



**T.C.**

**YEDITEPE UNIVERSITY**

**FACULTY OF ENGINEERING & ARCHITECTURE**

**DEPARTMENT OF COMPUTER ENGINEERING**

**ALGORITHMS FROM NATURE**

**&**

**A HYBRID ALGORITHM FOR**

**GRAPH COLORING PROBLEM**

**by**

**Özgür ÜLKER**

**Submitted to the Faculty of Engineering & Architecture in partial fulfillment of  
the requirements for the degree of Bachelor of Science  
in Computer Engineering**

**ISTANBUL - 2003**

**ALGORITHMS FROM NATURE  
&  
A HYBRID ALGORITHM FOR  
GRAPH COLORING PROBLEM**

by

**Özgür ÜLKER**

Approved by:

Assist. Prof. Dr. Ender Özcan .....  
(Supervisor)

Prof. Dr. Sebnem Baydere .....

Fatih Yesilova .....

Date of Approval: ... / ... / 2003

## TABLE OF CONTENTS

LIST OF ABBREVIATIONS .....	V
LIST OF FIGURES .....	VI
LIST OF EQUATIONS .....	VII
LIST OF TABLES .....	VIII
ACKNOWLEDGMENTS .....	IX
ABSTRACT .....	X
ÖZET .....	XI
<b>1. INTRODUCTION.....</b>	<b>1</b>
<b>2. PROBLEMS AND META-HEURISTICS .....</b>	<b>2</b>
<b>2.1. P, NP, NP-COMPLETE, NP-HARD PROBLEMS .....</b>	<b>2</b>
<b>2.2. COMBINATORIAL OPTIMIZATION.....</b>	<b>3</b>
<b>2.3. CONSTRAINT SATISFACTION PROBLEMS .....</b>	<b>4</b>
<b>2.4. ALGORITHMS.....</b>	<b>5</b>
<b>2.5. TABU SEARCH.....</b>	<b>5</b>
<b>2.5.1. Ingredients of Tabu Search:.....</b>	<b>6</b>
<b>2.5.2. Advantages and Disadvantages of Tabu Search .....</b>	<b>8</b>
<b>2.6. SIMULATED ANNEALING.....</b>	<b>9</b>
<b>2.6.1. Algorithm .....</b>	<b>9</b>
<b>2.6.2. Advantages and Disadvantages of Simulated Annealing: .....</b>	<b>10</b>
<b>2.7. GENETIC ALGORITHMS.....</b>	<b>11</b>
<b>2.7.1. Operators of a Genetic Algorithm .....</b>	<b>11</b>
<b>2.7.2. Advantages and Disadvantages of Genetic Algorithms.....</b>	<b>15</b>
<b>2.8. EVOLUTIONARY STRATEGIES .....</b>	<b>16</b>
<b>2.8.1. Mutation Operators:.....</b>	<b>17</b>
<b>2.9. CULTURAL ALGORITHMS .....</b>	<b>18</b>
<b>2.9.1. Algorithm Overview .....</b>	<b>19</b>
<b>2.10. PARTICLE SWARM OPTIMIZATION (PSO).....</b>	<b>21</b>
<b>2.10.1. The Algorithm.....</b>	<b>21</b>
<b>2.10.2. Advantages and Disadvantages of PSO.....</b>	<b>22</b>
<b>2.11. ANT COLONY OPTIMIZATION.....</b>	<b>22</b>
<b>2.11.1. Ant colony optimization method.....</b>	<b>23</b>
<b>2.11.2. ACO Metaheuristics.....</b>	<b>24</b>
<b>2.12. MEMETIC ALGORITHMS .....</b>	<b>26</b>

2.13.	NEURAL NETWORKS .....	28
2.13.1.	<i>Human Brain</i> .....	28
2.13.2.	<i>Artificial neurons</i> .....	30
2.13.3.	<i>Types of activation functions:</i> .....	31
2.13.4.	<i>Neural Network Structures</i> .....	31
2.14.	MEMBRANE COMPUTING (P SYSTEMS) .....	33
2.14.1.	<i>Biological Foundation of Cell Membranes</i> .....	34
2.14.2.	<i>Membranes in Membrane Computing (P Systems)</i> .....	35
3.	GRAPH COLORING PROBLEM.....	37
3.1.	HEURISTIC METHODS FOR GRAPH COLORING.....	37
3.2.	HYBRID EVOLUTIONARY TABU SEARCH (HET) FOR GRAPH COLORING.....	38
3.2.1.	<i>Representation</i> .....	39
3.2.2.	<i>Initialization</i> .....	39
3.2.3.	<i>Selection</i> .....	40
3.2.4.	<i>Crossover</i> .....	40
3.2.5.	<i>Mutation</i> .....	43
3.2.6.	<i>Local Search Procedure</i> .....	43
3.2.7.	<i>Replacement:</i> .....	44
3.3.	PROGRAMMING ENVIRONMENT C++ .....	45
3.4.	DATA STRUCTURES USED IN THE ALGORITHM .....	46
3.4.1.	<i>Set</i> .....	46
3.4.2.	<i>Individual</i> .....	47
3.4.3.	<i>Population</i> .....	48
3.4.4.	<i>Implementation Objects</i> .....	49
3.5.	EXPERIMENTS.....	50
3.5.1.	<i>DIMACS Graph Format</i> .....	50
3.5.2.	<i>Experimental Data</i> .....	52
3.5.3.	<i>Test Parameters</i> .....	53
3.6.	APPLICATION OF GRAPH COLORING INTO OTHER PROBLEM TYPES.....	58
4.	CONCLUSION AND FURTHER WORK.....	59
	REFERENCES .....	61

## List of Abbreviations

ACO	Ant Colony Optimization
ANN	Artificial Neural Networks
CA	Cultural Algorithms
COP	Combinatorial Optimization Problem
CSP	Constraint Satisfaction Problem
DIMACS	Discrete Mathematics and Computer Science
ES	Evolution(ary) Strategies
GA	Genetic Algorithms
GCP	Graph Coloring Problem
GPX	Greedy Partition Crossover
HET	Hybrid Evolutionary Tabu Search
LS	Local Search
MA	Memetic Algorithms
MC	Membrane Computing
NDTM	Nondeterministic Turing Machine
PSO	Particle Swarm Optimization
SA	Simulated Annealing
TS	Tabu Search
WCSC	Weighted Color Sets Crossover

## List of Figures

Figure 1 - An example of a suboptimal solution .....	7
Figure 2 - Various Genetic Algorithm Representations .....	12
Figure 3 - One point Crossover at location 3.....	13
Figure 4 - Traditional uniform crossover .....	13
Figure 5 - Mutation at location 4 .....	14
Figure 6 - A rather non uniform problem domain .....	27
Figure 7 - A typical biological neuron.....	29
Figure 8 - Nonlinear model of a neuron.....	30
Figure 9 - A Biological Plasma Membrane .....	34
Figure 10 - An Artificial Membrane Structure .....	35
Figure 11 - Example representation of two solution constituted of different independent sets .....	39
Figure 12 - An independent color set. ....	46
Figure 13 - The Structure of an Individual Object.....	47
Figure 14 - The Structure of a population .....	48
Figure 15 - Algorithm Objects Hierarchy.....	50
Figure 16 - A DIMACS ASCII graph instance .....	51

## LIST OF EQUATIONS

Equation 1 - Boltzmann Probability .....	9
Equation 2 - Selection Pressure on an individual in Rank Based Selection.....	40
Equation 3 - Weighting function in the Weighted Sets Crossover.....	42
Equation 4 - Neighborhood Percentage Calculation .....	44

## LIST OF TABLES

Table 1 - Miles1000 Graph with Tabu Length 100 iterations .....	55
Table 2 - Miles1500 Graph with Tabu Length 100 iterations. ....	55
Table 3 - DSJC and Leighton graphs Tabu Length 100 iterations .....	56
Table 4 - Effect of tabu Tenure on miles1000 graph.....	56
Table 5 - Effect of tabu lenght on miles1000 graph.....	57

## **Acknowledgments**

I sincerely thank Assistant Prof. Ender Özcan for his guidance throughout the project. I also thank my family and my research comrades in ARTI (Artificial Intelligence Laboratory) for their friendship throughout my university career.

## **Abstract**

Many difficult problems most of which belonging to NP-Complete family exist in the domain of computing, engineering, natural and social sciences. Traditional methods fail in solving these problems within practical time and storage limits. Clever heuristic search techniques consequently become necessary in solving these within reasonable resources.

The aim of this thesis is to make a survey of alternative algorithms derived from the nature in order to propose a solution method to these problems mainly in Artificial Intelligence, Combinatorial Optimization and in Operations Research. The inner mechanics of each algorithm is presented in detail by means of biological foundation. The implementation of each algorithm according to the natural phenomenon has been explained. The advantages and disadvantages of the algorithms with respect to each other are mentioned briefly. Finally, the application areas of these methods have been presented. Some comparative study has also been carried out.

For practical experimentation of these algorithms, two very different algorithms in nature are selected to solve the Graph Coloring Problem. We have experimented on GCP to create a new and efficient algorithm in terms of solution quality and time. A new variant of a crossover operator has also been proposed.

## Özet

Günümüzün bilgisayar, teknoloji, mühendislik ve sosyal bilimler dünyası pek çok zor problemlerle doludur. Geleneksel metodlar bu problemlerin çözümünde sistem kaynakları açısından yetersiz kalmaktadır. Dolayısıyla bu problemlerin çözümünde akıllı başka yöntemlerin kullanılması gerekmektedir.

Bu tezin amacı da bu doğadan ilham alınarak oluşturulmuş bu alternatif metodların incelenmesidir. Her algoritmanın doğanın neresinden esinlendiği ve doğal olguya nasıl dayanarak uygulandığı araştırılmıştır. Her algoritmanın iyi ve kötü yönleriyle nerelerde kullanılabilirliği açıklanmıştır. Ayrıca bazı karşılaştırmalı açıklamalar da yapılmıştır.

Bu algoritmaların pratik bir uygulaması olması açısından, Çizge Boyama Probleminin çözümünde iki tane doğal algoritmanın birleştirilmesinden faydalanılmıştır. Bu problemin çözümünde zaman ve verimlilik açısından güçlü yeni bir algoritma geliştirilmiştir. Ayrıca Çizge Boyama Problemi için yeni bir çaprazlama uzmanı da önerilmiştir.

## **1. Introduction**

Faced with the challenge to solve the hard problems surrounding us, classical methods often encounter great difficulty. Vitrally important applications in business, engineering, economics and science cannot be tackled with any reasonable hope of success, within practical time horizons, by solution methods that dominate the focus of the academic community throughout the past four decades. For instance an exhaustive search method will generally be infeasible, as required computational power will never be present. For example, a scheduling problem for distributing workloads among machines to assure all processing demands are adequately met over the time horizon may cover 300 to 500 variables and approximately 100 constraints. By the time an exhaustive search to solve this problem ends, the sun will have burnt out all of its hydrogen and will die out. As a result we need clever heuristics to search the large problem domain.

Naturally various methods have been developed to solve different difficult problems that cannot be solved in polynomial time (NP Complete Problems). This is where the algorithms inspired from the nature come into account. For 4.5 billion years nature handles its own problems consisting of millions of variables and constraints to build efficient solutions. Naturally, a clever way is to imitate the way the nature handles its problems. The imitation method can cover researching the behavior of birds, fish, ants etc, the perception of the brain, annealing of atoms, theory of evolution etc. Therefore the inner mechanics of each natural system deserves a scientific and thorough investigation.

In this thesis, we deal with each of these natural systems. Section 2 gives detailed information about the algorithms and the problems they are used for. Section 3 discusses about the well known combinatorial optimization problem, the Graph Coloring, and a proposed method for it which is derived from the hybridization of two selected algorithms. Finally Section 4 discusses the enhancements and directions for further research.

## 2. Problems and Meta-heuristics

There are many more types of difficult problems in the area of computing. The difficulty context assumed here is mathematical complexity in terms of computation or storage time. We first give the rigorous definitions of the most general concepts than go deeper into our main research area, combinatorial optimization and constraint satisfaction problems. Then we shift our interest into various algorithms, each explained in details.

