Chapter 1

Introduction

Chapter 1
Introduction

Optimization is an activity, in which we attempt to perform our best under the given conditions and constraints. This activity is a reality in many different industrial settings. In practice, optimization is a broad area in science and engineering with many interesting subareas. In the modern day, the term 'optimization' is widely being used when there is almost a clear mathematical representation of a problem to be optimized. The classical definition of optimization is the process of finding the highest or lowest ranked solution to a problem as measured by one or more *objective functions*. Most of the times, we use the term optimization in the sense that absolute optimum is not required. We can classify optimization into two major types; single objective optimization and multi-objective optimization.

One of the most active current research areas in global optimization is deriving the problem solving techniques specially for unstructured NP-hard problems arising from applications. Despite the inherent difficulty of most of these problems, significant progress has been achieved in last twenty years in the development of special solution strategies adapted to various mathematical and algorithmic structures. The design as well as analysis of such problem specific algorithms has been extensively studied for a wide range of problems. The goal in this field of research is to obtain an algorithm, that is provably optimal with respect to their run time and convergence of the solutions.

While looking at the various results obtained in this field, one can observe the following. Most algorithms, be it in optimization or any other fields, depend heavily on parameters. These parameters may be required to remain between two possibly finite bounds or may be constrained in a different way. The overall behavior of the algorithm is influenced by the numbers and the values of these parameters. Unfortunately, for most practical cases it remains unclear, how a user should proceed to determine the values for these parameters. Over-parameterization may play a negative role during the computation. The framework design of algorithm relies on the observation that, measures of performance can be derived from the

dependencies of the algorithm on its parameters. These measures are context and method dependent. For any formulated optimization problem, measure of performance needs to be minimized as a function of parameters over a domain of acceptable values.

This thesis will concentrate on one of the popular kind of bio-inspired algorithms, Ant Colony Optimization (ACO). The need for unified introduction to optimization problems is quite essential at this stage. The progress in the understanding of optimization problems has shown large amounts of diversity in their runtime and approximability. In addition, natural optimization problems do seem to exhibit noticeable special trends in their behavior. The literature has witnessed many attempts to describe this behavioral trends but, it is hard to describe and extract these trends. To understand the formal basis of such approximation schemes, we begin with the brief description of combinatorial optimization.

1.1 Combinatorial Optimization

A Combinatorial Optimization (CO) problem is an optimization problem, where the set of *feasible* (satisfying a set of constraints) solutions is obtained from *candidate solutions* (Papadimitriou and Steiglitz, 1982). They are abundant in quite a few areas, where resource constrained problems appear either naturally or when a discretization of a continuous problem may be useful. Examples of well known CO problems include the Vehicle Routing Problem (Clarke and Wright, 1964), Traveling Salesman Problem (TSP) (Applegate et al., 2007), Knapsack problem (Kelleres et al., 2004), Cutting Stock Problems (De Carvalho, 2002), Generalized Assignment Problem (Ross and Soland, 1975) and Train Scheduling Problem (Assad, 1980), etc. The CO problems are conceptually easy to model, but quite hard to implement and suffers from the curse of dimension i.e, selecting the best among the possible number of solutions grows exponentially with the increase in size of the problem instances.

The Combinatorial Optimization has reached the level of generalization based on primary knowledge accumulated in the domain of this field. The researchers

started devoting more attention towards the refinement of existing techniques and analyze the similarities, distinctions and conceptual characteristics inorder to increase the performance efficiency. The similarities can be studied through formulation of generalized search procedures, specifying components of which, one will remain almost same for variety of algorithms. Usually, distinctions shall be captured through a proper classification. The CO algorithms available in the literature fall broadly into two main classes:

- **Exact Algorithms** - These methods enumerate all the possible *candidate solutions* and return the best solution in the feasible solution set. However, the computation time for these methods increases exponentially with respect to the problem size, and often only small or moderate size problem instances can be practically solved like Greedy method (Dijkstra, 1959) and Dynamic programming (Bellman, 1958).

- **Approximate algorithms** - These methods provide relatively good solutions in short computational time without enumerating all the possibilities, but the obtained solutions may not be the optimal one. The basic approximate methods can be broadly classified as *constructive algorithms* (Dorigo, 1996) or *local search algorithms* (Croes, 1958). Constructive algorithms generate the solution from the scratch by adding-to an initially empty partial solution-components, until a solution is complete. They are typically the faster approximate methods, yet they often yield solution of lower quality unlike local search algorithms. Local search algorithms begin from some initial solution and iteratively try to replace the current solution by a better solution in an appropriately defined neighborhood of the current solution. Thirdly, a new class of approximate algorithms has emerged in the eighties of last century, called *metaheuristics*, which will be discussed in the next section.

