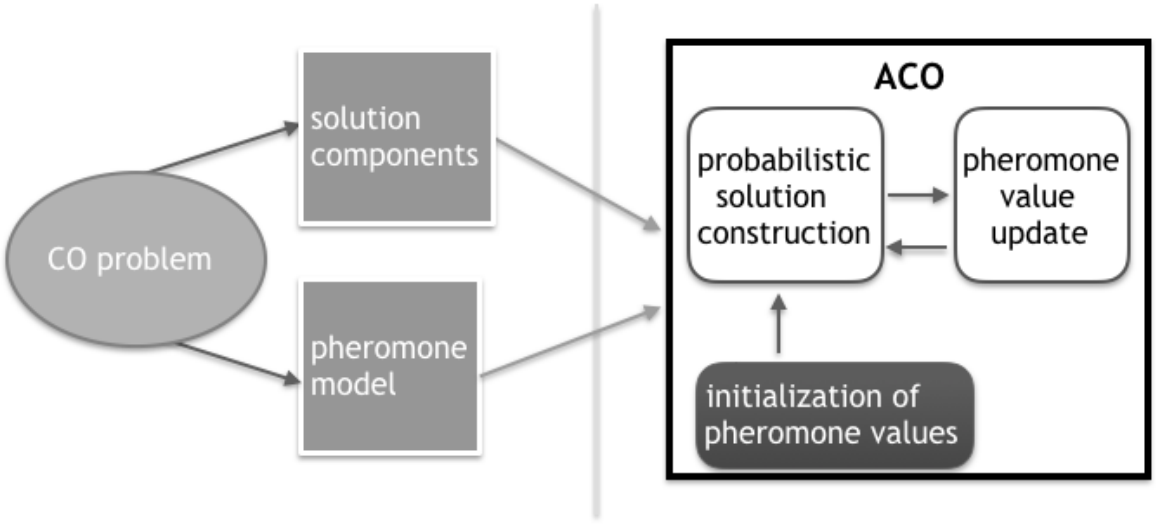

Figure 1.5: Basic Workflow of ACO

In 1999, Dorigo et al. [DBS06] first introduced the ACO metaheuristic, a general framework of ACO metaheuristic is given below: The basic workflow is mentioned in figure 1.5. From the figure,

we need solution components and pheromone model to solve a CO problem. Solution components are used to assemble solutions to the CO problem and Pheromone values is a set of values known as pheromone model. From a technical side, pheromone model is a parameterized probabilistic model which is the central component of ACO metaheuristic. The pheromone values $\tau_j \in P$ are normally associated with solution components, and pheromone model is used to generate the solution probabilistically. ACO follows two steps in general:

- Pheromone model is used to construct the solution which is a parameterized probability distribution over the solution space.
- Good solutions modify the pheromone values so that it gives the better solution in future.

Chapter 2

Flow shop Scheduling Problem

2.1 Introduction

Since the early '60s of last century researchers are working on developing heuristics to solve Flowshop Scheduling Problem (FSP) which falls under the NP-complete category. Johnson et al. first proposed the two and three-stage production schedules [Joh]. After that, many researchers have talked about many heuristics approaches to address this problem. As this is an NP-hard problem, we have to be more concerned about obtaining an economically sound and practical application of the solution than finding an optimal solution which led many other researchers as well as us to find another unconventional approximate method. There are many review papers on the heuristics [Gup79, PPE84]. By complexity and space, these heuristics and metaheuristics are different. One approach is to find a good solution by improving the initial sequence. In this thesis, we are approaching the Permutation Flowshop Scheduling Problem (PFSP), which is a special variant of FSP. Our approach got the inspiration from various papers [WH89, Moc95] where they improved the initial sequence using Tabu Search. [Ree93a, Tai90, Ree95] and we used two-opt technique and another novel technique.

2.2 Definition

This problem is so common that it is well defined even in Wikipedia [wik] where the definition is given as follows: Flow shop scheduling problems are a class of scheduling problems with a workshop or group shop in which the flow control shall enable an appropriate sequencing for each one of n job and processing on a set of m machines or with other resources $1, 2, ..., m$ in compliance with given processing orders.

2.3 Permutation Flowshop Scheduling Problem

Permutation Flowshop Scheduling Problem is a special type of FSP, defined as, A permutation flow shop is a flow shop in which jobs are processed in the same order on each machine [Bru04]. It has no intermediate storage. A job remains on its machine until the next machine is available. It has many applications in real life. For example,

- production facilities as to computing designs.
- Scheduling

Example

Jobs	J_1	J_2	J_3	J_4	J_5
M1 (p_{1,J_i})	5	5	3	6	3
M2 (p_{2,J_i})	4	4	2	4	4
M3 (p_{3,J_i})	4	4	3	4	1
M4 (p_{4,J_i})	3	6	3	2	5

Table 2.1: A schedule table for 4 machines and 5 jobs

Every job has some processing time for job i on machine j which is defined as $p_{ij}(i = 1, ...n; j = 1, ..., m)$. Table 2.1 shows the processing time table for four machines and five jobs. Our job is to find the sequence of jobs minimizing the maximum flow time or makespan. In our work, we are concentrating on reducing the makespan. For small data set optimal solution can be found even without a computer, or we can do complete enumeration, branch and bound techniques, integer programming to determine the optimal result but for a larger dataset, it becomes an NP-complete problem and heuristics are necessary to solve them. For example, in table 2.1 for the permutation order $0, 4, 1, 2, 3$ we got the makespan of 32 also known as C_{max} which is the optimal solution shown in figure 2.2. Another permutation schedule in which the jobs run in order for $0, 3, 1, 2, 4$ is shown in figure 2.1. Also, this gives the makespan of 37.

Sequencing Problem is numerical in Operational Research. Here we have multiple jobs which are to be processed on multiple machines. So this is an n-jobs m-machines problem. Sequencing problem emerges when we have different machines and numerous jobs to be handled in the best and ideal approach to spare time and cost.

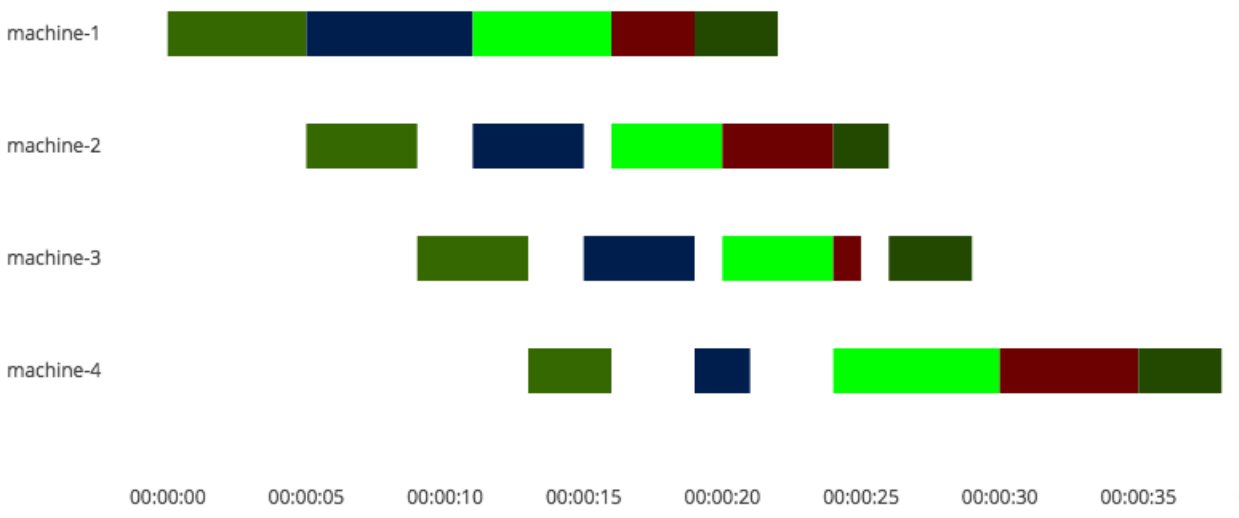

Figure 2.1: makespan for the permutation $0, 3, 1, 2, 4$

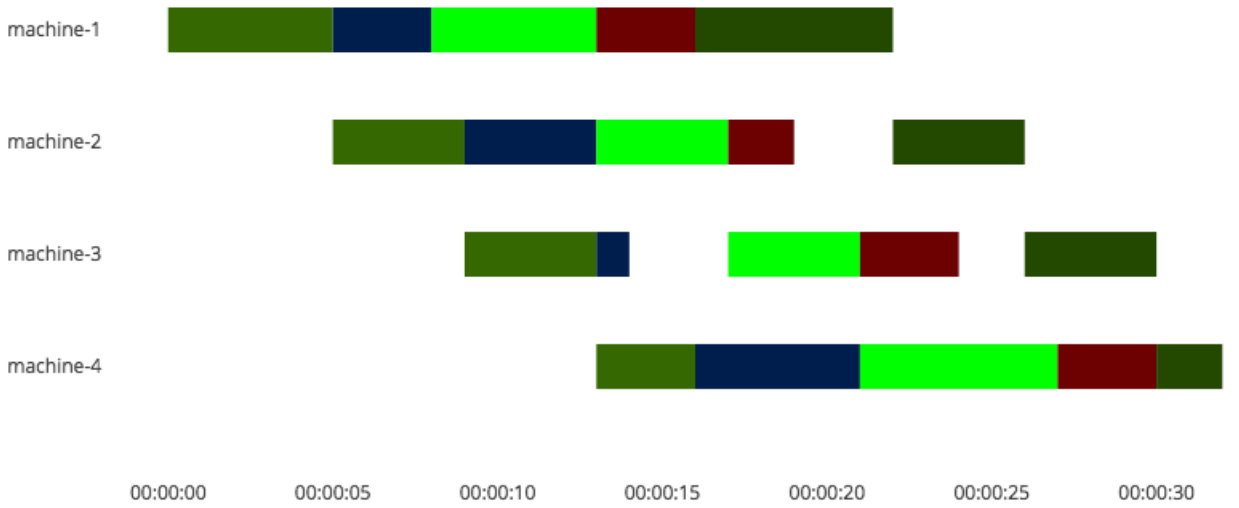

Figure 2.2: makespan for the permutation $0, 4, 1, 2, 3$

2.4 Previous Heuristics Procedure

Widmer and Hertz [WH89] described an efficient algorithm SPIRIT where they first compute the initial sequence using the analogy from the TSP and then applied Tabu Search technique to improve the solution. However, with more significant problems like for 100 jobs, SPIRIT did not perform well.