### 2.1. P, NP, NP-Complete, NP-Hard Problems

**P** is the set of all problems that can be solved in polynomial time on a (deterministic) Turing Machine.

**NP** is the set of all problems that can be solved in polynomial time on a nondeterministic Turing machine (NDTM). Because deterministic TMs are a subset of NDTMs, P is a subset of NP.

A problem is **NP-hard** if an algorithm to solve it in (deterministic) polynomial time would make it possible to solve all NP problems in polynomial time. NP-complete is the class of problems which are both NP-hard and themselves members of NP.

NP-hard problems are *at least as hard* as NP-complete ones. Consider this problem: solve *every* minimal graph-coloring problem of size  $n$ . Clearly, solving this problem in polynomial time would also allow one to solve the NP Complete minimal graph-coloring problem (and thus all NP problems) polynomially. However, the problem given is not itself a member of NP (it is too hard), so it is NP-hard but not NP Complete.

NP-complete problems have the property that any one of them can be reduced to any other in polynomial time. Thus if one can solve NP Complete problem A in polynomial time, he can also solve NP Complete problem B in polynomial time by first converting it to an equivalent instance of problem A in polynomial time, and then solving that problem. (Also, to prove that a problem is

NP-complete, it is sufficient to show that it is polynomial-time equivalent to some other NP-complete problem.)

Finally, any NP-complete problem can be solved by a TM in time no worse than  $O(2^{p(n)})$ , where  $p(n)$  is the number of steps used by an NDTM for the problem. This is possible by just simulating the NDTM.

An equivalent definition of NP is the set of problems for which a proposed solution can be verified or rejected in (deterministic) polynomial time.

## **2.2. Combinatorial Optimization**

Combinatorial optimization is involved with models and methods for optimization over discrete choices. It is rooted in the theory of linear programming, and has strong links with discrete mathematics, probability theory, algorithmic computer science, and complexity theory. Some problems in the area are relatively well understood and admit solution to optimality in polynomial time. Many others are NP-hard, and one is forced to go one of three ways. Either one chooses an enumerative method that is guaranteed to produce an optimal solution. Or one applies an approximation algorithm that runs in polynomial time. Or one resorts to some type of heuristic search technique, without any a priori guarantee in terms of solution quality or running time.

Two developments outside the area stimulated research in combinatorial optimization. First, the continued increase in computing power strengthened the need for efficient algorithms. The ability to handle bigger problems made the distinction between low and high order running times more pronounced and diminished the power of brute force. Second, at the application side, there has been an increasing confidence in the practical potential of optimization techniques. Large and difficult real-world problems that were out of reach ten years ago are now being solved. Notable examples occurred in airline crew scheduling, train timetabling, time-constrained vehicle routing, telecommunication network design, frequency allocation, VLSI layout synthesis, and statistical disclosure control.

### 2.3. Constraint Satisfaction Problems

A constraint satisfaction problem prescribes some requirements for a finite number of variables in the form of constraints. The set of possible values — the domain — for each variable is finite. A constraint tells which value tuples are allowed for a certain subset of all the variables. A constraint can be given either explicitly, by enumerating the tuples allowed, or implicitly, e.g. by an algebraic expression. The solution of a CSP is an instantiation of all the variables for which all the constraints are satisfied. A CSP is solvable if it has at least one solution; otherwise it is unsolvable or over-constrained.

Solving a CSP is usually understood as the task of providing a single solution for the problem. However, there are cases when one would like to get all the solutions. In the case of constraint optimization problems, the best solution are to be found, namely the one with the optimal value of a given optimization function. In some situations one is not interested in the solutions themselves, but in the number of solutions, particularly, if the problem is solvable or not.

After this brief introduction to CSP, the formal definition of a CSP can be presented as follows:

A *Constraint Satisfaction Problem (CSP)* is a triple  $\langle X, D, C \rangle$ , where:

- (i)  $X = \{ x_0, \dots, x_n \}$  is the set of *variables*.
- (ii)  $D = \{ D_0, \dots, D_n \}$  is the set of *domains*. Each domain is a finite set containing the possible values for the corresponding variable.
- (iii)  $C = \{ C_0, \dots, C_n \}$  is the set of *constraints*. A constraint  $C$  is a relation defined on a subset  $\{ x_0, \dots, x_n \}$  of all the variables, that is  
 $D_0 \times \dots \times D_n$  over  $C$ .

Given a (partial or complete) instantiation of the variables, the constraint  $C$  is *satisfied* if all the  $x_0, \dots, x_n$  variables got a value and such that the corresponding value tuple belongs to  $C$ . A

*solution* of a CSP is such a complete instantiation of the variables that all the constraints are satisfied. If for a CSP there is at least one solution, then the problem is *solvable*, otherwise *unsolvable*, or *inconsistent*, or *over constrained*. The set of all possible complete instantiations, that is,  $D_1 \times \dots \times D_n$  is often called the *solution space*, in the sense that the solution should be searched for in this space.

## 2.4. Algorithms

The below summarizes the algorithms surveyed during the project list. Note that this list does not cover all of the algorithms that were inspired from the nature.

It should be underlined that the below algorithms are not suitable to solve all difficult problems. These algorithms are just heuristic methods based on the natural metaphor, therefore they cannot guarantee that an optimal solution exist for all of the problem instances. Most of the problems are so hard that many of these heuristics will probably get stuck at suboptimal points. Fortunately most of the real life problems are not so optimal solution demanding (that we do need the exact solution), therefore we can settle with near optimal results.

## 2.5. Tabu Search

Tabu Search, first proposed by Fred Glover, [13][14] is a local search based algorithm which takes its inspiration from the human memory, that humans regards the history to perform future actions. The natural inspiration can be explained further in two steps, the cultural background of the tabu notion, and the associative memory derived from this context. The notion of tabu can be explained as follows:

[15]The Tongan (a Polynesian language) word Taboo is used to indicate things that cannot be touched because they are sacred. According to Webster dictionary, the word also means “a prohibition imposed by social custom as a protective measure” or of something “banned as constituting a risk”. The risk to be avoided in this case is that of following a counter-productive course, including one which may lead to entrapment without hope of escape. On the other hand,

as in the broader social context where “protective prohibitions” are capable of being superseded when the occasion demands, the “tabus” of tabu search are overruled when evidence of a preferred alternative becomes compelling. This notion will become more important once we move on with the Tabu Search.

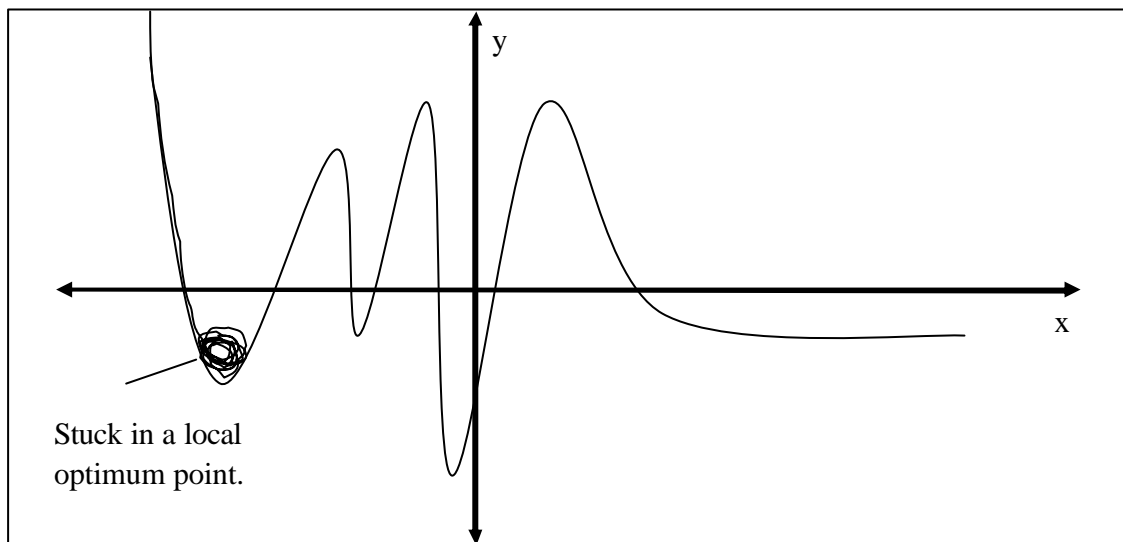
The most important aspect of this traditional usage is that tabus are transmitted by means of a social memory which is subject to modification over time. This creates the link to the notion of tabu in the context of tabu search. These forbidden elements receive their status by an evolving memory, which allows this status to shift according to time and circumstances. More particularly tabu search is based on the fact that an intelligent problem solving method has to incorporate adaptive memory and responsive exploration. The emphasis on responsive exploration in tabu search, whether in a deterministic or probabilistic implementation, derives from the fact that a bad strategic choice can give more information than a good random choice. In a system that uses memory, a bad strategic choice can provide clues about how the strategy may be changed with a gain. This adaptive memory notion can efficiently be used in problem solving as explained in Section 2.5.1.

### **2.5.1. Ingredients of Tabu Search:**

Approaching a problem by using tabu search begins with the identification of a neighborhood structure which can be reached by a single move from a given configuration. A move can be defined as the transition action which transforms a given configuration into another one. Thus the set of all future configurations which can be reached within a single move from an initial configuration is defined as the neighborhood of configuration.

A Tabu Search algorithm then moves from one configuration to another by means of a move (by using a steepest descent or mildest descent approach) and usually accepting the best move which results in the best improvement or least deterioration from the objective function value. This method resembling a simple hill climbing method suffers from some deficiencies. Without any control mechanisms, the process can get stuck at local optimal points rather than

converging to a global optimum. The main reason for this problem is the cycles which occur from revisiting a previously visited configuration thus ending in an infinite loop because of violently circulating between a set of configurations.



**Figure 1 - An example of a suboptimal solution**

In order to overcome this problem, tabu search uses a recency based short term memory structure, namely a tabu list. The aim of tabu lists is to store the attributes of the previously performed moves and to use these attributes to help to determine to set some future moves related with these attributes as forbidden (tabu). A most natural and most common tabu move is to set the inverse of a performed move as tabu, thus preventing the cycles from happening. Therefore we restrict ourselves to non-tabu moves (admissible moves). Thus the search continues in the non-tabu neighborhood.

Evidently, the key of tabu search resides in tabu list management, which is not clearly defined in the algorithm. One of the most prominent topics is the tabu tenure, the maximum time in which a tabu move will remain as forbidden. Another fact is which attributes or classes of attributes should be recorded in the tabu structures. In addition many tabu search algorithms also define additional memory structures based on frequency (modulated by a notion of move influence) and coordination of these memory elements is made to vary as the preceding short

term memory component becomes integrated with longer term components. The aim of this integration is to provide a balance between two global interaction strategies, namely intensification strategies and diversification strategies.

Intensification strategies are based on modifying choice rules to encourage move combinations and solution features historically found good. They may also initiate a return to attractive regions to search them more thoroughly. Since elite solutions must be recorded in order to examine their immediate neighborhoods, explicit memory is closely related to the implementation strategies.

It should be noted that intensification strategies generate neighbors by either grafting together the components of good solutions or by using modified evaluation strategies that favor the introduction of such components into a current (evolving) solution. The diversification stage on the other hand encourages the search process to continue examining unvisited regions and to generate solutions that differ in various significant ways from those seen before.

### **2.5.2. Advantages and Disadvantages of Tabu Search**

Tabu Search improves greedy local search method by avoiding getting trapped in a local optimum in the early stages of the search phase. It also offers extension of tabu list structure and rules a specific problem. These structures may be very helpful for Constrain Satisfaction Problems for fast constraint checking.

The downside of Tabu Search is it just uses a heuristics movement yet it can not guarantee anything in strict sense, thus there is no guaranteed convergence to a global optimum when compared to Simulated Annealing. Another important problem with TS is that the parameters although few in number should be adjusted according to the specifics of the problem however there are no general rules that would tell how to do it; much of this empirical testing process is left to the intuition of the researcher.

## 2.6. Simulated Annealing

Simulated Annealing (SA), [20] like Tabu Search, is another local search based algorithm to overcome the shortcomings of the cycles and sticking at the local optimal points. Simulated Annealing's inspiration comes from the statistical thermodynamics, the behaviour of the system in the limit of low temperature.