1.2 Metaheuristics

1.2.1 Introduction

Heuristics represent a class of methods for which, in general there is no formal or classical proof of their performances. Most of the approximation methods are based on some heuristic rules. One can also note that, most of the popular CO problems are NP-hard (Korte and Vygen, 2006) and the best exact algorithms known so far on these problems have exponential time complexity. Therefore approximate algorithms and in particularly heuristics may be the preferable way to hunt the solutions of 'better' quality in a reasonable duration.

As mentioned earlier, a new kind of approximate algorithm has emerged, which basically tries to combine heuristic methods in higher level frameworks aimed at efficiently as well as effectively to explore the search space. These methods are called metaheuristics. This term was first introduced by Glover (Glover, 1986). Etymologically, metaheuristics is derived from two Greek words. The word heuristic means "to search" and meta stands for "beyond a higher level". A simple and formal definition of metaheuristics is as follows (Osmen and Laporte, 1996): " It is an iterative generation process which guides a subordinate heuristics by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find a efficiently near optimal solution".

In recent years, there has been splurge in new approaches belonging to metaheuristics class. These approaches can be seen as a general algorithmic framework that can be applied to different optimization problems with relatively few modifications to mold them and adapt to a specific problem. Metaheuristics are typically high-level strategies which guide an underlying, more problem specific heuristics to increase their performances. The main goal here is to avoid the disadvantages of iterative improvement and, in particular, multiple descent by allowing the local search to escape from local optima. This is achieved by either allowing worsening moves or generating new starting solutions for the local search in a more intelli-

gent way than just providing random initial solutions. Many of the methods can be interpreted as introducing a bias (Blum and Roli, 2003) such that high quality solutions are produced quickly. This bias can be of various forms and can be descent bias (based on the objective functions), memory bias (based on previously made decisions) or experience bias (based on prior performance). Many of the metaheuristic approaches rely on probabilistic decisions made during the search. The main difference to pure random search is that, in metaheuristics algorithm randomness is not used blindly but in an intelligent, biased form.

We summarize the basic characteristics of metaheuristics as (Blum and Roli, 2003) follows:

- Metaheuristics are strategies that guide the search process. Their goal is to efficiently explore the search space in order to find (near-)optimal solutions.
- Metaheuristics may incorporate mechanisms to avoid getting trapped in confined areas of the search space.
- The basic concepts of metaheuristics can be described on an abstract level i.e., not tied to a specific problem.
- Metaheuristics often use the experience gained in previous searches to guide new searches.
- Metaheuristics may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy. Those strategies must be chosen in such a way to balance dynamically the exploitation of previously gained experience, called intensification and the exploration of the search space, called diversification. This balance is necessary to quickly identify the region in the search space where good solutions are present and not to waste time in searching the regions that have already been explored or that do not have good solutions.

1.2.2 Classification of Metaheuristics

There are different ways to classify the metaheuristic algorithms. Basically, all numerical and computational approaches for solving CO problems can be characterized as search algorithms (Hoos and Stutlze, 2005). For marking out the characteristic features of CO methods, the classification is strongly suggested. Stutlze (Stutlze, 1998) classified the metaheuristics by whether they are trajectory or discontinuous, by the number of operated solutions, by the memory usage, by the number of neighborhood structures, by the changes to the objective functions and by the sources of inspiration. Vaessens (Vaessens et al., 1998) classified more formally based on abstract algorithmic skeleton namely, the number of solutions operated at a time and the number of neighborhoods. Another classification of metaheuristics can be seen in the paper due to Talbi (Talbi, 2009). We briefly summarize all these classifications below.

- **Nature inspired and Non-nature inspired** - One way to classify metaheuristics is based on the inspiration (nature or non nature) for the origin of algorithms. Some of the examples for nature inspired algorithms are Genetic Algorithm(GA) (Holland, 1962), Artificial Bee Colony(ABC) (Basturk and Karaboga, 2006) etc and non nature inspired algorithms are Tabu Search(TS) (Glover, 1989), Iterated Local Search(ILS) (Den Besten et al., 2001). The resultant classification will lose meaning, if applied to recent hybrid algorithms as they may belong to both the categories.

- **Population based search and Single point search** - Another characteristic that can be used for classification is the number of solution components used at the same time. The algorithms like GA (Holland, 1962) works with population of points at a time and generates the set of possible solutions whereas algorithm like ILS (Den Besten et al., 2001) works with single point at a time and try to improvise the current best solution in successive interval of time.

- **Dynamic and Static objective functions** - Metaheuristics can also be classified according to the way they make use of the objective functions. While some algorithms like GA (Holland, 1962) do not change the objective function, but some others like Guided Local Search (GLS) (Voudouris, 1998) modify the search landscape in order to escape from the local minima. This modification results change in the objective function, incorporated with information collected during the search process.

- **One and Various neighborhood structures** - Most metaheuristic algorithms work on single neighborhood structure (Glover, 1989). The search landscape does not change in the course of the algorithm. Some metaheuristics like Variable Neighborhood Search (VNS) (Mladenovic and Hansen, 1997) defines a multiple set of neighborhood structures that helps in diversifying the search by swapping different landscapes.