Nawaz et al. in his paper [NEH83] mentioned about another heuristic NEH where he explained jobs with higher total process time should be given more top priority than the tasks with less entire process time. NEH performs well for more massive problems.

Moccellin et al. [Moc95] defined FSHOPH query by incorporating a new heuristic algorithm

for the FSP which is pretty similar to SPIRIT [WH89]. The main difference between them was presented as in finding the initial solution for the scheduling problem. The distance between two jobs is entirely different in SPIRIT. They mainly focused on the feasibility condition between machine and job's idle time in the FSP. They used the Farthest Insertion Travelling Salesman Procedure (FITSP) to find out the good initial solution and then applied Tabu Search to improve it. For other significant problem, FSHOPH does not perform well either.

In this paper [AB99] they talked about routing-scheduling version of the flow-shop problem. However, So far NEH and SPIRIT outperform other known heuristics.

2.4.1 Constructive heuristics

According to this paper [Lou96], a constructive heuristic is an algorithm that builds a sequence of jobs, and once a decision is made, it is never changed. They claimed if the running time is an important factor, Sevast'yanov's algorithm can be an excellent alternative especially in the presence of substantial scale instances with a relatively small number of machines. However, the meaning of constructive somewhat confusing mentioned in [FGL04] besides being a rather coarse classification and hence of limited scope. Pinedo defines constructive heuristics as opposed to composite heuristics in [con96].

2.5 Workflow of Permutation Flowshop Scheduling Problem

In this portion, we will describe the workflow of Permutation Flow shop Scheduling Problem briefly in a generalized manner. In this review paper [FGL04] they tried to give a general framework by three phases in the following:

- Phase 1: Index Development
- Phase 2: Solution Construction
- Phase 3: Solution Improvement

2.5.1 Index Development

The output of this phase is an order which can be a solution or can be an input for the next phase for improvement. How the jobs will be arranged depends on the input data. In this heuristic [cam] FSP instance is created by machine aggregation from the data given by the original PFSP instance.

By employing Johnson rule, FSP can be solved in polynomial time which is accepted as a solution for the original PFSP. Gupta [Gup79] mentioned that if there is no problem analogy, a function to rank the jobs, it can be interpreted as sorting analogy. In [FGL04] they mentioned two issues to consider if any problem analogy is employed.

- We should find the polynomial methods to solve problem analogy to ensure the simplicity optimally. This implies the complexity of the analogy problem indirectly.
- The correspondence between solutions of the analogy and solutions of the original problem.

So far three problem analogies have been used [FGL04] to get an order for Flowshop scheduling problem.

$F_2||C_{max}$ -analogy

This is the most widely used analogy via machine aggregation. Here the first issue simplicity can be achieved by setting m = 2 for which $F_2||C_{max}$ is optimally solvable in polynomial time [Joh]. However, seems like it is difficult to develop machine aggregation heuristics for $m \geq 2$.

TSP analogy

Gupta [Gup79] first introduced the TSP analogy to solve the Flowshop scheduling problem. The Stinson and Smith in 1982 [SS82] presented six ways to develop distance matrix using the processing timetable. After that, Widmer and Moccellin also presented a solution with the same analogy. All these formulas are mentioned in the table 3.1

Vector summation analogy

Sevast'janov et al. first introduced the vector summation analogy [Sev95] and implemented [Lou96] where the author concluded saying, there is not necessarily a correspondence between good solutions concerning the maximum norm of the partial sums and good solutions of the $F_m|prmu|C_{max}$-problem [FGL04]

2.5.2 Solution Construction

Until we get a solution, unscheduled jobs will be inserted into different positions of a partial solution recursively or iteratively. Therefore, this phase consists of two issues at any iteration:

- **Job selection**, which will be done according to the tour obtained from phase one. Jobs will be inserted following a certain schedule measure. For example, minimizing the makespan of the resulting partial solution. It is easy to understand, minimizing some idle time may lead to a low makespan.

- **Job Insertion**, which needs to be done in a way so that it minimizes some property (makespan) of the resulting partial schedule. Researchers have tried many different positions for inserting jobs to improve the solution [SL93, AS]. Nawaz et al. and Woo and Yim tried to insert the job in all possible position [WY98, NEH83].

2.5.3 Solution Improvement

We follow two main criteria when we want to improve the current solution.

- a current solution should be there which we are trying to improve

- solution output, cost of the objective function should be better or equal than the current solution.

There have been many studies to improve the initial solution. In this paper Dannenbring et al. mentioned abstract computational experience with eleven flow shop sequencing heuristics where two jobs are exchanged [Dan77a]. Alternatively, a set of adjacent jobs are exchanged [Pag61]. Many others presented their work on local search and metaheuristics to improve the makespan. [KS80, NS96, HC91, St97]. There are some improvements done by the technique genetic algorithm [CVA95, Ree93a, Ree95]. In this papers [OP89, OS90], they talked about Simulated Annealing to improve the makespan by a general neighborhood approach.

In the review paper [FGL04], they tried to classify all the works under general and specific neighborhood. There are again two categories in each, Descending local search and Metaheuristic.

They concluded saying that choice of the general/specific neighborhood does not matter so much as we all know that the efficient implementation of meta-heuristic involves on its parameters; also we need to tune them well. Thus, parameter design is more important no matter how general or specific the techniques are.

Chapter 3

Our Approach

We tried to solve the PFSP [TB87, Dan77b, NEH83] using the reduction method with the analogy of the TSP. Later, we tried the two-opt technique and improved our solution.

3.1 Reduction

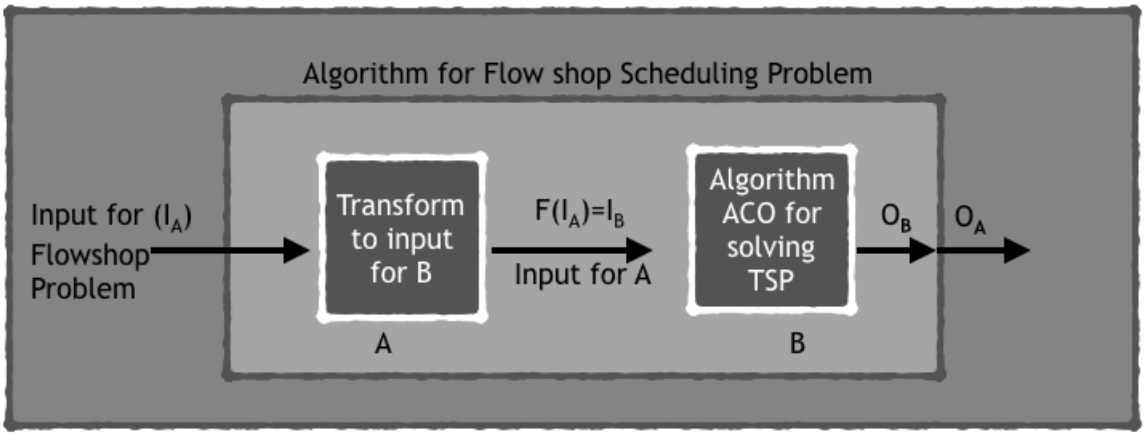

Figure 3.1: Reduction Technique

In the figure 3.1 we have our whole reduction technique mentioned. Here, we have a black box algorithm of ACO that can solve a known problem B which is TSP. Now we need to solve another problem A which is PFSP, and we do not know how to solve it. So here we will apply our reduction technique to solve B.

A = PFSP

B = TSP

I_A = Input for PFSP

I_B = Input for TSP

O_A = Output for PFSP

O_B = Output for TSP

$F(I_A)$ = Function that transform I_A to I_B

For this, the input of A needs to be transformed into the input of B, and this needs to be done at a polynomial time. Then when we get the output for B, we will transform that into an output for A in polynomial time. By this way Problem A can be reduced to problem B or in another word, Flowshop Scheduling Problem got reduced to the Travelling Salesman Problem. This technique is known as Reduction technique.

3.2 Transformation of Inputs

As we know from section 2 every job has a processing time and the processing time of each job is the input of PFSP. The processing time of job i on machine j is $P_{ij}(i = 1, ...n; j = 1, ..., m)$ like shown in table 2.1.

However, Our black box algorithm takes a different input as it solves a different problem. From the TSP definition explained in 1.8, we assumed it a complete graph. So we need a distance matrix to go from every city to others. If there are n cities, the input for this problem will be an n x n matrix. We need to convert $P_{ij}(i = 1, ...n; j = 1, ..., m)$ to a distance matrix C_{uv}, which is a n x n matrix.

3.3 Distance between two jobs

Framinan et al. in this paper [FGL04] mentioned about various distance metric used as heuristics to get a good solution. In table 3.1, all of them are accumulated.

We took the reference from the papers [WH89, SS82] and calculated our new distance which worked as the distance between two jobs. Distance is a function of processing times. All these functions mentioned in table 3.1, takes the processing time as input and gives us the distance matrix. We calculated our distance matrix using three different formulas from the table. We took the heuristics from Widmer et al.[WH89] and first two heuristics of Stinson and Smith out of six they mentioned in their paper [SS82]. In the paper of Widmer and Hertz, they have calculated the distance from job u to job v using the formula in equation 3.1 given that job v is scheduled after the job u. They claimed their distance formula is the most satisfactory one. The formula is given below

Reference	Distance matrix ($1 <= u, v <= n$)
Gupta(1968b)	$C_{uv} = (CT_m(u,v) - \sum_{j=1}^{m} p_{uj})$ where $CT_0(u,v) = 0$, and
	$CT_j(u,v) = max\{CT_{j-1}(u,v); \sum_{k=1}^{j} p_{vk}$
Stinson and Smith (1982)	$C_{uv} = \sum_{j=2}^{m} \lvert p_{uj} - p_{v,j-1} \rvert$
	$C_{uv} = \sum_{j=2}^{m} \{p_{uj} - p_{v,j-1}\}^2$
	$C_{uv} = \sum_{j=2}^{m} \lvert min\{p_{uj} - p_{v,j-1} + min\{(p_{u,j-1} - p_{v,j-2}); 0\}; 0\} \rvert$
	$C_{uv} = \sum_{j=2}^{m} \lvert p_{uj} - p_{v,j-1} + min\{(p_{u,j-1} - p_{v,j-2}); 0\} \rvert$
	$C_{uv} = \sum_{j=2}^{m} \{p_{uj} - p_{v,j-1} + min\{(p_{u,j-1} - p_{v,j-2}); 0\}\}^2$
	$C_{uv} = \sum_{j=2}^{m} max\{(p_{uj} - p_{v,j-1}); 0\} + 2\lvert min\{(p_{u,j} - p_{v,j-1}); 0\} \rvert$
Widmer and Hertz (1989)	$C_{uv} = P_{u1} + \sum_{j=2}^{m} (m-j) \lvert P_{uj} - P_{v,j-1} \rvert + P_{vm}$
Moccellin (1995)	$C_{uv} = UBX(m)_{uv}$ where $UBX(1)_{uv} = 0$, and
	$UBX(k+1)_{uv} = max\{0; UBX(k)_{uv} + (p_{vk} - p_{u,k+1})\}$