In order to understand the basics of simulated annealing, it is somewhat necessary to understand the basics of annealing behavior of the atoms. In statistical thermodynamics a fundamental problem is to determine the behavior of the system in the limit of low temperature. This usually covers the systems where atoms remain fluid or solidify and if so whether they form a crystalline structure. This process is generally carried out by a careful annealing method, thus melting and slowly cooling the substance until it reaches a strong crystalline structure, with few amorphous deficiencies. The behavior of the atoms during the annealing process is generally approximated with a Boltzmann probability factor, as seen in Equation 1. This process can be simplified as follows:

The system always accepts a configuration if the energy level of the system decreases.

If the energy level does not decrease, then the configuration is accepted with the following Boltzmann probability.

$$e^{-E/k_b T}$$

**Equation 1 - Boltzmann Probability**

where  $k_b$  is the Boltzmann coefficient, T is the temperature in Kelvin, and E is the energy change.

### 2.6.1. Algorithm

Applying this metaphor of physics to computing begins with Metropolis in the earliest days of scientific computing. Metropolis devised a simple and efficient algorithm that simulates the atoms in equilibrium at a given temperature. In each step of his algorithm, an atom is

given a random displacement (analogous to a local search move) and the generated change in the system energy is computed. If the energy change is less than zero, the resultant configuration is always accepted and the next step in algorithm continues from that atom. If not, then a Boltzmann probabilistic is introduced (as seen in Equation 1) to determine whether that configuration will be accepted or not. By iterating these basic steps, it is possible to simulate the behavior of the thermodynamical motion of the atoms at a temperature  $T$ .

The simulated annealing is built upon this Metropolis algorithm to solve optimization problems, by using a cost function instead of energy level, and setting parameters to generate a set of configurations. The simulated annealing procedure first melts the system that will be optimized (thus initializing the temperature to a level  $T$ ) and then lowering the temperature slowly to reach a freezing point (a stage that no change in cost occurs in the system afterwards). The trick of simulated annealing is to stay at some temperature for a period of iterations which is long enough (which is called thermalization or decorrelation time). If simulation does not proceed in some temperature as long enough, then similar to amorph structures in nonuniform crystallizations in physics, local deficiencies may dominate the search procedure (getting stuck at local optimal points, cycles etc). In theory, if the temperature is decreased infinitely slowly the solution will always converge to a global optimum. However, if the process continues for a long time in some temperature, then the algorithm will take a long time to converge to a global solution.

### **2.6.2. Advantages and Disadvantages of Simulated Annealing :**

SAs are simple algorithms to implement. They only require the definition of the neighborhood operator, the temperature conditions and may be restructured with a Boltzman probability function. Consequently SAs do not have that much algorithm parameters compared to others. SAs also do not need much insight into the problem structure enabling them to be easily extended to a wide range of applications. SAs generally produce reasonable if not the best solutions.

However, SAs also have important deficiencies. Efficiency of an SA algorithm is predominantly dependent on the temperature scheduling. With poor temperature management, the process may be prevented from sufficient exploration of the search space. The theoretical assumption that the solution will always converge to a global optimum if the temperature is lowered infinitely slowly is impractical, because even taking a relatively very small temperature decrement will increase the convergence rate to a solution gradually. Consequently, detailed experimentation, most importantly on temperature management will become necessary.

## **2.7. Genetic Algorithms**

Genetic Algorithms (GA), first proposed by John Holland in 1975, is an evolutionary population based algorithm which takes its inspiration from Darwinian evolution. Genetic algorithm uses the ‘survival of the fittest’ phenomena to solve problems. The genetic algorithms can be roughly defined as non-deterministic heuristic search strategies [25] that are based on iteratively evolving a population of individuals, each of which encodes one of the solutions to a given optimization problem, towards more promising zones of the space of solutions of the optimization problem.

### **2.7.1. Operators of a Genetic Algorithm**

- a) *Representation*: [5] The first step in every genetic algorithm is to decide on how to represent a configuration. The most common forms are binary coding (representing with binary numbers), decimal representations (representing with floating points), order based representations (which is generally used in problems in which a permutation of configuration elements exist as in Traveling Salesman Problem), and group based representations (a whole configuration constitutes of smaller groups). Deciding on a representation is clearly highly problem specific and careful observation of the inner mechanics of the problem is mandatory.

Figure 2 shows various representation methods.

<b>Binary Representation</b>	0	1	0	1	0	1	0	1									
<b>Floating Point Representation</b>	0.23	4.53	2.32	1.24	4.55	1.23	2.45	5.54									
<b>Order Representation</b>	0	→	4	→	2	→	7	→	1	→	8	→	3	→	5	→	4
<b>Group Representation</b>	0	4	1	7	2	3	6	8									

**Figure 2 -Various Genetic Algorithm Representations**

- b) *Selection*: The second step is to select (generally) two individuals (chromosomes) of the population for crossover (see crossover below). Selection methods usually differ from each other in terms of selection pressure, diversity preservation, convergence rate, etc. The most common three selection methods are
- i. **Roulette Wheel Selection**: The selection is made with a probability directly proportional to the fitness. (No standardization is applied on fitness). Negative effects of this selection are slow convergence if population is constituted of similar individuals or premature convergence if super individuals (with very good fitness values compared to others) exist in the population.
  - ii. **Rank Based Selection**: The difference of rank based selection from roulette wheel selection the individuals are selected with respect to their ranks of fitness not their actual fitness. Thus it is unimportant how fitter an individual from others as long as it is fitter. Rank based selection is generally the most common selection method in the literature.
  - iii. **Tournament Selection**: Tournament selection is similar to rank based selection but it is devised for parallel implementations. The population is divided into smaller groups and selection is carried out inside these groups (a tournament) independent of other groups.

c) *Crossover*: Perhaps the most important ingredient of a genetic algorithm is the crossover (recombination) of two selected individuals to form new individuals (offspring). Crossover can be seen as combination of good attributes of two individuals for the intention to form better off springs. The most common crossover types are as follows:

- i. n-ptx crossovers: The traditional n-ptx crossovers are usually applied to problems where binary representations are used. N points are chosen from the chromosomes and genes are exchanged between donating individuals between those points.

0	1	1	1	0	1	0	1	Chromosome 1
0	1	0	0	1	1	0	0	Chromosome 2
0	1	1	0	1	1	0	0	Offspring 1
0	1	0	1	0	1	0	1	Offspring 2

**Figure 3 -One point Crossover at location 3.**

- ii. Uniform crossover: Uniform crossover uses a Bernoulli principle to determine which gene value will be given to the offspring.

0	1	1	1	0	1	0	1	Chromosome 1
0	1	0	0	1	1	0	0	Chromosome 2
1	0	0	0	1	1	0	1	Bernoulli Vector
0	1	1	1	1	1	0	0	Offspring 1
1	0	0	0	0	0	0	0	Offspring 2

**Figure 4 -Traditional uniform crossover**

- iii. Order based crossovers: Usually used in order based problems such as TSP, the idea is to preserve and transmit the smaller highly fit order structures into the offspring.
- iv. Group based crossovers: Used in group based representations, the intention is to preserve and transmit the good groups to the offspring.

d) *Mutation*: Mutation operator is, unlike crossover operator, a passive mechanism whose main purpose is to add some diversity into the gene pool. Mutation is again usually related with the representation, however the main action is to select some gene value and replace it with another random one. As it is a backward operator, mutation rate is given very low values by the researchers with respect to crossover. Also many researchers have omitted the mutation operator from their algorithm and evolve the population by means of crossover.

1	2	3	4	5	6	7	8	Bit No
0	1	1	1	0	1	0	1	Chromosome 1
0	1	1	0	0	1	0	1	Mutated Chromosome

**Figure 5 -Mutation at location 4**

e) *Fitness Calculation*: Next phase is to determine how good a created offspring is. In evolutionary algorithms, fitness is the value attached to the individual to determine how far it is from the desired solution or how close it is thought to be to the searched solution. Fitness calculation is generally directly related with the costs attained or costs that are standardized by mathematical relation.

f) *Replacement*: The last step of a genetic algorithm iteration is replacement. The important question is how to add the newly created offspring after crossover and mutation into the population. Some accepted schemas are:

- i. Generational GAs: In this schema, offsprings whose numbers are equal to the population size are created and whole population is replaced by the new offsprings.
- ii. Steady State GAs: Only two offspring is created from the two selected parents and those two offspring replaces their parents.
- iii. Elitist Strategies: Main intention of this strategy is to preserve highly fit individuals in the population. These individuals are not replaced until better individuals appear in the population.

### **2.7.2. Advantages and Disadvantages of Genetic Algorithms**

Genetic Algorithms are general algorithms which can easily be applied to a wide range of applications. This direct application directly stems from the simple configuration and simple operations which are generally revolving around binary strings. GAs therefore give a good approximation of the optimal solution to a problem even if there are a large number of discrete or continuous parameters. GAs are also capable of optimizing just about any cost function with few, if any, mathematical limitations including complex multi-dimensional optimization problems, for which few other techniques can give any information

Another advantage of GAs is being easy to implement on parallel architectures because of its population based structure. The individuals and operations on them can be distributed within many processing units with ease.

Despite the well known advantages of genetic algorithms, some disadvantages have been broadly reported as well. Probably the most important drawback of genetic algorithms is their strong dependence on a set of parameters (e.g., size of the population, number of generations, probabilities for applying the random operators, rate of generational reproduction, etc.) that have to be experimentally tuned for the optimization problem at hand. Consequently, unless the user has experience in the resolution of the concrete optimization problem at hand by means of genetic algorithms, the choice of the most suitable values for all the parameters is converted itself into an optimization problem.

Since a GA works with populations of individual solution attempts, and each individual must be evaluated, a GA can be computationally expensive, with some problems requiring days or weeks to run. GA's are often still faster than a "brute force" combinatorial approach, however. Moreover, GA's can be applied to searches that are simply too large to be investigated any other way.

If the optimal region (but not point) of the search space is known, it may be difficult to direct the GA towards it. This is because of the high level of randomness associated with the operations of a GA.

Due to the large amount of randomness in the operation of a GA, it is not guaranteed to find the optimal solution. It may be somewhat wiser using a GA to find the optimal region of the solution space, then using combinatorial (brute force) methods to find the optimal solution.

## **2.8. Evolutionary Strategies**

Evolution(ary) Strategies (ES), [1] very similar to genetic algorithms, takes its inspiration also from Darwinian evolution. ES were first proposed by Rechenberg in 1965, and devised a similar but independently developed theory very similar to Genetic Algorithms. ES were frequently used for optimization of highly complex, multimodal and non differentiable functions.

Main difference of ES and GAs lies in their simulation of the evolution. In ES, individuals represent phenotypes whereas in GAs, they represent chromosomes (genotypes). In function optimization, GAs usually deals with binary strings, whereas ES operates on real valued vectors.

Another important difference of ES and GAs is in their search operators. As ES usually operates on real valued vectors, ES simulates evolution with mainly mutation, where GAs also use crossover. Mutations in ES are rather much more developed then their counterparts in GAs and plays a very active role in evolution.

The crossover operation in ES is defined as the recombination mechanism. As an ES algorithm operates on real valued vectors, it is not clear how to cross over two individuals by means of traditional n-ptx crossover. Therefore, recombination deals with the real valued mathematical operations. One example can be given as intermediate recombination where two parents are recombined by taking the weighted average of their allele values (gene value at a specific gene location) and the standard deviation.

The frontier methods in ESs are  $(\mu+\lambda)$ -ES and  $(\mu,\lambda)$ -ES, where  $\mu > 1$ ,  $\lambda > 1$ . In  $(\mu+\lambda)$ -ES,  $\mu$  parents produces  $\lambda$  offsprings. Then  $\mu+\lambda$  individuals will be reduced to  $\mu$  individuals of the next generation by selection. The advantages of the  $(\mu+\lambda)$ -ES are the ease of using adaptive search parameters by means of variances of the Gaussian random numbers used in ES.

However  $(\mu+\lambda)$ -ES suffers from an important disadvantage that it often gets stuck in local optimum. It favors small variances if certain conditions are true, thus it is susceptible to cycles. The  $(\mu+\lambda)$ -ES overcomes this difficulty by selecting only  $\lambda$  offsprings and replacing  $\mu$  parents. This schema is similar to the generational GAs discussed in GAs.