- **Memory usage and Memoryless methods** - The search history is another important feature that can be used for classification purpose. Memoryless algorithms perform a Markov process, and their next move depends on the current state of the search process (e.g. GRASP (Feo and Resende, 1995)). The memory based algorithms usually keep track of recent moves or solutions in order to take decision regarding future moves (e.g. Tabu Search (Glover, 1989)).

- **Iterative and Greedy** - Another way to characterize the metaheuristic algorithms is based on solution generation process. The iterative algorithms starts with a complete solution and transform it at each iteration using some search operators. Greedy algorithms start from an empty solution, and at each step, a decision variable of the problem is assigned until a complete solution is obtained.

- **Deterministic and Stochastic** - A deterministic metaheuristics solve an optimization problem by making deterministic decisions (e.g.tabu search

(Glover, 1989)). In stochastic metaheuristics, some random rules are applied during search (e.g., Simulated Annealing (Kirkpatrick et al.,(1983)).

Metaheuristics can also be classified by their special behaviors, such as structure, search process and performance characteristics. But, one important point is, though the classification by the complexity of the structure is important, it is hardly possible to distinguish algorithms strictly by the characteristics because there is no precise, formal as well as universally accepted definition for the term 'metaheuristics'.

1.3 Metaheuristic Algorithms

In this section, we will discuss some of the important and most commonly used metaheuristic algorithms for solving the CO problems.

1.3.1 Evolutionary Computation

Evolutionary Computation (EC) is the general term for the collection of nature inspired, population based algorithms. EC mimics the nature ability to retain the best species in the evolution process. This corresponds to *survival of fittest*, that can adjust to the ever changing environment. The evolutionary processes have led to the development of several computational models to tackle CO problems in an efficient manner. The basic structure of an EC algorithm is given by Algorithm 1.

Algorithm 1 Evolutionary Computation

P = GenerateInitialPopulation()
while termination condition not met **do**
  P' = Recombine(P)
  P'' = Mutate(P')
  Evaluate(P'')
  P = Select($P'' \cup P$)
end while

Each iteration of the algorithm corresponds to a *generation*, where certain operators are applied to some / all individuals of the current population to generate

the individuals of the population of the next generation. The following are the list of operators that are most commonly applied:

1. **Recombination or Crossover** - to recombine two or more individuals to produce new individuals.

2. **Mutation** - to modify single individuals to obtain self-adaptation.

3. **Selection** - to select the individuals based on their fitness. Individuals with higher fitness have a higher probability to be selected for next generation.

In literature, there are mainly three different categories of EC algorithms: Evolutionary Programming (EP) (Fogel, 1962), Evolutionary Strategies (ES) (Rechenberg, 1965) and Genetic Algorithms (Holland, 1962). The algorithms that fall in the EP and ES category mostly apply to continuous optimization problems, while GA are more specific for discrete and combinatorial optimization problems.

1.3.2 Tabu Search

Tabu Search (TS) is the most commonly used metaheuristics for CO problems. The basic idea was first introduced by Glover (Glover, 1989). TS is essentially an improved version of local search and it is also known as Hill Climbing technique. It generates the initial set of solutions and then selects the best one as the current solution. The new set of solutions is generated by looking at the neighbors of current solution and generation process is controlled by the neighborhood structure. If neighbor set contains the better solution, then current solution will be replaced with newly found best solution. TS maintains a *tabu-list*, also called as *short term memory* to keep track of *best-so-far* found solutions. The list will be updated in first-in first-out manner, when best solution evolves. The process of solution generation and tabu-list updation is repeated until an improved solution is found in the neighborhood of a current solution. This effectively leads to finding the local optimum solution which might be much worse than the global optimum. In practice, algorithm stops improving when local optimum is found. The strength of the TS metaheuristic lies in employing three important strategies to escape from

the local optimum. The TS-specific strategies are: *Best improvement, Tabu list* and *Aspiration criterion.*

Best improvement means that each current solution is always replaced by its best neighbor, even if the best neighbor is worse than the current solution. This ensures that, search won't get stuck in local optima. The problem with best improvement is possible cycling among already visited solutions due to situation like, the best neighbor of a solution is indeed the last visited current solution. The cycling can be avoided by keeping track of the most recently visited solutions in *tabu list* and forbids moves toward them. The tabu list stores some attributes of these solutions instead of whole solutions, which require large memory and efficient management. The choice of attributes is yet another issue that need to be resolved. Typically, tabu lists store the *moves* that need to be performed in order to go from one solution to another or the differences between solutions. In this way, the memory requirement of tabu lists becomes feasible, but another problem arises: forbidding all solutions corresponding to a tabu attribute may also forbid solutions that have not yet been visited, and possibly also very good or optimal solutions. TS employs *aspiration criteria* for solving this problem. An aspiration criterion is a condition, if satisfied, allows to replace current solution obtained by performing a tabu move. A typical example of aspiration criterion requires that a solution is better than the best solution found from the beginning of the algorithm. The skeleton of Tabu Search is given by Algorithm 2.