Table 3.1: Distance matrices constructed using TSP analogy

$$C_{uv} = P_{u1} + \sum_{k=2}^{m} (m-k) \lvert P_{uk} - P_{v,k-1} \rvert + P_{vm} \tag{3.1}$$

Jobs	J1	J2	J3	J4	J5	J6	J7
M1	5	5	3	6	3	1	3
M2	4	4	2	4	4	4	1
M3	4	4	3	4	1	6	2
M4	3	6	3	2	5	4	2

Table 3.2: A schedule table for 4 machine and 7 jobs table

2	2	4	4	2	6	5
2	2	4	4	2	6	5
7	7	3	9	3	3	4
2	2	4	4	2	6	5
5	5	3	7	5	9	2
4	4	6	6	4	8	6
10	10	4	12	6	2	5

Table 3.3: 7x7 distance matrix created from table 3.2 using Widmer formula

Take the table 3.2 as an example, this is the input matrix for the PFSP, which is 4x7 in dimension as this is for seven jobs running in 4 machines. Now we will need to transform this input into the input compatible with our approximation algorithm. As we have used three different formulas for the distance matrix to compare our result, all three of the distance matrices created from table 3.2 are shown. Table 3.3 shows the new distance matrix created using the formula in

2	2	3	3	3	6	5
3	3	6	4	6	3	8
5	5	2	6	4	5	4
3	3	4	4	2	7	4
5	5	4	6	8	7	4
3	3	6	4	6	7	8
8	8	3	9	5	6	3

Table 3.4: 7x7 distance matrix created from table 3.2 using Stinson's formula(1)

2	2	5	5	5	18	11
5	5	14	8	26	9	26
11	11	2	18	6	11	6
5	5	6	8	2	25	10
11	11	6	14	26	19	10
5	5	18	8	14	17	30
24	24	5	33	9	20	5

Table 3.5: 7x7 distance matrix created from table 3.2 using Stinson's formula(2)

equation 3.1. Table 3.4 and table 3.5 shows the distance matrices created by using the first two formulas of Stinson [SS82].

3.4 Run ACO on the new input (Distance Matrix)

Our approach to solving the PFSP is by using reduction. We first solved the TSP using the ACO approximation algorithm feeding our new distance matrix as input which will give us a tour. This tour will work as an initial solution for our PFSP. We have implemented the state of the art approach of Ant system by taking the reference from this paper [DMC96].

3.5 Two-opt techniques for improvement

Croes et al. suggested the two-opt technique [Cro58] which is used to improve the initial solution. The two-opt technique is mentioned in algorithm 4 We get the tour or permutation from the state of the art ACO approximation technique. Then at each step, we will take two random cities and will swap them to come up with a new tour and will check if it is improving the tour cost or not. In another word, as we are solving PFSP, we will check if the new order is giving us a shorter makespan. Generally, the idea is to repeat this technique until no improvement is made and save the iteration number. In our experiment, at first, we have defined a stopping rule to save some

computation time. We have chosen a fixed number of iteration. As long as we are not reaching the iteration, we will keep doing Two-opt. This technique has undoubtedly improved our solution. Moreover, to the best of our knowledge, no one has applied the two-opt method to improve their solution as us. There are other many improvement techniques for PFSP mentioned in section 2.4. We have introduced another novel technique to develop our initial solution in the following section 3.6.

3.6 Complete new technique for improvement

We first ran the two-opt technique to improve our solution. This technique has inspired us to come up with a completely new technique which surprisingly improved the solution's makespan better. In the two-opt technique, we only swap two positions for the improvement. In our novel technique, we will keep changing the sequence on the changed sequence got from the previous iteration. The results shown in the experiments section will surely appreciate the approach. This new technique will slightly change the algorithm 2. The new algorithm is shown in algorithm 5.

In Algorithm 2, we ran the ant system algorithm on new distance matrix and got the initial solution. Then we did two-opt technique which switches two jobs randomly. For example, For the input data in table 2.1 We got the initial solution as $0, 3, 1, 2, 4$. After running the two-opt technique on this sequence, we got different sequence $0, 4, 1, 2, 3$ which gave us the optimal solution in our setting, the lowest $C_{max} = 32$ shown in figure 2.2 We ran the two-opt technique for certain iterations as shown in algorithm 2 However, the number of permutation is huge $n!$. Therefore, we kept an iteration of 10 initially. Later, increasing the iteration number gave us a better solution which is also reported in the experiment section.

3.7 Algorithmic Form

while *termination conditions not met* **do**
  AntBasedSolutionConstruction()
  PheromoneUpdate()
end

Algorithm 1: Ant Colony Optimization (ACO)

There are other heuristic functions; For using each of these, we get a different solution. So, we can consider each of them as different strategies. We have compared our results for other equations

Input : n : number of jobs
m : number of machines
P_{ij} : Processing time of job i on machine j where $i = 1, ..., n$, $j = 1, ..., m$
iter : iteration number
Output: S : sequence that makes the makespan smallest
C_{max} : length of the makespan

$initial\_sequence \leftarrow ACO\_TSP(n, m, P_{ij})$
while $iteration < iter$ **do**
  $iteration \leftarrow iteration + 1$
  $new\_sequence \leftarrow TWO\_OPT\_SWAP(initial\_sequence)$
  if $f(new\_sequence) < C_{max}$ **then**
    $C_{max} \leftarrow f(new\_sequence)$
    $S \leftarrow new\_sequence$
  end
end

Algorithm 2: Main structure of the algorithm

1: **procedure** $ACO\_TSP(m, n, P_{ij})$
2: $C_{ij} \leftarrow distance(P_{ij})$
3: $S \leftarrow Ant\_System(m, n, C_{ij})$
4: **return** S
5: **end procedure**=0

Algorithm 3: ACO_TSP Procedure

1: **procedure** TWO_OPT_SWAP(S)
2: $i \leftarrow Random()$
3: $j \leftarrow Random()$
4: $temp \leftarrow S[i]$
5: $S[i] \leftarrow S[j]$
6: $S[j] \leftarrow temp$
7: **return** S
8: **end procedure**

Algorithm 4: Two_Opt_Swap Procedure

Input : n : number of jobs
m : number of machines
P_{ij} : Processing time of job i on machine j where $i = 1, ..., n$, $j = 1, ..., m$
iter : iteration number
Output: S : sequence that makes the makespan smallest
C_{max} : length of the makespan

$initial\_sequence \leftarrow ACO\_TSP(n, m, P_{ij})$
while $iteration < maxiter$ **do**
  $iteration \leftarrow iteration + 1$
  new_sequence ← TWO_OPT_SWAP(new_sequence)
  if $f(new\_sequence) < C_{max}$ **then**
    $C_{max} \leftarrow f(new\_sequence)$
    $S \leftarrow new\_sequence$
  end
end

Algorithm 5: Main structure of the algorithm for further improvement

too reported the experiment results. However, none of them, being heuristics for solving NP-hard problems, guarantee the optimal solution.

Chapter 4

Implementation and Experiments

In this segment, we will assess the execution of our proposed solution for Flowshop Scheduling Problem, and we will compare it with the best solution (upper and lower bound) reported in this technical report [Tai93] We ran all our experiments in Mac book Air Processor 1.6 GHz Intel Core i5 and Memory 8 GB 1600 MHz DDR3. We have actualized the cutting edge ACO algorithm in C++ language and compiled it in the C-lion compiler. We have evaluated our algorithm using the dataset of scheduling instances mentioned in this technical report [Tai93]

4.1 Dataset: Processing time generation

In the report [Tai93], they defined Flowshop Sequencing Problem as follows, The flow shop problems are characterized by the processing times P_{ij} of job j on machine i ($1 \leq i \leq m, 1 \leq j \leq n$) and they have generated the values or dataset in following way by generating a random number: They

while $i = 1tom$ **do**
  while $j = 1ton$ **do**
    $d_{ij} \leftarrow U[1, 99]$
  end
end

Algorithm 6: Generate processing time

created data sets using a uniform random number (1,99) for 5, 10 and 20 machines and from 20 to 500 jobs and computed the lower and upper bound of the makespan. We compared our result against the data sets mentioned here. Moreover, some small data we created by ourselves for

testing.

4.2 Parameter Setup

First, we did some parameter analysis for the ACO algorithm. We took the reference from Dorian et al. [GC05] where they tried to introduce automatic learning of the optimal parameters for a given TSP instance. We mostly followed the paper of Dorigo et al. [DMC96] where he mentioned the interdependencies of parameters on the problem for the quick convergence to the best possible solution. After getting inspired from the paper [SS13] We kept the parameter values as follows $\alpha = 1, \beta = 3, \rho = 0.5$, and Q=10.

4.3 Implementation of ACO for the Travelling Salesman Problem

Used the Ant Colony Optimization(ACO) meta-heuristics to implement the Np-hard Travelling Salesman problem. We have followed the algorithm mentioned in [DBS06] and applied it in C++ which takes the Euclidean distance matrix as input. This is a fully connected graph, so salesman can go to any city from his place but can not visit a city twice according to the definition of TSP. This will give us a tour of the shortest distance or cost.

4.4 Result

As mentioned, We have used the reduction method to come up with a solution for the PFSP. We have a black box that can solve the Travelling salesman problem in polynomial time. We are using that known solution to address the unknown flowshop problem in polynomial time. For this, the input of Flowshop problems needs to be transformed into the input of the Travelling Salesman Problem. Moreover, for this, we have built a new distance matrix from the processing time matrix using a different formula from the table 3.1

Jobs	J1	J2	J3	J4	J5	J6	J7	J8	J9
M1	5	5	3	6	3	1	3	9	3
M2	4	4	2	4	4	4	1	5	4
M3	4	4	3	4	1	6	3	2	7
M4	3	6	3	2	5	4	2	6	5
M5	5	1	3	7	6	7	8	9	1
M6	6	3	5	2	3	4	4	3	2

Table 4.1: A schedule table for 6 machine and 9 jobs table

To calculate the best and worst solution we have ran a Brute force algorithm. We have computed all the possible orders and makespans for each of them. We found the order which has the minimum and the maximum makespan. Then we have compared the result with our solution. Brute force is not natural for more significant problems, So here we explaining our idea for the smaller problem first.