### 2.8.1. Mutation Operators:

Since ES simulate natural evolution by means of mutation, many different mutation operators have been proposed by many researchers. As ES operates on vectors of floating point representation most of the mutation operators are highly enriched with statistical and mathematical foundations. We briefly discuss the most common mutation operators[24]

- Gaussian Mutation:

The most well known and popular mutation operator is the Gaussian Mutation which modifies all components of the solution vector  $x = (x_1, x_2, \dots, x_n)$  by adding a random noise of

$$x^{t+1} = x^t + N(0, \sigma)$$

where  $N(0, \sigma)$  is a vector of independent random Gaussian numbers with a mean of zero and standard deviation of  $\sigma$ .

- Non-Uniform Mutations where

$$X_k^{t+1} = \begin{cases} x_k^t + \Delta(t, r(k) - x_k) & \text{if random binary digit is 0} \\ X_k^t - \Delta(t, x_k - l(k)) & \text{if a random binary digit is 1} \end{cases}$$

For  $k = 1, \dots, n$ . The function returns a value in the range  $[0, y]$  such that the probability of  $\Delta(t, y)$  being close to 0 increases as  $t$  increases ( $t$  is the generation no).

Uniform Mutation:

Uniform mutation changes a single component of the solution vector; e.g., if  $x^t = (x_1, \dots, x_k, \dots, x_n)$  then  $x^{t+1} = (x_1, \dots, x'_k, \dots, x_n)$  where  $x'_k$  is a random value (uniform probability distribution) from the domain of variable  $x_k$ .

## 2.9. Cultural Algorithms

Cultural Algorithms (CAs) [28] are a class of computational models which derive their behaviors from a metaphor of the processes of evolution in human culture. Cultural Algorithms were first proposed by Dr. John Reynolds and his colleagues in Wayne State University. The CAs deal with the concept of the culture therefore it is supplied by the culture definition.

Culture is defined by Durham as a "system of symbolically encoded conceptual phenomenon that is socially and historically transmitted within and between populations". It has been suggested that over time humans have evolved a unique set of capacities that support the formation, encoding, and transmission of cultural information. A key factor underlying all of these capacities is the formulation of categories. Although the categories are represented by symbols, it is suggested that the ability to formulate the categories via experience is the critical factor in the development of a cultural capacity. The symbolization of an individual's past experience and forecasts concerning future experiences is internalized as a world map, or "mappa" in the *THINKS* model. These individual mappa can be merged, generalized, and specialized in order to form group mappa. These group mappa serve to direct the future actions of the group and its individuals.[29]

In recent years, researchers have attempted to model this cultural evolution process from both a micro-evolutionary perspective in terms of the transmission of behaviors or traits between individuals in a population and a macro-evolutionary perspective in terms of the formation of generalized beliefs based upon individual experiences. These generalized beliefs can serve to constrain the behaviors of individuals within the associated population. A dual inheritance

cultural system supports the transmission of information at both the individual and group level. Cultural Algorithms are a class of computational models of cultural evolution that support such a dual inheritance perspective. This approach provides a framework in which to describe all of the current models of cultural evolution from a computational point of view since any of the single inheritance systems can be produced as a special case.

### **2.9.1. Algorithm Overview**

In the model presented here, each individual can be described in terms of a set of traits or behaviors and a mappa or generalized description of their experiences. Traits can be modified and exchanged between individuals by means of a variety of socially motivated operators. Likewise, individual mappa can be merged and modified to form "group mappa". Various merging and modification operators are possible and will be discussed later. Operators for the modification of traits and mappa can be either generic (problem independent) or problem specific.

The symbols used to characterize traits and mappa can also be modified over time based upon experience. It is possible that traits can be lost from the population or added. In addition, symbols that are used to represent mappa can be forgotten and new symbols added. Thus, the representation of the trait sequences and the mappa can themselves evolve as a result of the groups experiences.

At any given time step in the model there are a set of individuals in the population space, each described in terms of currently applicable traits. The performance of each individual in solving a set of selected problems is *EVALUATED*. In addition, each individual will produce a generalized map of their experience during that time period. This process is called *OUTLINING*. The most general belief in a generated mappa is called its dominant belief. An individual's mappa can then be merged with currently existing group mappa in the belief space if the conditions for one or merging operations are met. If it cannot merge, it remains separate for that time step in the belief space. When mappa are merged, the performances of

the individuals associated with them are *COMBINED* in some fashion. If the combined performance of a mappa is less than some *ACCEPTABLE* level then that mappa is discarded or *PRUNED* from the belief space. The *ADJUSTED* belief space at a given time step is the set of currently *ACCEPTABLE* group mappa. Discarded mappa can be enforced or not at the population level. If the discarded mappa is enforced, then no individuals possessing the associated beliefs are allowed in the group in the future. If the discarding of mappa is not enforced by the group, then individuals with associated beliefs can reappear in future populations.

The current state of the belief space can then be used to modify the performance of individuals in the population, modify the set of allowable traits, enforce discarded mappa etc. The population is then used to generate a new population through the *SELECTION* of individuals to be parents for the next generation. These parents are used to *EVOLVE* a new population via the application of various modification operators. As such, the process is inherently parallel since it is possible that there can be many group mappa residing in the belief space, each supported by a subset of the current population and exploiting some niche or portion of the current problem solving environment. The processes of *OUTLINING*, and *MERGING* and *PRUNING* together with the *COMBINE* and *ACCEPTANCE* functions determine how the space of beliefs will be searched in parallel by individuals from the population.

How the current belief space affects the population of individuals and how individuals in turn affect the belief space is mediated by the nature of the communication channel or *PROTOCOLS* that interconnect them. There are several possible protocols. The standard protocol, *VOTE-INHERIT-PROMOTE*, supports the process of associating the performance of an individual with a mappa in the belief space(*VOTE*), then allowing the mappa to *INHERIT* the individuals performance, and finally *PROMOTING* those individuals in the population associated with current group mappa. Other protocols involve changing trait and belief representations.

## 2.10. Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) [19] takes its inspiration from two main component methodologies. It is tied with the artificial life and to bird flocking, fish schooling and swarming theory as well as with evolutionary computation, mainly from theories presented in the genetic algorithms.

The origin of PSO lies back to the scientists who have conducted computer simulations based on movements of various organisms some of few are flocking birds and schooling fish. The motive for these simulations was to model human behavior which is not directly identical to these organisms. The important difference is its abstractness. It is seen that birds and fish adjust their movement to seek food and mates, avoid predators, and optimize environmental parameters such as temperature. But humans adjust not only their physical movement but also their cognitive or experiential variables as well. Also two human individuals can hold identical attitudes and beliefs without banging together, but it is not possible for two birds to occupy the same physical space without banging with each other. Therefore it can be inferred that human social behavior can be mapped for the concept change whereas the birds' and fish' to movement.

### 2.10.1. The Algorithm

The PSO algorithm is directly related with the bird flocking movement and the social harmony of the bird colony. The birds in real life are represented in the algorithm with the name "particles" and the algorithm is simulated by means of the motion of these particles.

In PSO, Each particle keeps track of its coordinates in the problem space which are associated with the best solution (fitness) it has achieved so far. (The fitness value is also stored.) This value is called *pbest*. Another "best" value that is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the neighbors of the particle. This location is

called *lbest*. When a particle takes all the population as its topological neighbors, the best value is a global best and is called *gbest*.

The particle swarm optimization concept consists of, at each time step, changing the velocity of (accelerating) each particle toward its *pbest* and *lbest* locations (local version of PSO). Acceleration is weighted by a random term, with separate random numbers being generated for acceleration toward *pbest* and *lbest* locations.

### **2.10.2. Advantages and Disadvantages of PSO**

One of the reasons that particle swarm optimization is attractive is that there are few parameters to adjust. One version, with slight variations, works well in a wide variety of applications. Particle swarm optimization has been used for approaches that can be used across a wide range of applications, as well as for specific applications focused on a specific requirement.

The PSO is designed to work with floating point represented problems. Although PSO also has a discrete counterpart, this algorithm generally does not give very good results. A research may be carried on the discrete PSO algorithm.

## **2.11. Ant Colony Optimization**

Ant algorithms [6], as their name suggests, were inspired by the observation of the real ant colonies. Ants are social insects, thus the behavior of a single ant is not determined by just itself, but to the survival of the whole ant colony. As a matter of fact, ants are highly structured colonies tailored to perform complex tasks together. One of the most important behavior of a colony is its foraging behavior, that how ants are able to find the shortest paths between their nests and the food sources.

Experiments have shown that ants deposit to the ground some small amount of a chemical substance called pheromone. Ants, which are rather blind insects, can smell this substance, and when choosing their way to the food source, take the pheromone density into account. It is shown

experimentally that once pheromone trail following behavior rises, the ants usually follow the shortest path to the food source. Thus by exploiting the pheromone trails left by the former ants passing through some path, the whole ant colony discovers the shortest path to the food source.

It is clear that the above process is some kind of a distributed optimization mechanism where a single ant contributes a small amount. Although it is possible for a single ant to build a solution, it is the combination of the whole colony that builds an efficient solution. This property is the emergent property of the ant colony and it is the indirect communication via pheromone laying, known as stigmergy.

### **2.11.1. Ant colony optimization method**

In ACO meta-heuristics, [7] the aim is to build good solutions to difficult discrete optimization problems by using artificial ants. Thus the artificial ant concept and cooperation between these intelligent agents plays a crucial role in problem solving.

Similarities between real and artificial ants can be summarized as follows:

- *Colony of cooperating agents:* As real ant colonies, artificial ants are organized in a population which is concurrently and globally interacting within to find a good solution.
- *Pheromone trail and stigmergy:* Just like real ants, artificial ants modify their environment via an artificial pheromone trail. Artificial pheromone trail is usually implemented by storing and changing some numerical information in a problem state an artificial ant visited. The stigmergy formed by this artificial trail is the only communication mechanism between the artificial ants. The prominent effect of this method is interacting with the local environment based on the past history of the whole ant colony.
- *Shortest path searching and local moves:* The aim of artificial ants is same with real ones: finding the shortest cost (path in the case of real ants) joining the initial configuration to the target configuration. In order to do that, artificial ants only

follow the adjacent states, they don't suddenly from one state to another in an instant.

- *Stochastic and myopic state transition policy:* Artificial ants, as real ones build solutions based on a probabilistic decision policy which is formed by the past history of the whole ant colony. A single ant cannot evaluate the global information because the only move that it can make is the one that it can sense locally (although this local sense is formed by a global process) and it can only change its environment locally.

For computing reasons, there are some certain differences between artificial and real ants. The important differences are as follows:

- Artificial ants are in a discrete world (a set of move states) whereas real ants are naturally in a continuous real world.
- Artificial ants have a small memory (internal state) to store past actions.
- Artificial ants deposit pheromone trail proportional to the quality of the solution found. Although this is valid for some real ants which deposit more pheromone if they come across with a richer food source.
- Artificial ants pheromone laying is designed suitably to the problem.
- As ACOs are computer algorithms, they are naturally improved with complex techniques such as lookahead, local optimization, backtracking. It is also possible to hybridize the ACOs with other search methods.

### **2.11.2. ACO Metaheuristics**

In ACO algorithms, a finite number of artificial ants collectively searches for a good solution to the optimization problem. It is possible for an ant to generate a solution or a component of it, starting from an initial state selected according to the problem. While building its own solution, an ant collects local information and uses this information to modify the representation of the problem. Ants do not use communicate with each other directly; it is the stigmergy paradigm that govern the information exchange among the ants.

The ant algorithm, similar to local search methods, begins with a neighborhood definition which is problem dependent. Each ant builds a solution by moving from one state to a neighbor states. The move selection is carried out by applying a stochastic local search method which is led by the memory of each ant (the ant internal state, or memory), and the pheromone trail which is present for that ant. The ant's internal state stores information about the ant's past history. It can be used to carry useful information to compute the value of the generated solution and contribution of each executed move. Ants can consequently build feasible solutions that can be performed in the local state.

The local, public information comprises both some problem-specific heuristic information, and the knowledge, coded in the pheromone trails, accumulated by all the ants from the beginning of the search process. The time-global pheromone knowledge built-up by the ants is a shared local long-term memory that influences the ants' decisions. Ants can release pheromone while building the solution or after a solution has been built, moving back to all the visited states or both. The more ants choose a specific move, consequently the move is rewarded with the increased pheromone trail and the more interesting it becomes for the next ants. In general, the amount of pheromone deposited is made proportional to the goodness of the solution an ant has built. Once an ant has accomplished its task, consisting of building a solution and depositing pheromone information, the ant dies and deleted from the system.