Algorithm 2 Tabu Search

Generate a starting current solution x.
Initialize the tabu list.
for $k = 1, 2, \ldots n$ **do**
  Set $A(x,k) = \{y \in S(x) \backslash T(x,k) \cup \tilde{T}(x,k)\}$
  Set $x = \arg\min_{y \in A(x,k)} g(y)$
  Update the tabu lists and the aspiration criteria.
end for

Here x, y are feasible solutions of the combinatorial optimization problem, $A(x,k)$ is the set of solutions among which the new current solution is chosen at iteration k, $S(x)$ is the set of neighbors of x, $T(x,k)$ is the set of tabu moves at

iteration k, and $\tilde{T}(x,k)$ is the set of tabu moves satisfying at least one aspiration criterion. In TS, typical stopping criteria may be a maximum number of consecutive iterations not producing an improved solution or the emptiness of the set $A(x,k)$.

1.3.3 Simulated Annealing

Simulated Annealing (SA) is the first metaheuristic algorithm that had an explicit strategy to escape from local minima. It derives inspiration from Metropolis algorithm (Metropolis et al., 1953) and it was first proposed by Kirkpatrick (Kirkpatrick et al., 1983). The algorithm starts by initializing the temperature parameter T and generates an initial solution either randomly or heuristically. In each iteration, a solution s is generated and if it is better than the current solution, then it will be accepted as new current solution, otherwise its acceptance depends on probability function computed using Boltzmann distribution. The pseudo-code for SA algorithm is given by Algorithm 3:

Algorithm 3 Simulated Annealing

Initialize state x and temperature parameter T_1
for $k = 1, 2, \ldots .n$ **do**
  if $g(y) \leq g(x)$ **then**
    set $x = y$
  else
    if $\exp(\frac{g(x)-g(y)}{T_k}) \leq$ uniform[0,1]
    set $x = y$
    endif
  end if
  update T_k to T_{k+1}
end for

where, x and y are feasible solutions from S, $g(x)$ and $g(y)$ are objective functions of x and y, T_1, T_2 , . . . is a sequence of values for the temperature parameter and the update of values T_k is done according to cooling schedule, the sets $S(x)$ form the pre-defined neighborhood structure: to each feasible solution $x \in S$, a set $S(x) \subseteq S \backslash \{x\}$ of neighbor solutions is assigned, uniform $[\alpha, \beta]$ is a procedure selecting a uniformly distributed random number from the interval $[\alpha, \beta]$.

Normally the temperature T is decreased during the search process. At the beginning of the search, the probability of accepting uphill moves is high and it gradually decreases converging to a simple iterative improvement algorithm. This process is analogous to the annealing process of metals and glass, which assume a low energy configuration when cooled with an appropriate cooling schedule. The cooling schedule controls the diversification and intensification factors, hence appropriate choice is crucial for the performance of the algorithm.

1.3.4 Ant Colony Optimization

All the three metaheuristics described above are often applied to problems whose structure is not known or lacks in information like time, cost, efficiency etc or knowledge to obtain good specific algorithms. It has been widely acknowledged that a strong theoretical foundation for such metaheuristics is necessary. In light of this, another new kind of randomized search metaheuristics has emerged 21 years ago namely, Ant Colony Optimization (ACO). Like evolutionary algorithms, this also imitate the optimization process from nature, in this case the search for a common source of food by a group of ants. Solving CO problems by ACO techniques has become widely popular in recent years.

The conceiving of ACO is originally motivated by the attempt to solve the well known TSP, the researcher recognized that their technique is applicable to a much larger range of problems. In an explicit form, this insight was first established by Dorigo (Dorigo and Di Caro, 1999). In subsequent years, more than hundred quality papers have been published on the successful applications of ACO in different areas of diversity. This thesis deals with study along with original contributions to the development of ACO. The objective of the thesis is to propose new techniques in ACO that are useful in practice, which could yield a clear and well defined gain in academia. Thus, a detailed, easily understandable and properly sequenced introduction to ACO is essential at this juncture.

1.4 ACO: Algorithms, Theory and Applications

We present a detailed overview of the main concepts of ACO in three important perspectives.