For 7 jobs 4 machines

For the table in 3.2 where we have four machines and seven jobs, we first converted the input set to another input set compatible for ACO algorithm using Stinson's formula(1). The new input set is shown in table 3.4. Then ran the ACO algorithm using this new distance matrix and we got the result of the open TSP tour as follows $0, 3, 1, 5, 2, 4, 6$ which is a permutation result for the flowshop problem with a makespan of 44.

We will run the same algorithm now using Stinson's distance formula 2. We got this open TSP tour $0, 4, 2, 5, 1, 3, 6$ which again a permutation for the flow shop problem with a makespan of 39.

Let us do the same experiment for Widmer formula as well now. Which gives a tour $0, 4, 6, 5, 1, 3, 2$ and a makespan 39.

Now, we want to improve our result using the two-opt technique and our novel technique. We know the current best makespan for this example is 39.

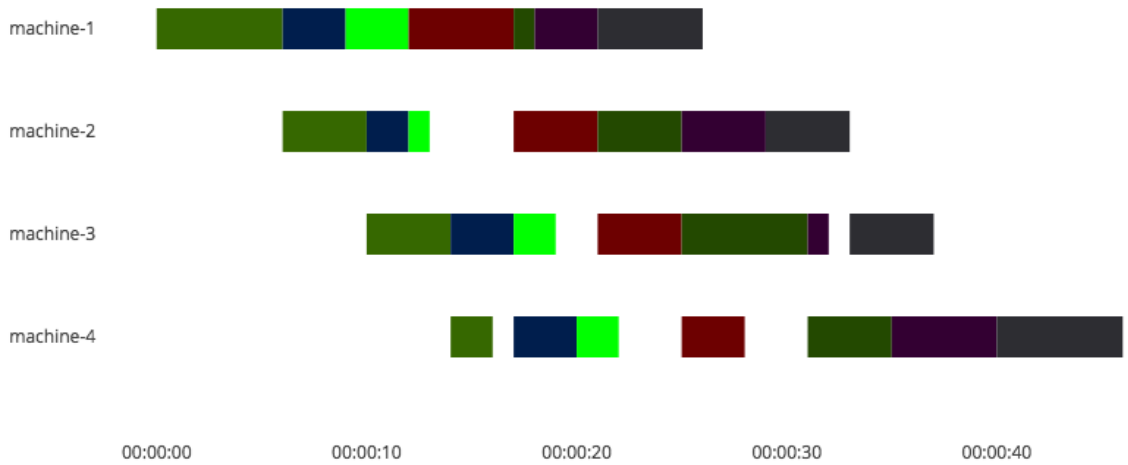

Figure 4.1: Solution $2, 3, 6, 0, 5, 1, 4$ with the worst makespan = 46 for table 3.2

We also have the best and worst result in hand which we got after running the brute force algorithm.

In figure 4.1, shows the worst solution which gives a order $2, 3, 6, 0, 5, 1, 4$ and a makespan 46.

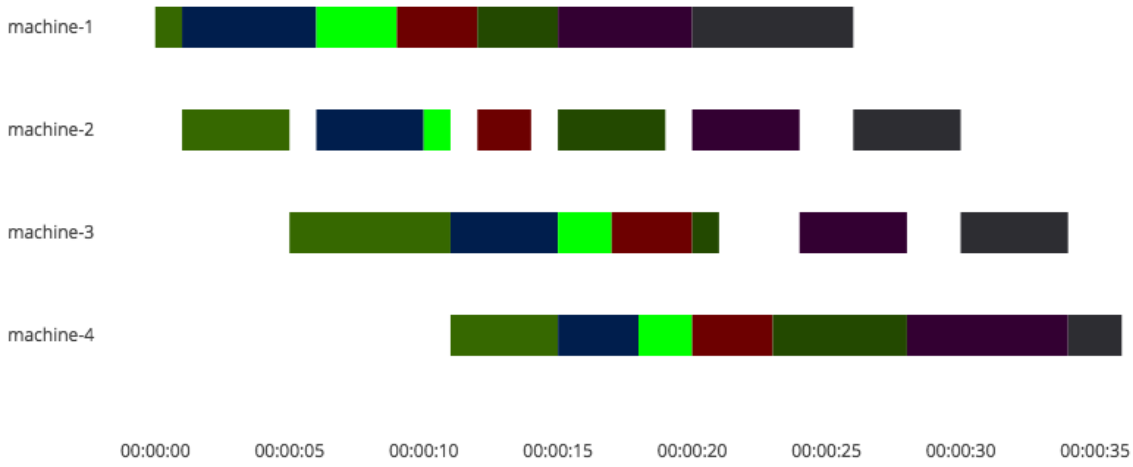

Figure 4.2: Solution $4, 6, 5, 1, 0, 2, 3$ with best makespan $= 36$ for table 3.2

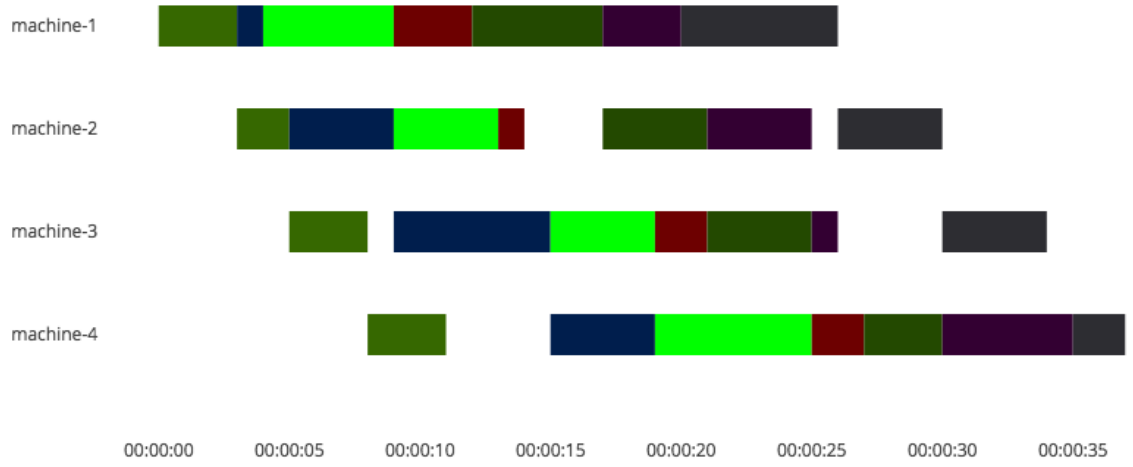

Figure 4.3: Solution $1, 4, 0, 6, 3, 5, 2$ with makespan $= 36$ for table 3.2

And in figure 4.2, shows the best solution which gives a order $4, 6, 5, 1, 0, 2, 3$ and a makespan 36.

Our solution did not get the best solution, but it is pretty close to the best solution and quite far from the worst solution.

We still have room for improvement. So, we will set current makespan as 39 and run some iterations continuing with the two-opt technique to see if it is improving the makespan.

In each iteration, if we get a new order which is minimizing the makespan, we are saving that order with the new makespan value. If it is not reducing, we are keeping the current solution.

The iteration number is a variable. We have tried increasing the iteration number until we get a good solution which comes with a cost of computation. However, it certainly helped in improving the solution.

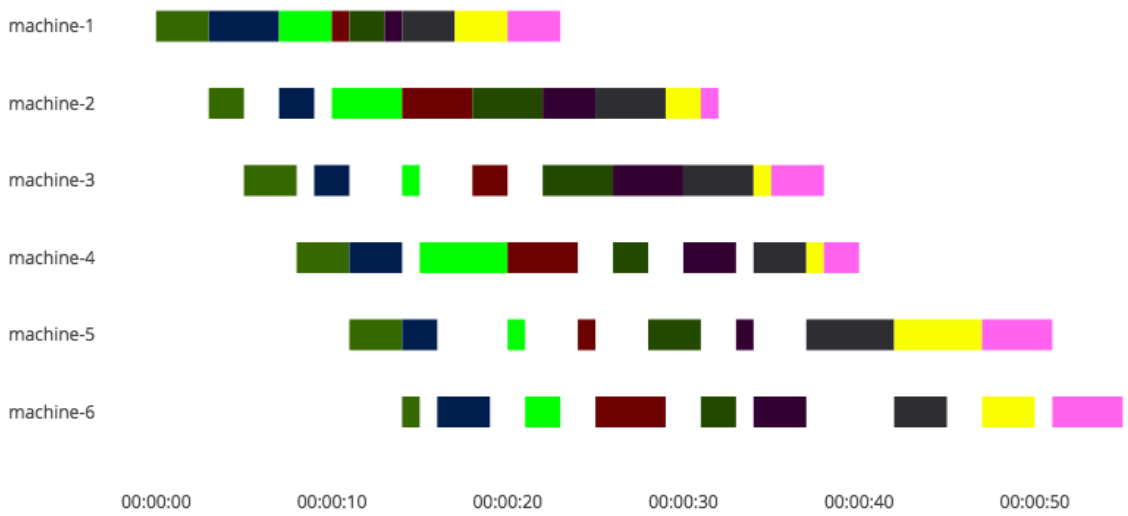

Figure 4.4: This is the solution with worst makespan for table 4.1

So, We have included the iteration value as part of our solution.

After running about 6000 iterations, We got a new order which gives us a makespan of 36 same as the best solution. Figure 4.3 shows our solution.

There can be multiple solutions which give the best solution. So there is no definite solution.

For 9 jobs 6 machines

We compared our solution for another example of 9 jobs and six machines where we got a satisfactory result again. For nine jobs we can have 362880 permutations. We generated all of them and found one of the best solutions and one of the worst solutions shown respectively in figure 4.5 and 4.4

So our solution after improvement gave quite a satisfactory result in figure 4.6

4.5 Drawing of Makespan

We have drawn few makespans to make a clear understanding with visualization. Here we have presented the worst and best solution as well for two examples.

For drawing the makespan figure, we used Pythons Plotly.