### **2.11.3. Applications of Ant Systems**

Ant colony optimization has been successfully applied to many combinatorial optimization and many other problems. The first application of ACOs was on the Traveling Salesman Problem as the mechanics of these problem was very suitable for the ACO meta-heuristic. ACOs were also applied on quadratic assignment, job-shop scheduling, vehicle routing, sequential ordering, graph coloring and some network routing problems. ACOs give competitive results with respect to other algorithms on generally small instances of such

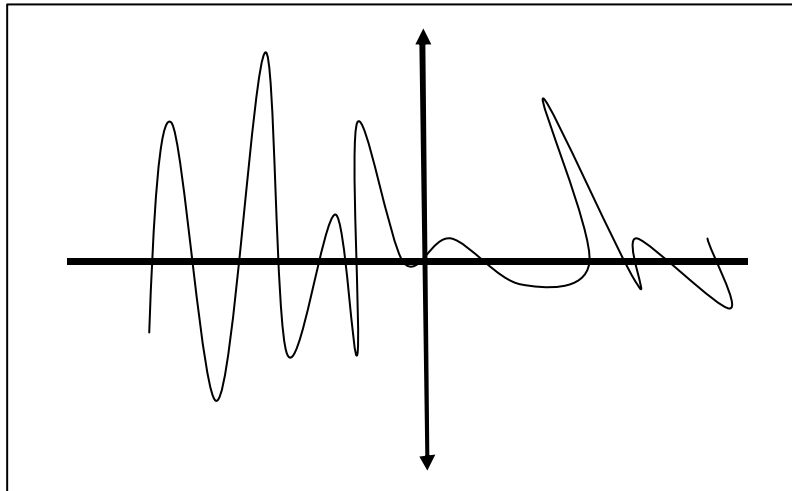
problems. However the time requirements of an ACO algorithm increases gradually making difficult to apply large scale problem instances.

## **2.12. Memetic Algorithms**

Local search and evolutionary algorithms have been successfully applied to many difficult real life problems. However both local search methods and evolutionary algorithms have their shortcomings. For example local search algorithms, as their name suggests, are highly successful in problem domains in which few hills and valleys occurs (thus optimized for ‘local search’). However when hills and valleys dominate the problem domain, the performance of local search may quickly deteriorate if necessary precautions are not taken to prevent cycles and stucking in local optimal values.

Evolutionary algorithms on the other hand are rather global search mechanisms. They may perform well under non uniform domains, however they are generally not very suitable for fine-grained local search.

Thus a natural approach is to combine both algorithms taking the best sides of those suitable to the problem. Most of the so called ‘memetic algorithms’ are thus the combination of an evolutionary algorithm with a local search method. Therefore memetic algorithms are often referred as hybrid algorithms. The benefit of using such a method can be described using the following figure.



**Figure 6 -A rather non uniform problem domain**

Consider we want to find the maximum (or alternatively minimum) point in this rather non uniform problem domain depicted in Figure 6. A general genetic algorithm model then will probably continue search within different hills and generally will be somewhat far away from the peak of that hill. Local search, alone, will however be traversing within a single peak whether that peak is optimal or not. Thus a good memetic algorithm is then preserving the diversity by means of a genetic search (finding prominent peaks) and optimizing the value with a local search.[27]

A clear problem in Memetic Algorithm community is the participating value of an evolutionary algorithm and a local search procedure within a memetic algorithm. Some researchers generally believe solution solving methods should depend on primarily crossover whereas others claim that local search methods should be given more importance to have a fine grained solution. In other words, it is unclear whether a hybrid algorithm should have many number of generations (i.e. a lot of global search at the population level) with a short local search time or it should have less number of generations with a much longer time for a local search (i.e., more time for local search at the individual level). Furthermore, it may be ideal if a memetic algorithm can adaptively switch from global to local search and vice versa according to different search spaces (landscapes).

## 2.13. Neural Networks

Neural networks specifically, attempts to mimic the fault-tolerance and capacity to learn of biological neural systems by modeling the low-level structure of the brain. The main branch of Artificial Intelligence research in the 1960s -1980s produced Expert Systems. These are based upon a high-level model of reasoning processes (specifically, the concept that our reasoning processes are built upon manipulation of symbols). It became rapidly apparent that these systems, although very useful in some domains, failed to capture certain key aspects of human intelligence. According to one line of speculation, this was due to their failure to mimic the underlying structure of the brain. In order to reproduce intelligence, it would be necessary to build systems with a similar architecture. Thus researchers concentrated into the structure of the human brain which we shortly describe as follows:

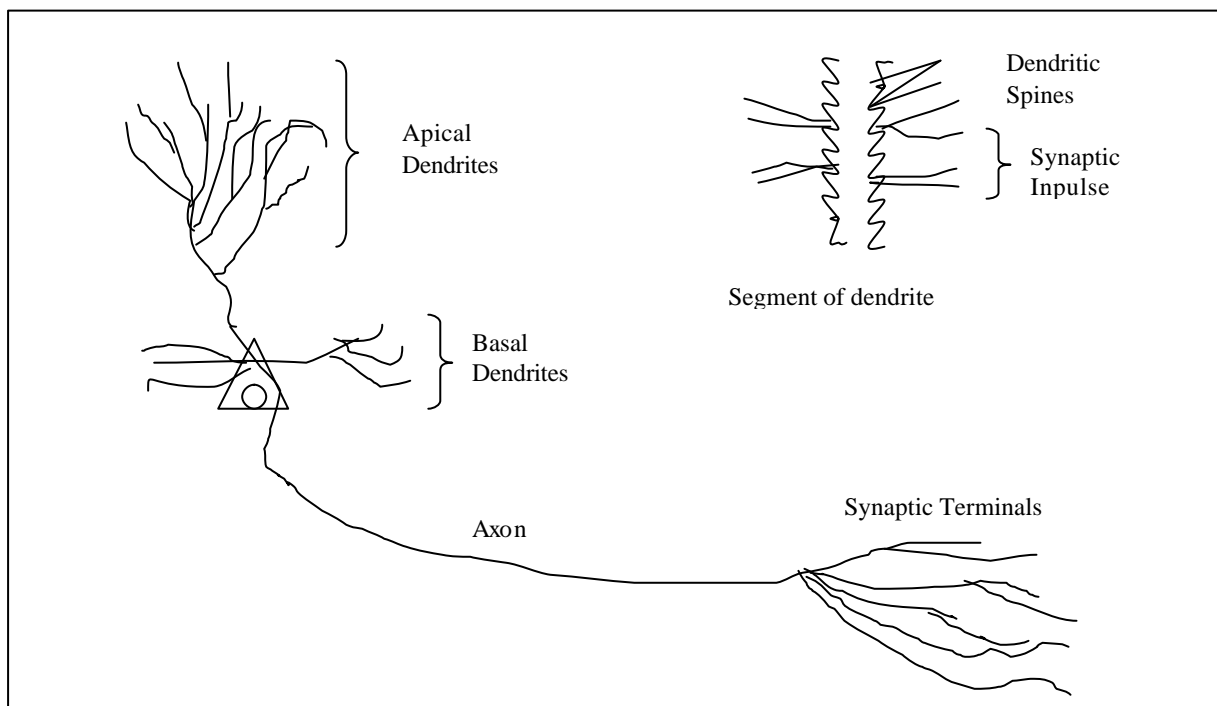
### 2.13.1. Human Brain

The brain is principally composed of a very large number (circa 10,000,000,000) of *neurons*, massively interconnected (with an average of several thousand interconnects per neuron, although this varies enormously). Each neuron is a specialized cell which can propagate an electrochemical signal. The neuron, as depicted in Figure 7, has a branching input structure (the dendrites), a cell body, and a branching output structure (the axon). The axons of one cell connect to the dendrites of another via a synapse. When a neuron is activated, it *fires* an electrochemical signal along the axon. This signal crosses the synapses to other neurons, which may in turn fire. A neuron fires only if the total signal received at the cell body from the dendrites exceeds a certain level (the firing threshold).

The strength of the signal received by a neuron (and therefore its chances of firing) critically depends on the efficacy of the synapses. Each synapse actually contains a gap, with neurotransmitter chemicals poised to transmit a signal across the gap. One of the most influential researchers into neurological systems (Donald Hebb) postulated that learning consisted principally in altering the "strength" of synaptic connections. Recent research in

cognitive science, in particular in the area of nonconscious information processing, have further demonstrated the enormous capacity of the human mind to infer ("learn") simple input-output covariations from extremely complex stimuli (e.g., see Lewicki, Hill, and Czyzewska, 1992).

Thus, from a very large number of extremely simple processing units (each performing a weighted sum of its inputs, and then firing a binary signal if the total input exceeds a certain level) the brain manages to perform extremely complex tasks. Of course, there is a great deal of complexity in the brain which has not been discussed here, but it is interesting that artificial neural networks can achieve some remarkable results using a model not much more complex than this.



**Figure 7 -A typical biological neuron**

### 2.13.2. Artificial neurons

A neuron [16] in the context of artificial neural networks (ANNs) is an information-processing unit that is fundamental to the operation of a neural network. Figure 8 shows the model of the neuron. The basic elements of the neuron model can be identified

- As a set of synapses or connecting links each of which is characterized by a weight or strength of its own. Specifically, a signal  $x_j$  at the input of synapse  $j$  connected to neuron  $k$  is multiplied by the synaptic weight  $w_{kj}$ . The weight  $w_{kj}$  is positive if the associated synapse is excitatory; it is negative if the synapse is inhibitory.
- An adder for summing the input signals, weighted by the respective synapses of the neuron; the operations described here constitute a linear combiner.
- An activation function for limiting the amplitude the output of a neuron. Typically, the normalized amplitude range of the output of a neuron is written as the closed unit interval  $[0,1]$  or alternatively  $[-1,1]$ .

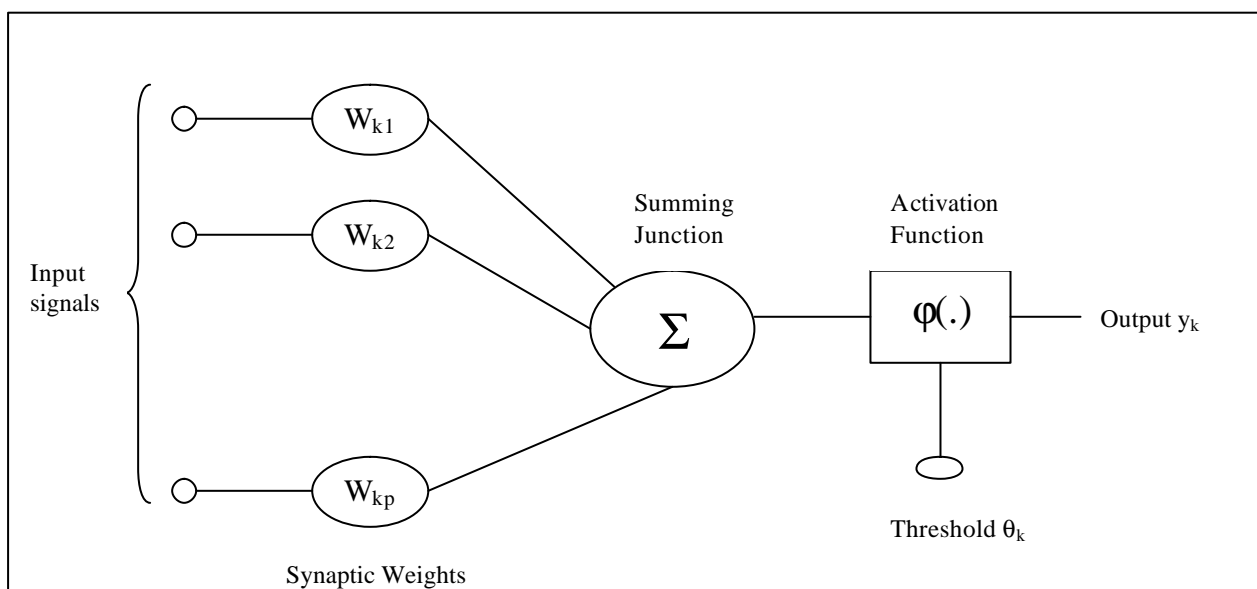


Figure 8 -Nonlinear model of a neuron

### 2.13.3. Types of activation functions:

- *Threshold functions*: In this model, the output of a neuron takes on the value of 1 if the total internal activity level of that neuron is nonnegative and 0 otherwise.
- *Piecewise – linear functions*: Step and ramp functions are special cases of piecewise linear functions that consist of some finite number of linear segments, and are thus differentiable almost everywhere with the second derivative = 0 wherever it exists.
- *Sigmoid functions*: The most popular activation function, sigmoid function is defined as strictly increasing function that exhibits smoothness and asymptotic properties. The advantage of these functions is that their smoothness makes it easy to perform mathematical manipulations and understand the behavior large networks whose nodes compute such functions.
- *Gaussian Functions*: Gaussian activation functions are based on the standard Gaussian distributions with variance  $\sigma$  and standard deviation  $\mu$ .