1.4.1 ACO: Algorithmic Perspective

Ant Colony Optimization (Dorigo and Stutzle, 2004) is a nature inspired, population based algorithm, fascinated by the foraging behavior of the ants. The tiny creature ants, which are almost blind, can establish the shortest path from nest to the food source. The food hunting activity provides the formal framework to solve the combinatorial optimization problems. While walking from food sources to the nest and vice versa, ants deposit a substance called *pheromone trial* on the ground. When they decide about which direction to proceed, they choose with higher probability, the paths that have strong pheromone concentration. The pheromone trial has two important roles to play in ACO framework. Firstly, it acts as an indirect medium of communication through which ants can share their journey experience. The journey experience is expressed by the amount of pheromone trails present on the traveled path and this helps the ants in making decision. The paths marked by stronger pheromone concentrations have the highest probability to be chosen for the journey. Secondly, being a volatile substance, it evaporates over a period of time and this mechanism helps to forget the bad experiences of the journey. The basic behavior of cooperative interaction and forgetting bad experiences through pheromone leads to the emergence of shortest paths. The first algorithmic framework that captures the essence of ant activity was given by Dorigo et al (Dorigo, 1992). In literature, there are several proposals to improvise and extend the basic ACO algorithm (Stutzle and Hoos, 2000; Bullnheimer et al., 1999). The generic pseudo-code that covers all of them is given in Algorithm 4:

CO problems addressed by ACO are usually encoded by a construction graph $G = (V, A)$, a completely connected graph whose nodes V are components of solutions and arcs A are connections between components. Finding a solution means constructing a feasible walk in G. For example, in the Traveling Salesman Prob-

Algorithm 4 Ant Colony Optimization

```
Initialize the pheromone values.
while termination conditions not met do
  START ScheduleActivities
  ConstructAntsSolutions
  UpdatePheromone
  DeamonActions
  END ScheduleActivities
end while
```

lem (TSP), nodes correspond to cities, arcs correspond to paths connecting cities, and a feasible solution is a Hamiltonian path on the graph.

The ACO algorithm essentially consists of three procedures (Dorigo et al., 1999; Bianchi, 2006) *ConstructAntsSolutions, UpdatePheromones, and Deamon-Actions* gathered in *ScheduleActivities* construct.

ConstructAntsSolutions is the process by which artificial ants construct the solution in an incremental fashion by traversing the construction graph. An ant construct the solution using pheromone trial and heuristic value information associated with each arc. *UpdatePheromones* is the process by which pheromone is modified on arcs. The pheromone trail is decreased on all the arcs as soon as they add a solution component to the partially constructed solution and this operation is called *local update*. The local update reflects the evaporation of chemical substance that happens over a period of time and this action avoids the rapid convergence of the algorithm to suboptimal solutions. Moreover, pheromone is increased on selected good paths to have strong bias in subsequent iterations and this operation is called *global update*. Since ACO is an iterative algorithm, the best selected path for updation can be either *Global best path* or *iterative best path* (Stutzle and Hoos, 2000). The path that is best from the start of the algorithm is called global best path and the path that is best in the current iteration, called iterative best path. The updation can be further classified based on number of ants used: if all the ants get the chance to update the paths then it is called *Communism* approach else, it is called *Elitism* approach, where only the best ants are allowed to update the path. *DeamonActions* are centralized operations, such

as comparing solution values among ants in order to find the best solution or running a local search procedure. *ScheduleActivities* does not specify how these three activities are scheduled, synchronized and it is left to the algorithm designer.

In the literature one can see several variants of ACO algorithms. Here, we present the original Ant System and then some of its successful variants.

Ant System(AS)

The first ant algorithm proposed was Ant System(AS) (Colorni et al., 1991). The algorithm was evaluated using TSP and Quadratic Assignment Problem (QAP) as benchmark problems and it works as follows: during the solution construction phase, ants are randomly placed and they are asked to complete the tour. A transition probability function p_{ij} is defined for the ants to select the next city to be visited. Suppose the ant is in city i and it selects the next city j based on the probabilistic function given by the equation:

$$p_{ij}^{k}(t) = \frac{[\tau_{ij}]^{\alpha}.[\eta_{ij}]^{\beta}}{\sum_{k \notin tabulist}([\tau_{ik}]^{\alpha}.[\eta_{ik}]^{\beta})} \tag{1.1}$$

where τ_{ij} is the pheromone strength on arc ij, η_{ij} is the heuristic information called *visibility* of the arc ij, α and β are parameters, which control the importance of pheromone strength and visibility. The algorithm follows the communism approach and the pheromone updation is given by the expression:

$$\tau_{ij} = \rho \cdot \tau_{ij} + \Delta\tau_{ij} \tag{1.2}$$

where $\Delta\tau_{ij}$ is

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$$

and ρ is the pheromone persistent factor. $\Delta\tau_{ij}^{k}$ is computed according to the equation:

$$\Delta\tau_{ij}^{k} = \begin{cases} Q/L_k & \text{if (i, j)} \in k^{th} \text{ ant's tour list} \\ 0 & \text{otherwise} \end{cases}$$

where Q is the algorithmic constant and L_k is the tour length of k^{th} ant. There are three variants of AS, namely *ant-cycle, ant-density* and *ant-quantity*, where each of them has different pheromone updation strategies. An extension to ant algorithm called *Elitist strategy* (Dorigo et al., 1996) has been proposed, where *best-so-far* tours will be reinforced in addition to standard reinforcement. The additional reinforcement is given by the equation:

$$e \cdot Q/L^* \tag{1.3}$$

where e is the number of elitist ants and Ł* is the length of best tour. The disadvantage of the ant system is that its performance suffers for larger problem size and search stagnation occurs much earlier without proper exploitation.