4.6 Deviation

Deviation formula is defined as follows:

$$deviation(\%) = 100 * (our\_soln - opt)/opt \tag{4.1}$$

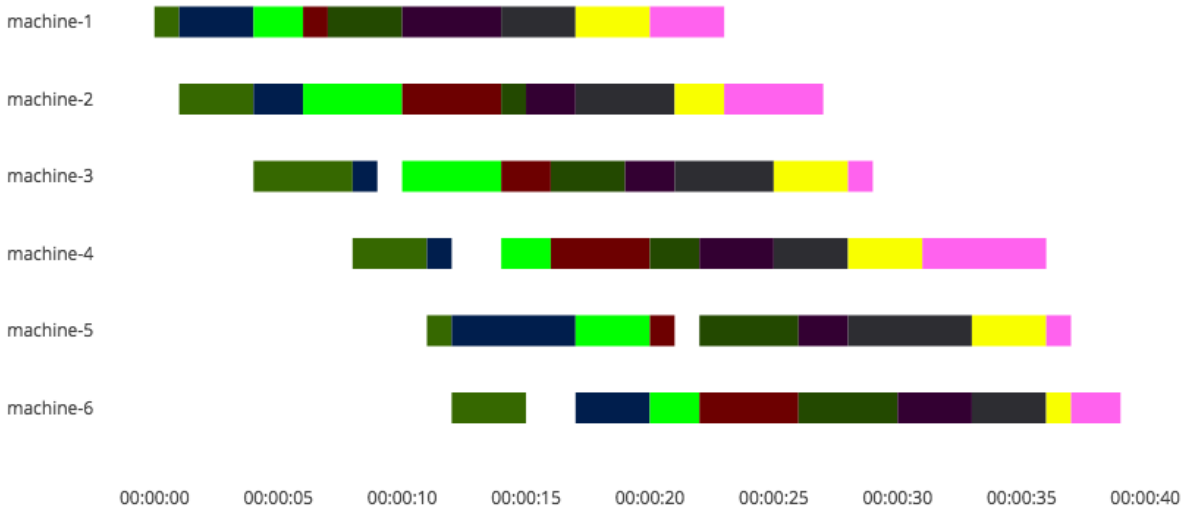

Figure 4.5: This is the solution with best makespan for table 4.1

where, our_soln = soln we got from our algorithm

Opt = solution that gives minimum makespan

Moreover, we can also say, for this particular example deviation(%) is also 0.

4.7 Result Analysis

For other examples of 20, 50, 100 200 jobs it was difficult to calculate the best and worst solution result using brute force technique. So we for those examples we compared our result against the lower and upper bound mentioned in the report. For this particular example sets, the optimal results are reported as the upper bound [Tai93].

4.7.1 Smaller Dataset

We have classified our experiments for smaller datasets and bigger datasets. We are considering following as smaller datasets:

- ***20 jobs 5 machines:*** Optimal makespan: 1278

  Stinson1: 0,1,14,5,4,17,6,19,3,9,13,15,8,7,18,12,11,10,2,16 **makespan:** 1514

  Stinson2: 0,1,12,19,11,10,8,14,13,15,5,18,7,16,2,6,3,4,17,9 **makespan:** 1395

  Widmer: 0,15,14,5,4,6,11,10,8,7,18,12,9,19,17,3,1,13,16,2 **makespan:** 1465

  Improvement1: 8, 16, 9, 6, 3, 14, 10, 18, 5, 11, 12, 2, 1, 4, 19, 17, 0, 13, 15, 7 **makespan:** 1317

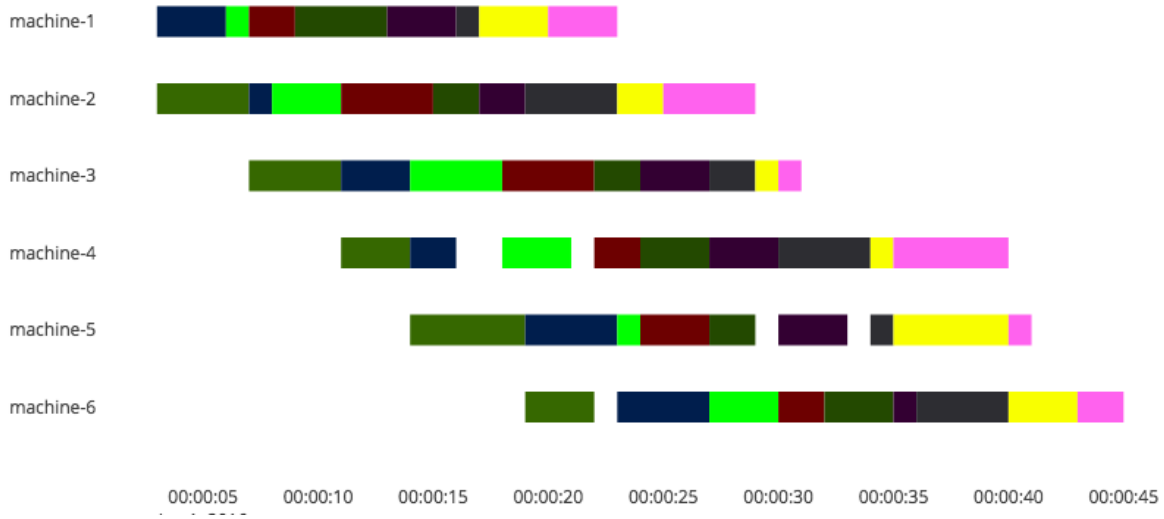

Figure 4.6: This is the solution our algorithm has found for table 4.1

Improvement2: 1, 6, 15, 2, 3, 16, 17, 10, 13, 19, 4, 11, 5, 18, 7, 12, 14, 8, 0, 9 **makespan:** 1324

Deviation: 3%

- ***20 jobs 10 machines:*** Optimal makespan: 1582

  Stinson1: 0,5,18,4,2,16,6,1,8,12,11,9,10,13,7,19,17,3,14,15 **makespan:** 1907

  Stinson2: 0,5,18,14,11,9,10,17,3,6,4,8,19,2,16,13,1,15,7,12 **makespan:** 2087

  Widmer: 0,5,18,14,17,4,7,12,19,2,16,13,11,10,6,8,9,3,1,15 **makespan:** 1887

  Improvement1: 2, 17, 13, 11, 14, 8, 3, 6, 1, 15, 7, 16, 19, 4, 0, 12, 5, 10, 9, 18 **makespan:** 1760

  Improvement2: 5, 4, 0, 3, 9, 11, 17, 8, 13, 6, 1, 14, 10, 12, 15, 16, 2, 19, 7, 18 **makespan:** 1758

  Deviation: 11%

- ***20 jobs 20 machines:*** Optimal makespan: 2297

  Stinson1: 0,12,7,6,8,1,2,14,13,9,19,3,16,10,11,18,17,4,5,15 **makespan:** 2762

  Stinson2: 0,4,9,19,3,14,18,15,17,13,2,16,6,8,1,11,12,10,5,7 **makespan:** 2549

  Widmer: 0,11,13,14,19,17,12,7,8,4,9,15,5,6,18,2,10,1,16,3 **makespan:** 2649

  Improvement1: 1, 17, 4, 14, 2, 8, 11, 10, 3, 0, 19, 6, 13, 15, 7, 5, 16, 9, 12, 18 **makespan:** 2476

Improvement2:17, 8, 3, 5, 9, 19, 16, 10, 2, 6, 14, 1, 7, 0, 11, 13, 18, 4, 12, 15 **makespan:** 2465

Deviation: 7%

- ***50 jobs 5 machines:*** Optimal makespan: 2724

  Stinson1: 0,31,49,39,12,46,27,24,42,7,41,15,13,47,19,11,36,35,8,1,10,20, 26,4,32,28,33,29,3,34,22,16,23,2,43,37,25,40,21,14,18,9,45,30,38,48,6,5,17,44 **makespan:** 3162

  Stinson2: 0,31,49,2,12,46,15,34,22,30,38,48,9,23,39,41,18,16,45,25,21,24, 42,20,43,19,11,5,17,27,44,13,14,6,36,35,37,29,10,8,47,1,33,40,3,28,26,4,32,7 **makespan:** 2986

  Widmer: 0,31,37,23,40,27,14,20,24,46,7,41,18,38,48,42,4,34,36,2,12,22,9,30, 29,10,21,32,19,1,33,3,28,15,43,47,16,11,5,35,39,13,44,6,45,25,17,49,26,8 **makespan:** 3113

  Improvement1: 12, 48, 28, 5, 9, 3, 31, 33, 34, 26, 39, 22, 21, 23, 46, 49, 7, 37, 11, 4, 8, 18, 30, 32, 27, 38, 44, 35, 14, 17, 10, 47, 19, 41, 42, 0, 6, 40, 25, 15, 1, 16, 20, 2, 13, 29, 24, 45, 43, 36 **makespan:** 2855

  Improvement2: 25, 36, 22, 5, 15, 13, 38, 45, 47, 0, 37, 35, 11, 7, 34, 32, 33, 4, 29, 1, 42, 19, 49, 24, 14, 2, 48, 8, 44, 23, 17, 40, 3, 39, 20, 21, 41, 28, 31, 9, 6, 30, 12, 16, 18, 46, 10, 43, 26, 27 **makespan:** 2856

  Deviation: 7%

- ***50 jobs 10 machines:*** Optimal makespan: 3025

  Stinson1: 0,13,43,29,17,19,41,6,36,21,18,16,31,7,9,47,44,10,11,5,32,1,20, 40,8,27,34,28,39,22,46,15,3,2,35,25,24,37,45,48,23,49,26,30,14,4,12,33,42,38 **makespan:** 3657

  Stinson2: 0,42,33,29,37,2,46,36,6,30,28,19,11,5,25,32,1,13,43,9,27,34,38, 47,26,48,10,3,7,22,8,12,40,49,15,18,45,35,39,14,20,44,21,17,31,41,4,16,24,23 **makespan:** 3558

  Widmer: 0,19,28,33,26,6,49,27,34,43,3,7,22,24,23,30,1,17,11,5,25,13,29,18, 16,2,37,15,39,38,36,42,41,8,4,48,31,12,35,45,14,44,46,20,47,21,40,10,9,32 **makespan:** 3697

  Improvement1: 15, 26, 22, 4, 5, 33, 37, 12, 27, 10, 29, 47, 39, 28, 36, 31, 21, 38, 41, 13, 1, 16, 44, 35, 3, 49, 11, 43, 46, 34, 9, 23, 14, 8, 30, 7, 32, 2, 0, 25, 6, 42, 45, 20, 48, 19, 17, 24, 40, 18 **makespan:** 3452