### 2.13.4. Neural Network Structures

#### 2.13.4.1. Perceptrons:

The perceptron is the simplest form of a neural network used for the classification of a special type of patterns said to be linearly separable (patterns that lie on opposite sides of a hyperplane). Basically, it consists of a single neuron with adjustable synaptic weights and thresholds.

Perceptrons with single layer structures has a single neuron, however such a perceptron is limited to performing pattern classification with only two classes. By expanding the output layer of the perceptron to include more than one neuron, we may correspondingly form classification with more than two classes. However the classes would have to be linearly separable for the perceptrons to work properly.

#### *2.12.5.2. Single Layer Feedforward Networks*

A layered neural network is a network of neurons organized in the form of layers. In the simplest form of a layered network, there is just an input layer of source nodes that projects onto an output layer of neurons, not vice versa. This explains the feedforwardness of these networks.

#### *2.12.5.3. Multilayer Feedforward Networks*

The second class of a feedforward neural network distinguishes itself by the presence of one or more hidden layers, whose computation nodes are correspondingly called hidden neurons or hidden units. The function of the hidden neurons is to intervene between the external input and the network output. By adding one or more hidden layers, the network is enabled to extract higher-order statistics, for the network acquires a global perspective despite its local connectivity by virtue of the extra set of synaptic connections and the extra dimension of neural interactions.

Typically the neurons in each layer of the network have as their inputs the output signals of the preceding layer only. The set of output signals of the neurons in the output layer of the network constitutes the overall response of the network to the activation pattern supplied by the source nodes in the input layer.

#### *2.12.5.4. Recurrent (Back Propagation) Networks*

A recurrent neural network distinguishes itself from a feedforward neural network in that it has at least one feedback loop. A recurrent network may consist of a single layer of neurons with each neuron feeding its output signal back to the inputs of all the neurons. The presence of feedback loops, has a direct impact on the learning capability of the network, and its performance. Moreover the feedback loops involve the use of particular branches composed of unit-delay elements which result in a nonlinear dynamical behavior by virtue of the

nonlinear nature of the neurons. Nonlinear dynamics plays a key role in the storage function of a recurrent network.[18]

#### *2.12.5.5. Lattice Structures:*

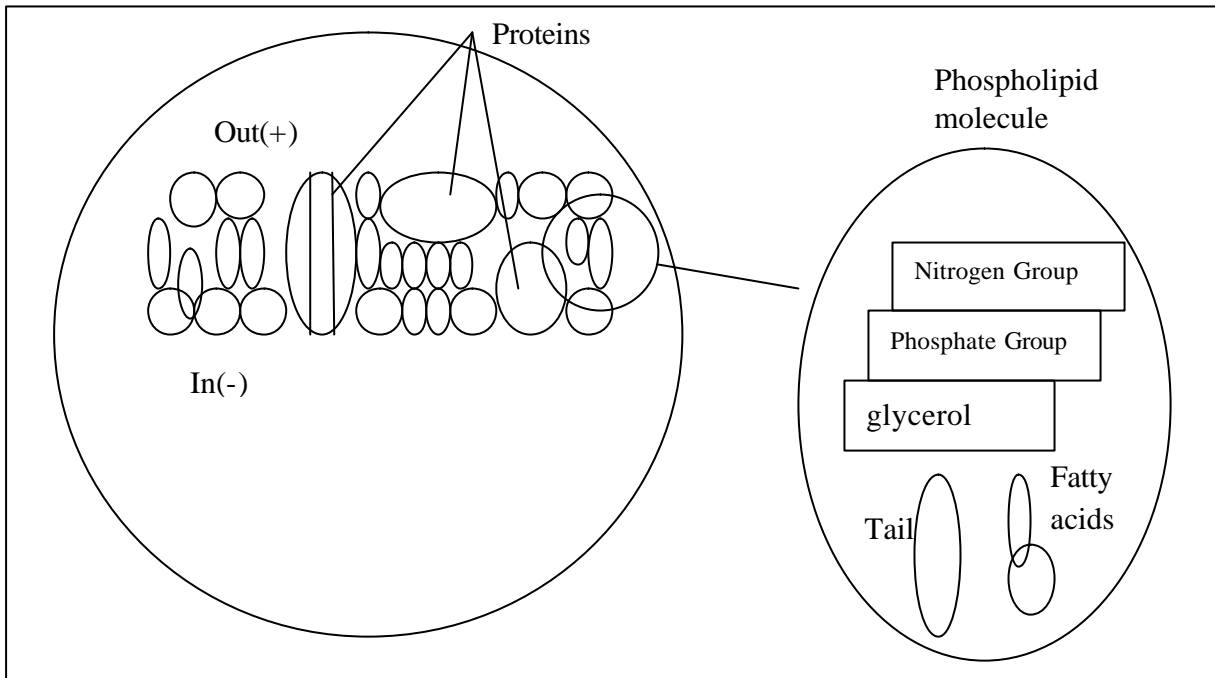
A lattice consists of a one-dimensional, two-dimensional, or higher-dimensional array of neurons with a corresponding set of source nodes that supply the input signals to the array; the dimension of the lattice refers to the number of the dimensions of the space in which the graph lies. A lattice network is actually a feedforward network with the output neurons arranged in rows and columns.

## **2.14. Membrane Computing (P Systems)**

A P system [26] is a computing model which abstracts from the way the alive cells process chemical compounds in their compartmental structure. In short, in the regions defined by a membrane structure there are objects which evolve according to given rules. The objects can be described by symbols or by strings of symbols (in the former case their multiplicity matters, that is, one can work with multisets of objects placed in the regions of the membrane structure; in the second case one can work with languages of strings or, again, with multisets of strings). By using the rules in a nondeterministic, maximally parallel manner, one gets transitions between the system configurations. A sequence of transitions is a computation. With a halting computation we can associate a result, in the form of the objects present in a given membrane in the halting configuration, or expelled from the system during the computation.

The basic function of biological membranes is to define compartments and to relate these to their environment, including neighboring ones. For example, the plasma membrane keeps certain substances within the cell, while other substances, (for example harmful toxic molecules) stay out of the cell. Moreover, membranes allow certain molecules to pass through cell pores, waste products to leave, and certain nutrients to enter. Membranes also form a communication structure, allowing signals to be received and / or transmitted by the enclosed

area with its own set of molecules and reactions, with the transport of molecules and the communication through membranes, is the paradigm underlying membrane systems.



**Figure 9 -A Biological Plasma Membrane**

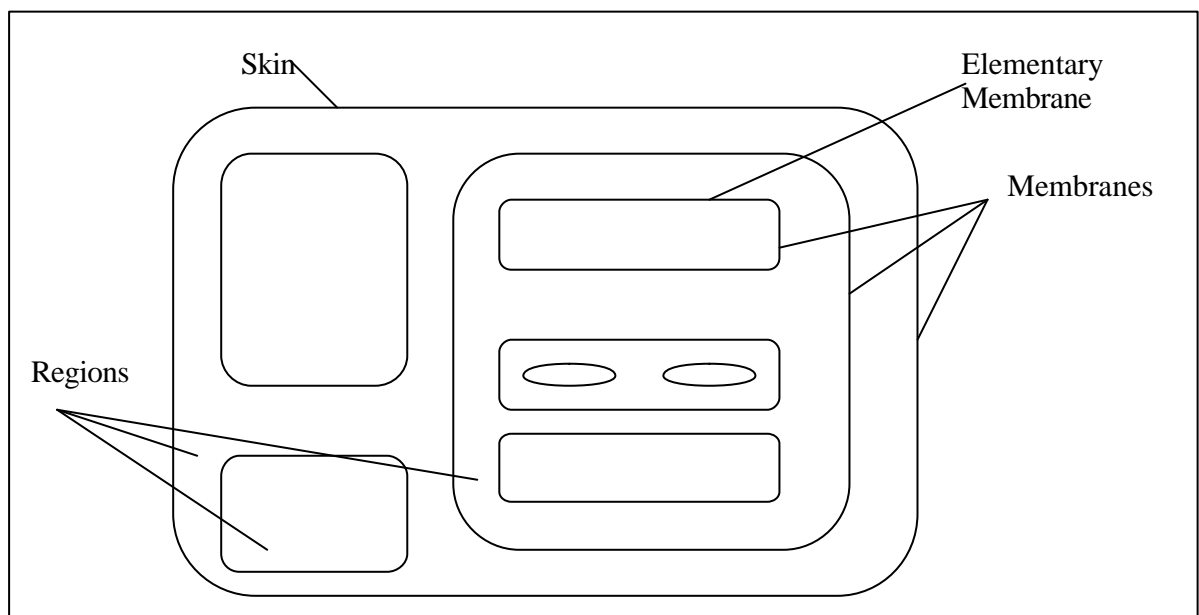
### **2.14.1. Biological Foundation of Cell Membranes**

A cell has a complex structure, with several compartments delimited inside the main membrane by several inner membranes: the nucleus, the Golgi apparatus, various vesicles, etc. In principle, all of these membranes fulfill the same main roles: they are separators and filters. The currently accepted model of the membrane structure is the so-called fluid-mosaic model proposed in 1972 by S. Singer and G. Nicolson. According to this model, a membrane is a phospholipid bilayer in which protein molecules are totally or partially embedded. The membrane structure is only partially permeable. For instance, small noncharged molecules cross the membrane freely whereas large molecules can pass only if they are assisted. The transmembrane transfer of molecules can take place in a passive manner (diffusion towards the region of lower concentration) and in an active way (water passing through protein channels in the membranes).

Other important functions of membrane proteins are the catalytic activity (reaction occurring by the help of enzymes), recognition and binding activities (certain proteins recognize certain molecules or even catch them and keep them bound to the membrane).

### 2.14.2. Membranes in Membrane Computing (P Systems)

The membrane structure of a P system, as shown in Figure 10, is a hierarchical arrangement of membranes, embedded in a skin membrane, the one which discloses the system from the environment. A membrane without any membrane inside is called elementary. Each membrane defines a region of its own. For an elementary membrane this is clearly the space enclosed by it, while the region for a other membranes is the space in and between the membrane and the membranes directly included in it.



**Figure 10 –An Artificial Membrane Structure**

Each region contains a multiset of objects and a set evolution rules. The objects are represented with the symbols for a given alphabet. An evolution rule from region  $r$  is of the form  $ca \rightarrow cb_{inj}d_{out}d_{here}$  which means that a copy of the object  $a$ , in the presence of a copy of the catalyst  $c$  (an object which assists the evolution of other objects – a metaphor to the

chemical catalysts), is replaced by a copy of the object  $b$  and two copies of the object  $d$ . The copy of  $b$  has to “immediately” enter the membrane of region  $r$  labeled by  $h$  ( to enter region  $j$ ). a copy of object  $d$  is sent out through the membrane of region  $r$ , and a copy of  $d$  remains in region  $r$ . It should be noted that the considered evolution rule can be applied in the region  $r$  only if the membrane  $j$  is included in this region.

Membrane systems are synchronous, in the sense that a global clock is assumed, the same clock holds for all regions of the system. In each time unit a transformation of a configuration of the system takes place by applying rules in each region, in a nondeterministic and maximally parallel manner. This means that object which evolve and rules which govern the evolution are chosen in a nondeterministic and exhaustive way; once a choice is made, no rule can be applied in the same evolution step.