Ant Colony System (ACS)

Ant Colony System (ACS) (Dorigo and Gambardella, 1996) is a modified version of AS. In solution construction phase, ants select the next city to be visited based on *pseudo-random-propositional rule*. Suppose the ant is in city r and probabilistically would select the city s according to the equation:

$$s = \begin{cases} arg \max\limits_{u \in J_k(r)} \left\{ [\tau(r,u)] \cdot [\eta(r,u)]^{\beta} \right\} & \text{if } q < q_0 \text{ (Exploitation)} \\ S & \text{otherwise (Exploration)} \end{cases} \tag{1.4}$$

where q is a random number uniformly distributed between $[0, 1]$, q_0 is a parameter in the range $(0 < q_0 < 1)$ and S is a random variable selected according to the probability distribution given in equation (1.1), $\tau(r, u)$ and $\eta(r, u)$ are the pheromone and visibility on the (r, u) path and $J_k(r)$ set contains available choices to make a move from city r to the next city. The pheromone updation phase

consists of *global updation* and *local updation*. The global updation is given by the equation:

$$\tau(r,s) \leftarrow (1-\alpha)\cdot \tau(r,s) + \alpha \cdot \Delta\tau(r,s) \tag{1.5}$$

and

$$\Delta\tau(r,s) = \begin{cases} (L_{gb})^{-1} & \text{if (r, s)} \in \text{global best tour} \\ 0 & \text{otherwise} \end{cases}$$

where $0 < \alpha < 1$ is the *pheromone decay* parameter and L_{gb} is the global best tour. Similarly, local updation is given by the equation:

$$\tau(r,s) \leftarrow (1-\rho)\cdot \tau(r,s) + \rho \cdot \Delta\tau(r,s) \tag{1.6}$$

where $0 < \rho < 1$ is a *pheromone reinforcement* parameter and $\Delta\tau(r,s)$ is set to value τ_0.

Rank-based Ant System (RA)

The basic idea of elitist ant strategy has been incorporated in RA (Bullnheimer et al., 1999). In Rank-based ant algorithm, each ant is assigned a rank based on its performance. The solution construction phase is same as AS. The pheromone updation combines elitist and rank strategy to update the selected best tours and is given by the equations.

$$\tau_{ij} = \rho \cdot \tau_{ij} + \Delta\tau_{ij} + \Delta\tau^{*}_{ij} \tag{1.7}$$

where

$$\Delta\tau_{ij} = \sum_{\mu=1}^{\sigma-1} \Delta\tau^{\mu}_{ij}$$

and

$$\Delta\tau^{\mu}_{ij} = \begin{cases} (\sigma-\mu)Q/L_{\mu} & \text{if the } \mu^{th} \text{ best ant travel on edge (i, j)} \\ 0 & \text{otherwise} \end{cases}$$

and

$$\Delta\tau_{ij}^{*} = \begin{cases} \sigma Q/L^{*} & \text{if edge (i, j) is a part of the best solution} \\ 0 & \text{otherwise} \end{cases}$$

where $\Delta\tau_{ij}$ and $\Delta\tau_{ij}^{*}$ specify the amount of pheromone increase due to ranking mechanism and elitist ants. μ specifies the ranking index and σ is the number of elitist ants.

In first updation, trial contribution by an ant on a traveled path will be proportional to its rank and secondary updation will follow the elitist approach. If ant finds a better solution, then its ranking will be better and it will make a better trial contribution.

Max-Min Ant System(MMAS)

Max-Min Ant System is a modified version of elitist ant system (Stutzle and Hoos, 2000). The solution construction phase is same as AS but it differs in pheromone updation phase in two aspects. The first difference is, only the best path is updated with pheromone in order to exploit the best solution. The second difference is pheromone strength τ_{ij} on all the edges will be in the range specified by τ_{min} and τ_{max}, where $\tau_{min} < \tau_{ij} < \tau_{max}$. This is done to avoid the early search stagnation. The updation of best path can be done with either global best or iteration best tour. A new technique called *branching factor* was introduced to determine whether algorithm has converged or not. The λ branching factor is given by the inequality:

$$\tau > \tau_{il}^{min} + \lambda \cdot (\tau_{ij}^{max} - \tau_{ij}^{min}) \tag{1.8}$$

where τ_{ij}^{max} is the maximal, τ_{ij}^{min} is the minimal and τ_{il}^{min} is the minimal trial intensity on arcs exiting from node i. The λ branching factor is the number of out going edges having a trail intensity τ satisfying the equation (1.8). The average branching factor is computed by considering all the edges. If $\tau \approx 1$, then there exists only one edge for exiting node, an indication that ants have found a better

path. In order to explore the new tours, a new mechanism called *smoothing* of trials have been proposed. The smoothing mechanism will adjust the trial intensity on all edges by $\tau_{max} - \tau_{ij}$ factor to facilitate the search in an unexplored region. Further developments on MMAS can be seen in the recent papers (Matthews, 2008; Ivkovic et al., 2011).