  Improvement2:30, 24, 13, 48, 34, 1, 21, 4, 12, 46, 19, 7, 28, 6, 25, 42, 43, 3, 31, 17, 49, 27, 35, 14, 26, 36, 32, 0, 10, 5, 47, 44, 33, 22, 15, 16, 9, 45, 11, 38, 23, 37, 41, 18, 20, 39, 40, 8, 2,

29 **makespan:** 3355

Deviation: 10%

- ***50 jobs 20 machines:*** Optimal makespan: 3875

  Stinson1: 0,23,29,9,37,32,7,30,45,5,38,34,20,26,42,8,44,41,28,3,48,14,18,
  21,47,6,36,46,2,11,25,10,49,4,33,16,24,19,27,31,15,13,1,40,43,35,12,39,22,17 **makespan:** 4672

  Stinson2: 0,15,12,3,8,29,23,14,43,13,37,45,2,44,7,30,38,46,34,4,16,49,48,
  39,31,17,21,25,27,26,19,28,24,22,41,32,18,35,47,9,33,20,5,6,10,36,40,11,42,1 **makespan:** 4939

  Widmer: 0,20,7,1,3,47,29,35,9,43,28,11,34,5,46,22,31,17,32,49,27,10,19,18,
  45,38,39,12,36,4,25,8,21,40,42,33,16,2,15,23,48,41,26,37,6,14,44,30,24,13 **makespan:** 4809

  Improvement1: 32, 49, 27, 2, 17, 7, 10, 29, 20, 5, 44, 19, 12, 13, 38, 40, 37, 18, 8, 46, 30, 16, 6, 39, 48, 4, 0, 9, 35, 3, 11, 23, 33, 25, 26, 34, 15, 41, 36, 47, 14, 24, 42, 22, 45, 28, 21, 31, 1, 43 **makespan:** 4481

  Improvement2: 12, 9, 13, 4, 18, 41, 22, 24, 44, 48, 14, 29, 17, 0, 23, 36, 39, 21, 34, 16, 20, 15, 28, 25, 3, 31, 10, 32, 6, 8, 45, 42, 40, 37, 38, 26, 49, 1, 47, 46, 35, 30, 27, 33, 7, 5, 19, 2, 43, 11 **makespan:** 4506

  Deviation: 15%

Id	m	n	upperbound	lowerbound	our solution	iteration
1	20	5	1278	1232	1424	20000
2	20	5	1359	1290	1504	20000
3	20	5	1081	1073	1313	20000
4	20	5	1293	1268	1552	20000
5	20	5	1236	1198	1390	20000
6	20	5	1195	1180	1381	20000
7	20	5	1239	1226	1347	20000
8	20	5	1206	1170	1451	20000
9	20	5	1230	1206	1396	20000
10	20	5	1108	1082	1326	20000

Table 4.2: Results for 5 machine and 20 jobs table

4.7.2 Larger Dataset

We have experimented against larger datasets as well in the following:

Id	m	n	upperbound	lowerbound	our solution	iteration
1	50	10	3025	2907	3573	15000
2	50	10	2892	2821	3479	15000
3	50	10	2864	2801	3585	15000
4	50	10	3064	2968	3753	15000
5	50	10	2986	2908	3655	15000
6	50	10	3006	2941	3538	15000
7	50	10	3107	3062	3580	15000
8	50	10	3039	2959	3465	15000
9	50	10	2902	2795	3481	15000
10	50	10	3091	3046	3669	15000

Table 4.3: Results for 10 machine and 50 jobs table

- ***100 jobs 5 machines:*** Optimal makespan: 5493

  Stinson1: 0,92,73,66,27,20,96,7,33,94,45,4,98,89,24,88,78,47,1,15,65,60,
  23,84,26,42,17,58,55,83,9,18,95,30,62,48,56,13,28,74,32,69,79,61,50,8,80,40,77
  ,2,86,22,90,5,63,12,43,97,37,36,75,68,52,29,49,6,91,53,64,71,46,35,11,19
  ,39,44,14,76,82,70,38,10,31,21,57,3,85,16,51,67,34,59,25,81,54,93,87,72,99,41 **makespan:** 5855

  Stinson2: 0,61,15,65,60,82,70,92,18,58,55,83,9,19,39,44,77,2,4,64,71,95,
  84,30,79,33,43,16,10,62,78,17,29,49,67,81,63,97,75,45,25,98,89,24,14,90,22,20,96
  ,32,91,54,86,59,42,47,1,94,76,34,52,31,11,87,3,85,7,12,6,13,41,21,37,36,8
  ,80,69,56,35,23,48,73,66,27,28,74,57,26,40,51,53,93,38,68,46,88,5,99,72,50 **makespan:** 5809

  Widmer: 0,84,26,42,56,17,64,79,41,21,37,5,55,58,49,24,36,3,29,59,65,6,66,
  27,90,50,75,63,7,44,12,67,86,89,54,82,70,92,73,13,28,74,32,33,60,81,9,38,
  10,87,52,98,30,69,2,35,23,61,11,14,72,97,71,20,25,95,83,18,16,45,4,19,39,
  94,76,1,15,78,46,93,31,48,40,88,22,47,8,91,80,85,99,43,34,57,77,53,68,96,51,62 **makespan:** 5911

  Improvement1: 23, 95, 74, 30, 66, 94, 59, 89, 63, 44, 5, 70, 22, 60, 92, 83, 43, 57, 6, 19, 91, 98, 11, 62, 16, 42, 0, 52, 38, 45, 81, 10, 26, 56, 40, 86, 18, 87, 79, 55, 93, 64, 3, 36, 71, 53, 49, 65, 34, 41, 85, 35, 67, 90, 21, 13, 20, 2, 80, 69, 31, 7, 76, 47, 82, 61, 4, 51, 9, 39, 32, 15, 75, 37, 25, 27, 77, 99, 24, 88, 73, 96, 50, 46, 68, 12, 29, 28, 54, 8, 1, 97, 58, 33, 17, 78, 84, 48, 14, 72 **makespan:** 5657

  Improvement2: 73, 31, 22, 37, 13, 99, 39, 33, 98, 27, 19, 1, 9, 4, 43, 62, 88, 61, 72, 26, 58, 24, 79, 21, 51, 97, 8, 84, 93, 96, 15, 94, 77, 11, 86, 95, 3, 81, 92, 75, 63, 28, 10, 66, 90, 29, 12, 64, 30, 2, 60, 76, 45, 70, 7, 32, 16, 5, 74, 23, 34, 47, 17, 18, 36, 65, 46, 69, 52, 67, 40, 38, 59, 48, 68, 41, 25, 91, 82, 80, 89, 87, 54, 55, 71, 35, 85, 49, 53, 83, 42, 6, 78, 14, 56, 20, 50, 57,

44, 0 **makespan:** 5696

Deviation: 2.9%

- ***100 jobs 10 machines:*** **Optimal makespan:** 5770

  Stinson1: 0,31,48,10,33,44,17,77,45,3,2,91,18,52,29,58,64,92,15,6,99,25,
  41,79,88,62,34,76,94,13,83,38,4,35,90,55,86,46,54,51,30,20,72,93,21,65,70,
  87,69,27,98,53,61,47,73,32,26,96,22,24,14,39,28,75,85,8,71,11,36,59,95,68,
  89,37,74,78,80,19,5,56,16,12,82,50,97,1,43,42,7,57,60,84,49,67,40,9,63,
  66,23,81 **makespan:** 6830

  Stinson2: 0,87,53,98,61,58,36,39,82,54,32,95,89,85,52,99,10,19,5,84,70,96,
  18,29,25,11,35,76,48,80,40,49,20,14,77,73,88,42,63,90,66,38,30,41,62,12,
  1,4,47,55,93,45,68,13,64,37,59,33,9,2,46,72,91,56,75,83,78,3,22,67,21,79,
  60,34,31,65,86,51,44,17,16,27,24,23,97,28,7,50,43,69,92,94,57,26,8,71,6,
  81,15,74 **makespan:** 6797

  Widmer: 0,31,47,79,87,29,25,12,99,78,66,85,9,90,23,97,37,68,15,58,55,82,
  46,30,40,44,48,77,2,65,7,10,33,76,19,28,13,49,5,6,59,95,96,54,81,14,39,73,
  88,35,80,86,1,92,94,42,64,57,69,60,34,84,70,8,24,51,36,11,61,41,32,89,83,
  38,3,22,26,27,98,50,72,18,53,71,20,56,93,21,67,45,52,74,75,17,62,91,4,43,
  16,63 **makespan:** 6972

  Improvement1: 9, 38, 5, 31, 53, 81, 87, 47, 26, 1, 96, 56, 69, 70, 63, 74, 54, 59, 68, 98, 36, 7, 11, 30, 24, 58, 43, 46, 78, 76, 95, 80, 3, 18, 48, 79, 88, 27, 19, 66, 10, 77, 92, 52, 6, 17, 8, 61, 15, 44, 50, 57, 29, 94, 4, 89, 93, 25, 2, 32, 39, 37, 97, 82, 34, 23, 71, 0, 41, 84, 22, 86, 75, 42, 62, 73, 67, 51, 64, 28, 83, 14, 99, 60, 16, 35, 55, 45, 33, 90, 91, 20, 85, 12, 72, 21, 65, 40, 13, 49 **makespan:** 6444

  Improvement2: 98, 1, 16, 67, 27, 74, 36, 65, 86, 6, 60, 34, 35, 44, 24, 73, 88, 47, 59, 15, 54, 75, 66, 87, 10, 31, 99, 45, 95, 94, 41, 2, 70, 4, 30, 58, 22, 93, 28, 21, 64, 63, 55, 7, 40, 68, 78, 81, 11, 5, 79, 39, 14, 92, 90, 49, 91, 89, 84, 69, 9, 56, 25, 51, 62, 37, 42, 0, 18, 17, 48, 13, 71, 32, 43, 72, 8, 85, 20, 53, 33, 50, 3, 76, 57, 19, 26, 83, 52, 61, 23, 77, 97, 80, 38, 29, 82, 46, 96, 12 **makespan:** 6420