It is seen that a single step transformation of a configuration of the system as a “macro-step” consisting of several “micro-steps” performed after each other. Considering a single region  $r$  of the system, first the objects from  $r$  to rules from  $r$  are assigned nondeterministically choosing rules and objects until no further assignment is possible. Then all of these assigned objects are removed from the current multiset of objects in  $r$ , and all objects specified by the right hand sides of the chosen rules are added to this multiset together with their transfer commands,  $in_j$ ,  $out$ ,  $here$ . Now all transfers indicated by commands  $in_j$  and  $out$  are executed and copies of objects with the transfer command  $here$  remain in region  $r$ . Finally the transfer commands (subscripts) are removed, and a “macro-step” is completed for  $r$ . Since all regions are simultaneously and synchronously processed, the global macro-step is completed.

With the above process, one can simulate the transitions between the configurations of the system. A sequence of transitions is called a computation. A configuration is halting, if no rule is applicable in any region (nothing can happen anymore). A computation is halting, if it reached a halting configuration. The result of a halting computation is the number of objects sent through the skin membrane to the environment during the computation.

### 3. Graph Coloring Problem

Graph Coloring Problem (GCP) [22] is one of the most well known and studied Combinatorial Optimization Problems (COP). An informal definition can be summarized as one wants to color the vertices of an undirected graph with a minimum number of colors given one constraint that two adjacent vertices should be given different colors. A more formal definition is as follows:

Let  $G = (V, E)$  be an undirected graph with a vertex set  $V$  and an edge set  $E$ . An independent set is defined as a set in which no adjacent vertices are located. A  $k$ -coloring problem is then a partition of set  $V$  into  $k$  independent sets. An optimal coloring is consequently a  $k$ -coloring with the minimum possible  $k$  ( $k$  is in such case defined as the chromatic number  $\chi(G)$  of graph  $G$ ). The Graph Coloring Problem is thus to find the optimal coloring of a given graph  $G$ .

GCP is, although at first sight seems a simple problem, is proved to be in NP-Complete class. Thus no polynomial time algorithm can be proposed to solve all instances of GCP. Approximating the problem is also exceptionally difficult, no proposed polynomial algorithm is able to color any graph using number of colors less than  $2 * \chi(G)$ . Johnson et. al. observed that even for relatively small graphs (around 90 vertices) with edge densities around 0.5, no exact algorithm is known.

#### 3.1. Heuristic Methods for Graph Coloring

Many heuristics have been proposed the graph coloring problem. The most important of them are as follows:

*Greedy Constructive Methods:* The principle of a greedy constructive method is to color successively the vertices of the graph. At each step, a new vertex and a color for it are chosen according to some particular criteria. Two well known greedy heuristics are as DSATUR and RLF (Recursive Largest First) Algorithms.

*Genetic Algorithms:* Pure genetic algorithms [5] were applied into graph coloring only by Davis where the GA gave very poor results. It is thought that pure GAs are not competitive for GCP.

*Local Search Based Algorithms:* Many LS algorithms most of which are based on simulated annealing and tabu search are proposed to solve graph coloring. Local Search algorithms generally perform competitive in GCP.[8][30][18]

*Hybrid Algorithms:* Memetic algorithms incorporating local search and crossovers were proposed which gives very good results in the standard test instances.[4][12][11]

*Ant Colony Optimization:* Ant algorithms were proposed to solve graph coloring which gives competitive results for small instances.[7][3]

*Successive building of color classes:* This strategy builds successive different color classes by identifying each time a maximal independent color set and removing its vertices from the graph. [21]

### **3.2. Hybrid Evolutionary Tabu Search (HET) for Graph Coloring**

In order to solve Graph Coloring Algorithm Hybrid Genetic Tabu Search Algorithm has been devised. Although Genetic Search and Tabu Search are separately implemented and thus able to work alone, the main intention is to test under a hybrid algorithm. Thus many of the algorithm operators which are listed in their respective sections are optimized for a hybrid algorithm.

The hybrid algorithm can be roughly summarized as follows.

*INITIALIZE* Population

repeat

*SELECT* two individuals from the population for crossover

*CROSSOVER* two individuals to give one or two offspring

    Improve the offsprings by using *TABU SEARCH*

*REPLACE* the parents with the improved offspring

Until a valid coloring with k colors is obtained.

### 3.2.1. Representation

Since we are trying to solve k-coloring problem, each configuration is represented with k color classes  $C_1, 2, 3, \dots, k-1, k$ . Each color class contains vertices  $V_i$  which will bear the color of that class. The cost of each color class will then become the number of edges between conflicting vertices.

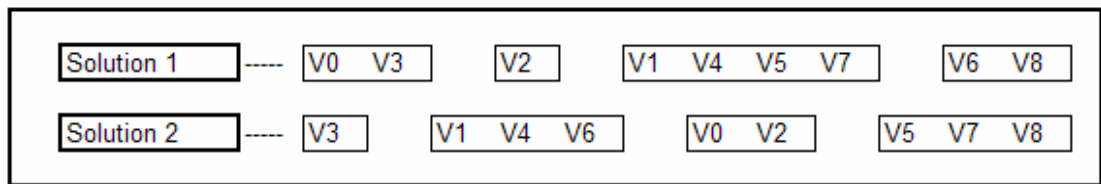


Figure 11 -Example representation of two solution constitute d of different independent sets

### 3.2.2. Initialization

In order to create a population with n individuals, a random initialization procedure which randomly assigns the vertices of the graph to the color classes, has been applied. Although this process slows down the convergence rate in the beginning of the algorithm, creates a rather balanced population which is sufficiently diverse although it is not so optimal.

Another approach tested is a greedy assignment of vertices into color sets based on the conflict counts of each vertex. A vertex is assigned to the color class in which it conflicts with the least number of vertices in terms of color. We have observed that for some easy graph instances we have used, this method gives a direct solution in the initialization. However, for more difficult instances, it seems more reasonable to begin the algorithm with somewhat worse individuals than the seemingly best individuals at first iterations. Thus we have applied a probabilistic method and assign some vertices randomly. The percentage of those randomly assigned vertices is another algorithm parameter.

### 3.2.3. Selection

In order to apply crossover, it is necessary to select two individuals beforehand. For the tests Roulette Wheel Selection and Rank Based Selection have been applied. Rank based selection is preferable because for a hybrid algorithm, we need a diverse population and roulette wheel selection may suffer from a quick premature convergence. In order to give rank to the individuals, we have sorted the population in ascending order by using an insertion sort, and giving a rank equal to (population size – index of individual after sort). Thus the selection pressure on the (n – j)th individual is as shown in Equation 2 where n is the population size.

$$P_{n-j} = \frac{j}{\sum_{k=1}^n k}$$

Equation 2 - Selection Pressure on an individual in Rank Based Selection

We have also tested under Roulette Wheel Selection, both under a random initialization and a greedy assignment of vertices. When the population is initialized with a randomly generated population the algorithm quickly converges to a local optimum prematurely. This results because of the super individuals created by the local search procedure. When a greedy initialization is used, the effects of premature convergence diminishes and the population begins to resemble to the one formed with Rank Based Selection. It is because greedy initialization creates rather uniformly distributed individuals in terms of cost and the cost variance is rather low.

### 3.2.4. Crossover

One of the most important aspects of a genetic algorithm is the crossover mechanism. It is important that a good crossover is the one which transmits the good characteristics of a configuration into its children. Thus a good crossover has to incorporate specific domain knowledge. More precisely, designing crossovers requires to identify properties that is related

with the problem and then to develop a recombination mechanism that transmits these properties from parents to offspring.

Because of the reasons explained above, traditional blind genetic operators cannot lead to an efficient solution as they do not take the specifics of the problem into account. It is also seen that when a partitioning (grouping) approach is applied into graph coloring, we need genetic operators tailored to work with groups rather than a binary representation. Then a genetic operator should transmit the attributes of a good color class into offspring. The following crossover methods have been proposed for this purpose

**Greedy Partition Crossover (GPX):** This crossover method is proposed by Gallinier and Hao [12] based upon Falkenauer's work on Grouping Genetic Algorithms [10]. In this crossover method, the offspring is created from two parents as follows by finding the largest independent from one parent individual and assigning those set as an element of the offspring. By alternating the donor individual between two parents and deleting the vertices that are assigned to the offspring from both of the parents, an offspring is created. A pseudo code of the algorithm can be summarized as follows:

*N is number of color sets in an individual*

*Father and Mother are two parent individuals*

*For l = 1 to N*

*If l is odd, then donor is father, else donor is mother.*

*Choose the largest color Set in the donor.*

*Assign the elements of this largest set to the lth colorSet of the offspring.*

*Remove the assigned vertices from both father and mother.*

*End*

*If any vertices remain unassigned assign them to a random color class of the offspring.*

*If some color classes of the offspring remain empty, split larger classes of the offspring, and Assign some vertices of one splitted set into empty sets.*

This aim of this crossover method is to preserve the large independent sets into the offspring. However this crossover is not suitable for a pure genetic algorithm and the generated individuals should be improved by a local search procedure. Another shortcoming of this crossover is that it may give an unbalanced color sets in terms of length, which means the variance of the color set length may be somewhat large.

**Weighted Color Sets Crossover:** It is clearly seen that Greedy Partition Crossover although preserves the large independent sets, it does not care whether that set is good in terms of edge conflict. An improvement of GPX can be instead of just assigning the larger sets into the offspring, one may want to rank the color sets in terms of set penalty (number of conflicting edges) and the length of the set. The ranking method that is used employed in this proposed algorithm

$$R_i = a \text{length}_i - \beta \text{setCost}_i$$

**Equation 3 - Weighting function in the Weighted Sets Crossover**

where  $\text{length}_i$  is the number of vertices currently present in the corresponding set and  $\text{setCost}_i$  is the number of conflicting edges between adjacent vertices in the respective set.  $a$  and  $\beta$  are the coefficients to determine the balance between the length and the set cost. For a generic purpose these coefficients can be set to 1 to give equal privilege to both these properties.

Another improvement in the crossover method is to improve the random assignment if a vertex can not be assigned a set. In our observations it is seen that about 10-30 percent of the vertices are assigned randomly. Although this may add up additional probabilistic behavior which may enlarge the search space and add some variety into the gene pool, this amount of randomness is rather large. Therefore A balancing factor has been applied in the assignment of these vertices. Instead of a random choice, the vertex is given the best color (i.e. the color in which the assigned vertex results the least amount of edge conflicts). However the number of sets that these unassigned vertices will be placed, have been narrowed in order to preserve the good sets created by the initial part of the

crossover. The choice is made between the lastly created sets in the initial portion of the crossover process. This neighborhood defining the sets that the individuals can be assigned is another search parameter.

### **3.2.5. Mutation**

Mutation as in many other genetic algorithms serves as a background operation whose main purpose is to add some diversity into the population. The mutation method that is used in this algorithm is based on shuffling. Two color sets from the selected individual have been randomly selected, and then a permutation from the elements of those sets have been created. This process ends by assigning the elements (vertices) back to the two sets. The mutation's main purpose is to add some variety into the gene pool when the diversity of the population decreases. Therefore we have applied rather a large mutation rate compared to other algorithms. Another reason for that is we improve the individual formed after the mutation with a local search procedure, thus possible disruptions resulting from the mutation is repaired afterwards.

### **3.2.6. Local Search Procedure**

Local Search Method that we have used to improve the offspring created by the crossover is an improved version of Tabu Search proposed by Hertz and de Werra. In this algorithm a simple neighborhood structure is obtained by moving a single vertex  $v$  from a color set into another one. To speed up the neighborhood generation and cost calculation, conflict arrays which hold the number of conflicting edges of each vertex are used. This not only eases the cost calculation but also prohibits the moves of the vertices which are not conflicting in their current class. The size of the neighborhood is, unlike other algorithms, dynamic and it increases with the number of iterations. The aim of this approach is that in the beginning of the local search process, large gains can be gained without checking the entire neighborhood. However when the algorithm converges to a solution (whether optimal or not), gains becomes

minimal and thus the search should cover larger search space. The size of the neighborhood determined as a percentage is

$$\%Size = \left( 1 - \frac{NoofConflictingEdges}{NoofEdges} \right)^n * 100\%$$

**Equation 4 - Neighborhood Percentage Calculation**

where NoOfEdges is the number of edges in the graph to be colored, NoOfConflictingEdges is the number of conflicting edges in the individual. N is nonlinearity factor of the search space. The aim of this subsampling is to narrow the search space if we are very far away from the desired solution. For  $n = 1$ , this will become a linear narrowing, and for more greedy narrowing higher values of n can be used.