Kcc ants and Exponential Local Update (ELU) ants

The Kcc ant and Exponential Local Update ant algorithms are interesting variations to the ant algorithms (Naimi and Taherinejad, 2009). It is based on the following observation: ants have large and equal choice to select the next city at the start of the iteration, but as the tour proceeds, choice to select the next city will be restricted and will be forced to select the best in the available choices. The algorithms consider the quality of decision taken for updating the pheromone trial. The selection of cities at early stage of the tour will have more weightage to pheromone contribution than the later stage and this mechanism have been incorporated as a *local update rule* in the algorithm. The state transition rule for the solution construction phase is given by the equation:

$$p_{ij}^{k}(t) = \frac{[\tau_{ij}]^{\alpha} + [\eta_{ij}]^{\beta}}{\sum_{k \notin tabulist}([\tau_{ik}]^{\alpha} + [\eta_{ik}]^{\beta})} \tag{1.9}$$

This method involves two new local updating rules namely Kcc and ELU. The local updating rule for Kcc is given by the equation:

$$\tau(r,s) = \tau(r,s) + \frac{K.cc.\tau_0}{C_1^{\frac{cc}{\eta}}} \tag{1.10}$$

where cc is the current city number, C_1 is the length of the traveled path, K and η are two parameters which determine the significance of cc and C_1 in the updating process. The local updating rule for ELU is given by the equation:

$$\tau(r,s) = \tau(r,s) + \tau_0 \cdot e^{-\frac{5cc}{M}} \tag{1.11}$$

where M is the number of cities. This method reported best optimal results for some of the TSP datasets.

Table 1.1: An overview of the prominent ACO algorithms for NP-hard problems available in the literature.

ACO Algorithms	**Contributors**	**Year**
Ant System	Dorigo	1992
Elitist AS (EA)	Dorigo	1992
Ant-Q	Gambardella and Dorigo	1995
Ant Colony System	Gambardella and Dorigo	1996
MMAS	Stutzle and Hoos	1996
Rank-based AS	Bullnheimer, Hartl and Strauss	1997
ANTS	Maniezzo	1998
Best-Worst AS	Cordon, Fernandez de Viana and Herrera	2000
Population-based ACO	Guntsch and Middendorf	2002
Beam-ACO	Blum	2004

One can find number of additional ACO algorithms available in the literature, the descriptions of each and every one is not possible in the context of the relevance of work involved in this thesis. The Table 1.1 shows the progress towards ACO in chronological order achieved by the researchers over the years.

1.4.2 ACO: Theoretical Perspective

At the dawn and initial stages of the development of theory of ACO, it was driven by experimental work with the aim of showing that ideas underlying these techniques can lead to successful algorithms. But gradually researchers widened the techniques by giving theoretical support.

The primary question coming to our mind is; will a given ACO algorithm yield an optimal solution? The first convergence proofs were given by Gutjhar (Gutjhar, 2000), who proved the convergence to the optimal solution with probability $1 - \epsilon$. The convergence for ACS and MMAS algorithms has been proved (Dorigo and

Stutzle, 2004; Stutlze and Dorigo 2006). All these convergence results do not have strong base to predict how quickly optimal solutions can be found. Recently, Gutjhar (Gutjhar, 2006) came with an analytical framework that allowed theoretical predictions about the speed of convergence of specific ACO algorithms.

Another portion of ACO theory deals with the establishment of formal links of ACO to other techniques for *learning* and optimization. In this context, one focus was on the relations between ACO and the fields of optimal control and reinforcement learning (Birattari et al., 2002). A differently directed research activities where conducted focusing on connections between ACO and probabilistic learning algorithms such as stochastic gradient ascent (Mealeau and Dorigo, 2002) and the cross entropy (Zlochin et al., 2004) techniques. Even though, these convergence proofs exhibit some relevant mathematical features of algorithms, they usually do not direct the user for the implementation of efficient algorithms.

More relevant research outcomes for practical applications are the ones which aim at better understanding of the behavior of ACO algorithms. The papers due to Blum (2004) and Blum and Dorigo (2005) showed that ACO algorithms in general suffer from *first order deception* in the same way as genetic algorithms suffer from deception. Blum and Dorigo (Blum and Dorigo, 2003) introduced the second order deception, where some solution components on average, receive updates from more solutions than others with which they compete (Blum et al., 2002). The study on the behavior of ACO algorithms by analyzing the dynamics of pheromone model is also an interesting area (Merkel and Middendoff, 2002). Other theoretical issues related to ACO are pheromone updation strategies (Gambardella et al., 1999, Guntsch and Middendorf, 2001), adaptive parameters (Randall, 2004; Pellegrini et al., 2011) and pheromone evaporation strategies (Gambardella and Dorigo, 1995; Gambardella and Dorigo, 1996).