  Deviation: 11%

- ***100 jobs 20 machines:*** Optimal makespan: 6286

  Stinson1: 0,62,95,61,91,42,68,71,33,66,60,79,50,87,83,99,81,82,25,32,52,13
  ,73,74,14,70,45,17,57,53,8,38,16,75,43,2,31,86,89,40,6,56,10,55,80,30,41,
  54,48,65,88,35,96,69,15,94,21,4,77,46,37,29,18,44,22,26,39,19,47,93,11,3,
  28,72,7,24,97,49,84,23,9,27,1,12,98,59,85,5,36,64,76,51,58,67,90,63,20,92
  ,78,34 **makespan:** 7827

  Stinson2: 0,84,73,79,38,2,59,14,18,16,20,46,7,34,31,47,51,40,13,85,19,4,
  15,22,55,96,81,99,62,35,37,65,5,57,74,89,82,24,48,49,71,83,58,45,3,29,52,
  54,90,66,53,77,88,44,17,42,12,60,30,32,92,1,6,76,98,25,72,10,64,39,68,75,
  36,33,94,21,9,23,67,87,50,56,70,95,11,63,27,97,43,93,69,28,86,61,91,8,41,26,78,80 **makespan:** 7623

  Widmer: 0,37,26,3,88,84,25,28,98,81,12,92,79,6,66,46,31,67,21,53,11,56,8,
  50,54,19,39,91,43,69,17,61,44,24,51,22,75,90,60,96,4,38,59,64,57,86,74,18,
  93,87,65,76,99,35,85,7,73,47,42,78,40,77,58,41,97,49,94,95,36,30,32,80,5,
  2,52,70,9,10,89,15,34,72,82,48,83,14,16,33,62,45,27,68,20,63,29,55,13,23,
  1,71 **makespan:** 7578

  Improvement1: 51, 98, 87, 92, 37, 45, 76, 54, 3, 93, 96, 72, 69, 63, 90, 9, 6, 61, 70, 68, 65, 28, 19, 43, 12, 26, 46, 30, 4, 2, 57, 21, 48, 24, 80, 53, 50, 85, 34, 16, 75, 42, 95, 8, 13, 84, 36, 1, 64, 41, 67, 40, 27, 88, 33, 55, 25, 20, 32, 44, 66, 31, 74, 81, 83, 10, 56, 97, 29, 89, 73, 82, 62, 0, 60, 58, 7, 59, 15, 5, 91, 11, 94, 49, 71, 52, 79, 17, 78, 39, 14, 35, 47, 99, 77, 18, 86, 23, 38, 22 **makespan:** 7350

  Improvement2: 56, 30, 12, 18, 0, 13, 32, 49, 26, 66, 3, 83, 2, 71, 46, 99, 38, 62, 20, 31, 9, 48, 39, 14, 50, 98, 40, 95, 37, 51, 85, 77, 92, 44, 41, 61, 75, 91, 78, 24, 52, 96, 15, 97, 17, 28, 16, 25, 5, 42, 72, 67, 10, 81, 68, 43, 11, 35, 90, 59, 73, 54, 23, 80, 45, 7, 33, 8, 79, 6, 34, 70, 93, 19, 76, 47, 65, 53, 55, 4, 60, 21, 29, 82, 88, 84, 36, 69, 1, 64, 22, 86, 94, 58, 74, 87, 63, 89, 57, 27 **makespan:** 7307

  Deviation: 16%

- ***200 jobs 10 machines:*** Optimal makespan: 10868

  Stinson1: 0,36,63,141,40,171,149,53,150,55,143,10,28,58,110,39,96,5,159,3,1,109,
  75,11,127,23,172,197,199,104,30,187,2,98,79,115,113,118,12,68,148,154,47,
  184,4,182,151,166,35,139,91,84,189,95,133,114,49,44,144,176,169,34,60,100,51
  ,90,163,168,15,178,19,152,198,179,135,87,121,72,14,123,106,61,107,173,158

,122,116,146,186,196,132,89,136,26,153,117,18,99,188,13,130,112,192,37,67
,41,105,170,175,17,162,33,128,64,167,20,165,25,73,62,190,161,81,50,71,16,
29,56,21,32,42,193,78,48,138,134,155,147,194,46,31,74,94,101,24,181,191,
77,197,120,88,80,69,65,119,76,83,164,66,45,8,70,145,140,103,157,108,9,59,
38,177,183,57,111,142,129,92,54,185,195,22,43,131,93,86,82,180,125,6,137,
7,160,124,126,52,102,27,85,156,174 **makespan:** 11933

Stinson2: 0,93,154,188,127,107,26,25,34,134,139,23,150,17,24,116,114,
163,103,3,112,140,125,19,152,39,110,71,196,136,6,32,137,173,9,53,7,83,4,165
,195,180,15,45,42,41,149,75,89,157,48,141,76,167,84,104,79,29,151,120,8,
146,124,81,121,40,74,12,10,55,18,67,14,187,147,132,28,33,175,153,156,186,143
,49,62,174,178,1,129,122,60,59,54,158,97,82,47,193,161,199,168,126,56,190
,182,64,66,99,30,11,170,113,198,179,177,94,117,164,90,108,185,138,51,160
,145,181,128,166,36,16,159,46,44,2,98,35,22,176,31,183,155,194,131,69,130
,184,96,68,171,133,169,101,21,192,37,13,115,106,105,100,85,191,73,52,91,63
,72,86,109,197,43,57,148,162,172,70,27,92,38,111,142,144,50,88,65,80,118,
61,78,102,5,119,189,95,20,58,77,87,135,123 **makespan:** 11956

Widmer: 0,36,63,141,40,171,149,53,150,55,143,10,28,58,110,39,96,5,159,3,
1,109,75,11,127,23,172,197,199,104,30,187,2,98,79,115,113,118,12,68,148,
154,47,184,4,182,151,166,35,139,91,84,189,95,133,114,49,44,144,176,169,34,
60,100,51,90,163,168,15,178,19,152,198,179,135,87,121,72,14,123,106,61,107
,173,158,122,116,146,186,196,132,89,136,26,153,117,18,99,188,13,130,112,
192,37,67,41,105,170,175,17,162,33,128,64,167,20,165,25,73,62,190,161,81,
50,71,16,29,56,21,32,42,193,78,48,138,134,155,147,194,46,31,74,94,101,24,
181,191,77,97,120,88,80,69,65,119,76,83,164,66,45,8,70,145,140,103,157,
108,9,59,38,177,183,57,111,142,129,92,54,185,195,22,43,131,93,86,82,180,125
,6,137,7,160,124,126,52,102,27,85,156,174 **makespan:** 11978

Improvement1: 139, 71, 100, 105, 158, 112, 89, 184, 103, 29, 42, 179, 185, 99, 169, 123, 142, 131, 38, 8, 159, 96, 28, 197, 176, 190, 156, 175, 93, 18, 69, 174, 48, 56, 122, 157, 59, 51, 41, 194, 78, 30, 104, 189, 66, 9, 47, 178, 3, 81, 192, 154, 61, 187, 94, 147, 17, 177, 115, 20, 53, 39, 162, 57, 12, 141, 180, 46, 173, 119, 153, 22, 5, 198, 6, 199, 37, 171, 25, 127, 172, 91, 121, 11, 102, 114, 150, 77, 40, 136, 14, 106, 191, 182, 23, 149, 82, 92, 167, 197, 76, 50, 101, 58, 95, 86, 168, 10, 111, 126, 16, 170, 151, 128, 73, 2, 132, 125, 4, 36, 67, 72, 135, 195, 124, 85, 140,

196, 43, 15, 26, 90, 64, 79, 181, 60, 80, 74, 34, 133, 160, 130, 164, 108, 145, 88, 110, 32, 1, 75, 70, 63, 7, 116, 68, 186, 33, 163, 120, 134, 183, 31, 24, 113, 54, 84, 155, 65, 165, 49, 129, 35, 83, 27, 109, 188, 107, 146, 62, 138, 98, 143, 52, 45, 118, 161, 117, 21, 166, 137, 148, 13, 55, 87, 193, 0, 19, 44, 152, 144 **makespan:** 11605

Improvement2: 61, 121, 21, 119, 143, 139, 171, 120, 5, 113, 23, 157, 180, 25, 158, 118, 35, 18, 52, 29, 92, 70, 164, 98, 46, 115, 106, 124, 194, 66, 65, 166, 3, 7, 81, 159, 181, 184, 142, 151, 99, 32, 149, 48, 64, 56, 102, 132, 0, 104, 60, 150, 101, 36, 196, 8, 71, 43, 160, 153, 54, 199, 197, 74, 22, 162, 133, 161, 131, 127, 170, 37, 182, 31, 109, 197, 177, 55, 188, 147, 191, 192, 49, 111, 11, 40, 173, 62, 85, 90, 78, 77, 51, 145, 165, 75, 24, 30, 176, 73, 148, 84, 94, 189, 122, 42, 168, 186, 17, 190, 91, 167, 183, 79, 34, 134, 136, 114, 19, 88, 178, 110, 140, 146, 28, 107, 76, 1, 59, 112, 12, 69, 89, 135, 45, 128, 4, 10, 185, 33, 80, 14, 137, 96, 187, 154, 47, 39, 38, 50, 87, 156, 86, 117, 27, 95, 6, 16, 195, 93, 116, 2, 126, 67, 82, 44, 141, 26, 20, 9, 125, 169, 152, 15, 58, 163, 103, 193, 130, 41, 129, 53, 123, 198, 13, 63, 155, 83, 179, 175, 100, 72, 174, 105, 138, 144, 108, 172, 57, 68 **makespan:** 11724

Deviation: 6.7%

Id	m	n	upperbound	lowerbound	our solution	itertion
1	100	20	6286	5851	7545	10000
2	100	20	6241	6099	7544	10000
3	100	20	6329	6099	7759	10000
4	100	20	6306	6072	7568	10000
5	100	20	6377	6009	7461	10000
6	100	20	6437	6144	7612	10000
7	100	20	6346	5991	7491	10000
8	100	20	6481	6084	7723	10000
9	100	20	6358	5979	7679	10000
10	100	20	6465	6298	7969	10000

Table 4.4: Results for 20 machine and 100 jobs table

Id	m	n	upperbound	lowerbound	our solution	itertion
1	200	20	11294	10979	13191	3000
2	200	20	11420	10947	13779	3000
3	200	20	11446	11150	13400	3000
4	200	20	11347	11127	13370	3000
5	200	20	11311	11132	13513	3000
6	200	20	11282	11085	13694	3000
7	200	20	11456	11194	13318	3000
8	200	20	11415	11126	13580	3000
9	200	20	11343	10965	13881	3000
10	200	20	11422	11122	13589	3000

Table 4.5: Results for 20 machines and 200 jobs table

Chapter 5

Conclusion and Future work

In this thesis, we have proposed a combinatorial approach to evaluate PFSP by getting inspired by the foraging behavior of ant. To the extent of our belief, nobody has ever approached the procedure we followed here to solve PFSP. We have done a detailed analysis of parameter which is important to have a better starting point. We introduced a new technique for the PFSP solution improvement. It gave a satisfactory performance for larger input data with a very low deviation. For smaller input set the solution is almost perfect.