1.4.3 ACO: Application Perspective

Over the years, many successful applications of ACO to a wide range of different discrete optimization problems have been tried. The majority of these applications are on NP hard problems. It has been observed that ACO algorithms are preferred choice to handle these problems. Other popular applications are telecommunication networks (Schoondenwoerd, 1996; Di Caro and Dorigo, 1998; Di Caro, 2004), forest planning (Zeng, 2007) and several subareas of civil engineering (Christodoulon and Ellinas, 2010; Kumar and Reddy, 2006). These applications have motivated the industry people to adapt ACO for solving industrial problems, there by accepting the computational intelligence involved in ACO.

Table 1.2: Applications of ACO algorithms to standard NP-hard problems.

Problem Type	**Problem Name**	**References**
Routing	Vehicle Routing Problem (VRP)	Bullnheimer et al., 1999
		Gajpal and Abad, 2009
	VRP with time windows	Gambardella et al., 1999
		Favaretto et al., 2007
	VRP with loading constraints	Doerner et al., 2006
		Feullerer et al., 2009
Scheduling	Single Machine	Den Bensten et al., 2000
		Holthaus and Rajendran, 2005
	Flow Schedule	Stutzle, 1997
		Yagmahana and Yenisey, 2008
	Job Shop	Blum, 2004
		Huang and Liao, 2008
Subset	Multiple Knapsack	Ke et al., 2010
		Fidanova, 2008
	Bin Packing	Levine and Ducatelle, 2003
		Brugger et al., 2004

Problem Type	Problem Name	References
	Set Covering	Lessing et al., 2004
Assignment	Quadratic Assignment	Maniezzo et al., 1994
		Tsutsui and Liu., 2007
	Graph Coloring	Costa and Hertz, 1997
		Salari and Eshghi, 2008
	MAX-SAT	Pinto et al., 2007
		Villagra and Baran, 2007
Machine Learning	Bayseian networks	De Campos et al., 2002
		Daly and Shen, 2009
	Classification Rules	Parpinelli et al., 2009
		Otero et al., 2008
	Clustering	Azzag et al., 2003
		Herrmann and Ultsch, 2008

Table 1.3: Applications of ACO algorithms to non-standard NP-hard problems.

Problem Type	Problem Name	References
Multiobjective	Portfolio Selection	Doerner et al., 2001
		Gutjahr et al., 2010
	Orienteering	Schilde et al., 2009
		Tricoire et al., 2010
	Knapsack	Alaya et al., 2007
		Ibanez and Stutlze, 2010
Continuous	Neural Networks	Socha and Blum, 2007
		Movahedipour, 2011
	Test Problems	Socha and Dorigo, 2008
Stocastic	Probabilistic TSP	Bianchi et al., 2002
		Bianchi et al., 2005
	Vehicle Routing	Bianchi et al., 2006

Problem Type	Problem Name	References
		Donati et al., 2008
	Screening Policies	Brailsford et al., 2006
Dynamic	Network Routing	Di Caro and Dorigo, 1998
	Dynamic TSP	Cauvery and Viswanatha, 2008
		Guntsch and Middendorf, 2001
		Rais et al., 2007
	Vehicle Routing	Montemanni et al., 2005
		Donati et al., 2008

Even though, ACO is one of the youngest metaheuristics, the number of applications go far beyond we explained above. The Table 1.2 presents an overview of some important applications of ACO algorithms over the years.

1.5 Research Objectives

The following are the main objectives of the thesis.

- To assess the performance of standard ACO variant algorithms by applying punishment mechanism.
- To introduce the decision making ability in ants.
- To explore some unconventional but significantly useful pheromone updation strategies.
- To develop mathematical expression for pheromone evaporation variable.
- To develop a model and solve the train scheduling problem using ACO algorithms.

1.6 Structure of the Thesis

This dissertation is organized into six chapters.

- **Chapter 1** contains the background and objectives of the study.
- **Chapter 2** introduces the extension to Elitist Ant and Rank based ACO algorithm. The first extension is the incorporation of punishment mechanism and second extension is the integration of Machine Learning (ML) philosophy in ACO algorithms. The classification task is employed to select the elite ants.
- **Chapter 3** discusses the unconventional pheromone updation strategies. The nearby tour performances will be updated with the same amount of pheromone trail with the aim to exploit the region near the best solution. The integration of clustering mechanism in ACO algorithm is explained here.
- **Chapter 4** presents some theoretical results on pheromone update mechanism and evaporation of pheromones. This chapter also includes runtime analysis of a modified ACO variant.
- **Chapter 5** demonstrates the applications of ACO described in the earlier chapters to a popular CO problem, Train Scheduling Problem.
- **Chapter 6** summarizes the major contributions of the dissertation and outlines the possible future work with enhancement.