5.1 Future Work

There can be many other directions to solve PFSP which we will try in future.

- Parameter tuning for ACO algorithm might improve the result.
- Applying ACO technique to PFSP without any reduction technique can be another approach to explore.
- Applying Other methods, e.g., Simulated Annealing, Tabu Search, Genetic Algorithm, Path Algorithms, Particle Swarm Optimization for improvement we will explore in future as well.

Bibliography

[AB99] Igor Averbakh and Oded Berman. A simple heuristic for m-machine flow-shop and its applications in routing-scheduling problems. *Oper. Res.*, 47(1):165–170, January 1999.

[AS] Sumer C. Aggarwal and Edward Stafford. A heuristic algorithm for the flowshop problem with a common job sequence on all machines*. *Decision Sciences*, 6(2):237–251.

[BR03] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, 2003.

[Bru04] Peter Brucker. *Scheduling algorithms (4. ed.)*. Springer, 2004.

[cam]

[CD11] Gianni Di Caro and Marco Dorigo. Antnet: Distributed stigmergetic control for communications networks. *CoRR*, abs/1105.5449, 2011.

[con96] Scheduling: Theory, algorithms, and systems. *IIE Transactions*, 28(8):695–697, 1996.

[Cro58] A. Croes. A method for solving traveling salesman problems. *Operations Research*, 5:791–812, 1958.

[CVA95] Chuen-Lung Chen, Venkateswara S. Vempati, and Nasser Aljaber. An application of genetic algorithms for flow shop problems. *European Journal of Operational Research*, 80(2):389 – 396, 1995.

[DAGP90] J. L. Deneubourg, S. Aron, S. Goss, and J. M. Pasteels. The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behavior*, 3(2):159–168, Mar 1990.

[Dan77a] David G. Dannenbring. An evaluation of flow shop sequencing heuristics. *Management Science*, 23(11):1174–1182, 1977.

[Dan77b] David G. Dannenbring. An evaluation of flow shop sequencing heuristics. *Management Science*, 23(11):1174–1182, 1977.

[DBB+04] Marco Dorigo, Mauro Birattari, Christian Blum, Luca Maria Gambardella, Francesco Mondada, and Thomas Stützle, editors. *Ant Colony Optimization and Swarm Intelligence, 4th International Workshop, ANTS 2004, Brussels, Belgium, September 5 - 8, 2004, Proceedings*, volume 3172 of *Lecture Notes in Computer Science*. Springer, 2004.

[DBS06] Marco Dorigo, Mauro Birattari, and Thomas Sttzle. Ant colony optimization. 1:28–39, 12 2006.

[DC99] Marco Dorigo and Gianni Di Caro. Ant colony optimization: A new meta-heuristic. In *PROCEEDINGS OF THE CONGRESS ON EVOLUTIONARY COMPUTATION*, pages 1470–1477. IEEE Press, 1999.

[DG97] Marco Dorigo and Luca Maria Gambardella. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, April 1997.

[DMC96] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. The ant system : Optimization by a colony of cooperating agents. 1996.

[DS04] Marco Dorigo and Thomas Stützle. *Ant colony optimization*. MIT Press, 2004.

[FGL04] J M Framinan, J N D Gupta, and R. Leisten. A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. *Journal of the Operational Research Society*, 55(12):1243–1255, Dec 2004.

[Fla98] Gary William Flake. *The Computational Beauty of Nature*. MIT Press, Cambridge, MA, USA, 1998.

[GC05] Dorian Gaertner and Keith Clark. On optimal parameters for ant colony optimization algorithms. In *Proceedings of the International Conference on Artificial Intelligence 2005*, pages 83–89. CSREA Press, 2005.

[GJ79] M. R. Garey and David S. Johnson. Computers and intractability: A guide to the theory of np-completeness. 1979.

[Gup79] Jatinder N. D. Gupta. *A review of flowshop scheduling research*, pages 363–388. Springer Netherlands, Dordrecht, 1979.

[HC91] Johnny C. Ho and Yih-Long Chang. A new heuristic for the n-job, m-machine flow-shop problem. *European Journal of Operational Research*, 52(2):194 – 202, 1991.

[HS04] Holger Hoos and Thomas Sttzle. *Stochastic Local Search: Foundations & Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[Joh] S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68.

[JP] R. Johnson and M. G. Pilcher. The traveling salesman problem, edited by e.l. lawler, j.k. lenstra, a.h.g. rinnooy kan, and d.b shmoys, john wiley & sons, chichester, 1985, 463 pp. *Networks*, 18(3):253–254.

[KS80] J. R. KING and A. S. SPACHIS. Heuristics for flow-shop scheduling. *International Journal of Production Research*, 18(3):345–357, 1980.

[Lan95] Christopher G. Langton. *Artificial Life: An Overview*. MIT Press, Cambridge, MA, USA, 1995.

[Lou96] Helena Ramalhinho Loureno. Sevast'yanov's algorithm for the flow-shop scheduling problem. *European Journal of Operational Research*, 91(1):176 – 189, 1996.

[Moc95] Joo Vitor Moccellin. A new heuristic method for the permutation flow shop scheduling problem. *The Journal of the Operational Research Society*, 46(7):883–886, 1995.

[NEH83] Muhammad Nawaz, E Emory Enscore, and Inyong Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91 – 95, 1983.

[NS96] Eugeniusz Nowicki and Czesaw Smutnicki. A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research*, 91(1):160 – 175, 1996.

[OP89] IH Osman and CN Potts. Simulated annealing for permutation flow-shop scheduling. *Omega*, 17(6):551 – 557, 1989.

[OS90] F.A. Ogbu and D.K. Smith. The application of the simulated annealing algorithm to the solution of the n/m/cmax flowshop problem. *Computers Operations Research*, 17(3):243 – 253, 1990.

[Pag61] E. S. Page. An approach to the scheduling of jobs on machines. *Journal of the Royal Statistical Society. Series B (Methodological)*, 23(2):484–492, 1961.

[PPE84] YANG BYUNG PARK, C. DENNIS PEGDEN, and E. EMORY ENSCORE. A survey and evaluation of static flowshop scheduling heuristics. *International Journal of Production Research*, 22(1):127–141, 1984.

[PS82] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, Englewood Cliffs, NJ, 1982.

[Rai06] Günther R. Raidl. A unified view on hybrid metaheuristics. In *Proceedings of the Third International Conference on Hybrid Metaheuristics*, HM'06, pages 1–12, Berlin, Heidelberg, 2006. Springer-Verlag.

[Ree93a] Colin R. Reeves. Improving the efficiency of tabu search for machine sequencing problems. *Journal of the Operational Research Society*, 44(4):375–382, 1993.

[Ree93b] Colin R. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*. John Wiley & Sons, Inc., New York, NY, USA, 1993.

[Ree95] Colin R. Reeves. A genetic algorithm for flowshop sequencing. *Computers Operations Research*, 22(1):5 – 13, 1995. Genetic Algorithms.

[Sev95] Sergey Sevast'Janov. Vector summation in banach space and polynomial algorithms for flow shops and open shops. *Mathematics of Operations Research*, 20(1):90–103, 1995.

[SH96] Thomas Sttzle and Holger Hoos. Improvements on ant-system: Introducing max-min ant system, 1996.

[SL93] S Sarin and M Lefoka. Scheduling heuristic for the n-jobm-machine flow shop. *Omega*, 21(2):229 – 234, 1993.

[Soc04] Krzysztof Socha. ACO for continuous and mixed-variable optimization. In *Ant Colony Optimization and Swarm Intelligence, 4th International Workshop, ANTS 2004, Brussels, Belgium, September 5 - 8, 2004, Proceedings*, pages 25–36, 2004.

[SS82] JOEL P. STINSON and ARTHUR W. SMITH. A heuristic prorammin procedure for sequencin the static flowshop. *International Journal of Production Research*, 20(6):753–764, 1982.

[SS13] Km. Shweta and Alka Singh. An effect and analysis of parameter on ant colony optimization for solving travelling salesman problem. 2013.

[Ste94] Ian Stewart. The ultimate in anty-particles. *Scientific American - SCI AMER*, 271:104–107, 07 1994.

[St97] Thomas Sttzle. An ant approach to the flow shop problem. In *In Proceedings of the 6th European Congress on Intelligent Techniques Soft Computing (EUFIT'98*, pages 1560–1564. Verlag, 1997.

[Tai90] E. Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1):65 – 74, 1990.

[Tai93] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, January 1993.

[TB87] Scott Turner and Dean Booth. Comparison of heuristics for flow shop sequencing. *Omega*, 15(1):75 – 78, 1987.

[UM93] Ron Unger and John Moult. Finding the lowest free energy conformation of a protein is an np-hard problem: Proof and implications. *Bulletin of Mathematical Biology*, 55(6):1183–1198, Nov 1993.

[WH89] Marino Widmer and Alain Hertz. A new heuristic method for the flow shop sequencing problem. 41:186–193, 02 1989.

[wik] Flow shop scheduling: https://en.wikipedia.org/wiki/flow_shop_scheduling.

[WN99] Laurence A. Wolsey and George L. Nemhauser. *Integer and Combinatorial Optimization*. Wiley-Interscience, New York, NY, 1 edition, nov 1999.

[WY98] Hoon-Shik Woo and Dong-Soon Yim. A heuristic algorithm for mean flowtime objective in flowshop scheduling. *Computers Operations Research*, 25(3):175 – 182, 1998.

Curriculum Vitae

Graduate College
University of Nevada, Las Vegas

Tasmin Chowdhury
tasmin.buet@gmail.com

Degrees:

Bachelor of Science in Computer Science and Engineering 2016
Bangladesh University of Engineering and Technology

Thesis Title: Combinatorial Ant Optimization and The Flowshop Problem

Thesis Examination Committee:

Chairperson, Dr. Wolfgang Bein, Ph.D.
Committee Member, Dr. Ajay Datta, Ph.D.
Committee Member, Dr. Justin Zhan, Ph.D.
Graduate Faculty Representative, Dr. Pradip Bhowmik, Ph.D.