Once a best move within a neighborhood has been found, the move is performed. When such a move is performed, the inverse of that move is recorded into the tabu list to prevent from returning to a previously visited configuration and thus preventing cycles. How long a tabu move will remain forbidden (tabu tenure) is another algorithm parameter. A static tabu tenure is setting it around the number of vertices in the graph. Alternatively tabu tenure is set dynamically which depends on a probabilistic factor and the number of conflicting edges in the configuration. While a tabu move's tabu tenure does not expire, it is not possible to perform that move.

### **3.2.7. Replacement:**

After the offspring created in crossover operation and improved by the local search method, it is time to enter this offspring into the population. One possibility is to replace the worst of the parents with the child without taking the cost of the child and parents into account. However in the later sections of the search process, it is not so easy to find improved offspring which are better than their parents, therefore we have replaced the parent with the child if and only if its individual cost (how far away from the target solution) is higher than the child. Although

this procedure may result in a decrease in diversity level (and may result with a premature convergence), it is preferred because it rather increases the convergence rate.

### **3.3. Programming Environment C++**

C++ is designed in the early 1980's by Bjarne Stroustrup of the Bell Labs. The aim of creating C++ was to create a “general purpose programming language designed to make programming more enjoyable for the serious programmer.” (Stroustrup). C++ supports the procedural programming which it borrowed from his precursor C. The other main properties of C++ are as follows:

- Stronger type checking
- Supports data abstraction
- Support object-oriented programming
- Supports generic programming
- Portable (by using preprocessor directives)
- Supports polymorphism by run-time (dynamic) bindings of function calls
- Supports all types of inheritance (public, protected, private; singular, multi) by reusing interfaces and implementations
- Can link with compiled C code (and libraries)
- Adds generic code (template class) support
- Adds exception handling
- Supports large-scale programming by means of separate compilation, namespaces, and libraries (archives)

C++ is later improved by many compiler vendors by means of namespaces, exception handling, run-time type identification (RTTI), improved templates (etc). C++ is later standardized by ANSI, DIN, BSI and ISO.

Today C++ is used to maximize execution speed, to support software reuse with separation of interface and implementation, for data abstraction and dynamic binding, portability and backward compatibility with C. Because of its huge advantages C++ is one of the most important de facto computer languages in the programming arena.

### 3.4.Data Structures used in the Algorithm

The following structures constitutes the data structure used by our algorithm. These give more information how the operators discussed above works on the data.

#### 3.4.1. Set

Set is the most basic data structure in our algorithm. A set represents an independent color set with operations defined on it. A set is represented with an array containing the ids of each vertex assigned to some color and an additional conflict array to store the number of conflicts of each vertex with every other one. The representation of a set is as in Figure 12.

Basic operations on a set are the adding and removing vertices from it, calculating the penalty of this independent set. It is obvious that adding and removing vertices also requires the calculation of the penalty of the set, thus lengthy operations on the conflict array. For performance issues, non updating set operations have been optionally defined to avoid these operations.

SetElements	0	1	3	7	3	0
Conflict Array	1	0	1	0	1	1
Max Set Size	6					
Current Set Size	5					
Set Cost	2					

**Figure 12 -An independent color set.**

### 3.4.2. Individual

Individual is the basic operational unit in a search procedure. In its simplest form it is an array of independent sets with inter-set operations defined in it. The tabu search algorithm directly uses the individual for the search mechanism. Thus the following tabu moves are all defined on the individual.

*Simple move:* It is a simple move which removes an element from the given set, and adds it to the recipient set. It is the most basic move implemented in many Tabu Search algorithms and it constitutes about the all of the moves. The individual is shown as in Figure 13.

The following moves are implemented however are not tested. These moves are a series of simple moves combined together.

*Swap move:* Swap move swaps two elements between two elements.

*Ejection moves:* Ejection moves are a series of moves followed by each other.

The aim of swap and ejection moves is to provide a simple look-ahead mechanism to direct the search into promising areas.

Apart from moves, individual is responsible for instantiating color sets, finding the largest set, and accessing any specific independent.

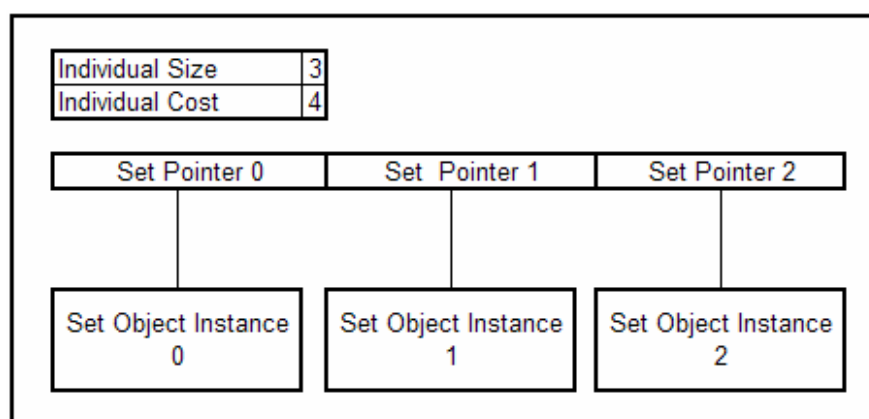


Figure 13 -The Structure of an Individual Object

### 3.4.3. Population

Population is the main structure in any evolutionary based algorithm. Population mainly consists of individuals and operations on them. Main operations on the population on the population are sorting and searching individuals based on the penalty function of each corresponding function.

The sorting method that we have used in our algorithm is a simple insertion sort with worst time complexity  $O(n^2)$  and best time complexity is  $O(n)$ . Since the population is always sorted after each local search step, the population is generally almost sorted so  $O(n)$  time complexity will occur.

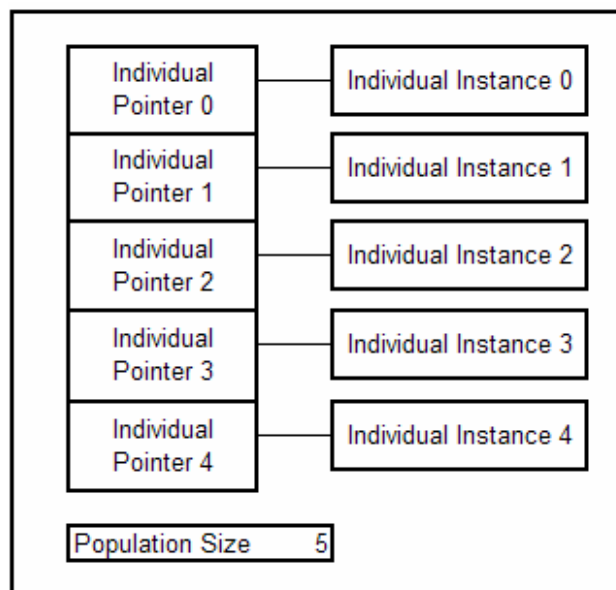


Figure 14 - The Structure of a population

#### 3.4.4. Implementation Objects

- Class Graph: A constraint satisfaction graph which holds the vertices and the edges of the graph. Each constraint is given with an edge between two conflicting vertices.
- Class Set: Set class defines the operations of an independent set.
- Class Individual: Individual defines the operations that can be performed within a series of set.
- Class Population: This class encapsulates all of the Individuals and Sets present in the algorithm.
- Class Algorithm: The algorithm class serves as a basis for all of the heuristics algorithms that will be derived from it. Algorithm class contains routines responsible for time management, initialization of a graph structure from a graph file.
- Class Descent: Descent object encapsulates a simple steepest descent algorithm without any checking on the circular searches, local optimum sticking etc.
- Class TabuSearch: Tabu Search related algorithms are present in this object. It encapsulates the operations for finding the best move in one iteration. This is carried out by different move operations presented in the individual operations.
- Class TabuList: Tabu list structure is defined in this object. Operations such as setting an attribute as tabu or non tabu move, setting tabu tenures for different tabu moves are all defined here.
- Class GeneticAlgorithm: The Genetic Algorithm operators are all defined in here. This includes the crossover and mutation mechanism, population initialization, etc.
- Class GeneticTabu: This encapsulates the main algorithm presented in this thesis. This object is inherited from the Genetic Algorithm and the Tabu Search.
- Class GeneralParameters, TabuParameters, GeneticParameter: These structures are for setting the parameters used in the algorithm with ease. More information is explained in the parameters section.

### Algorithm Hierarchy

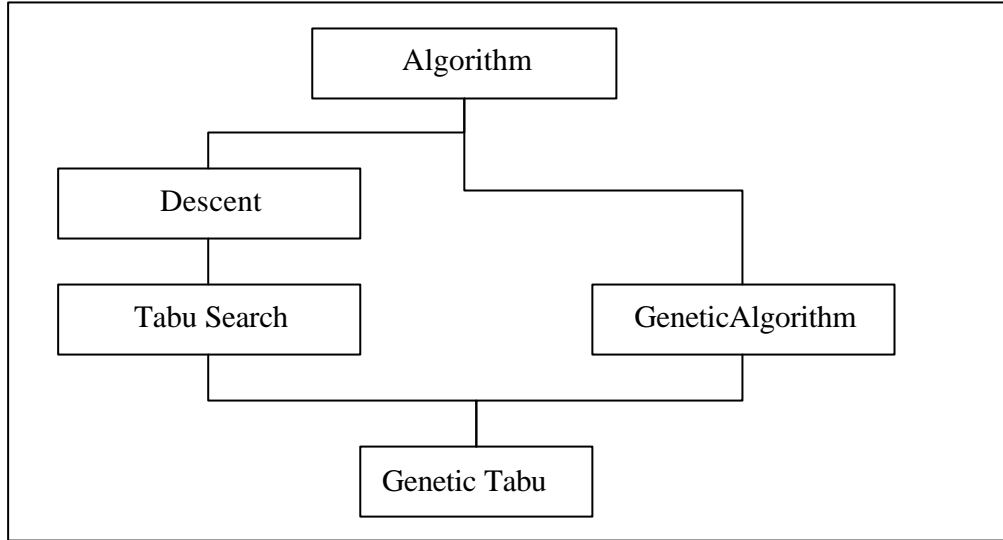


Figure 15 -Algorithm Objects Hierarchy

## 3.5. Experiments

Various tests have been applied to observe the behavior of the algorithm. First the graph format used in the algorithm is given, then test parameters have been explained and then test results and comments on them are given.

### 3.5.1. DIMACS Graph Format.

In order to standardize the file format used to solve graph coloring problems, a graph format is proposed in the DIMACS (Discrete Mathematics and Computer Science Department in Rutgers State University) Challenge organized in 1992 for various graph coloring solving heuristics. There are two types of DIMACS format, ASCII and binary. ASCII format is a simple human readable text file with the meaning of the lines are represented with initials. A sample DIMACS ASCII format is as in Figure 16.

```
c FILE: DSJC125.1
c
c SOURCE: David Johnson (dsj@research.att.com)
c
c DESCRIPTION: Random graph used in the paper
c               "Optimization by Simulated Annealing: An
c               Experimental Evaluation; Part II, Graph
c               Coloring and Number Partitioning" by
c               David S. Johnson, Cecilia R. Aragon,
c               Lyle A. McGeoch and Catherine Schevon
c               Operations Research, 39, 378-406 (1991)
c
p edge 125 736
e 5 1
e 6 2
e 8 4
e 9 4
e 9 6
e 11 2
e 13 5
e 14 7
e 14 9
e 14 13
```

**Figure 16 -A DIMACS ASCII graph instance**

As seen from Figure 16, lines represented with a c are comment lines, whereas p and edge lines represents the number of vertices and number of edges present in the graph. The edges are shown within e lines where each edge is represented with the starting and ending vertices of it.

Usually binary format generally takes huge spaces, therefore a fast and storage friendly binary representation is formed. In this binary format only the number of vertices and edges and the edge vertices are present. There is no need for comment lines as this is not a human readable format.

### 3.5.2. Experimental Data

We have used the instances presented in the second DIMACS challenge. Our main concentration was in the Leighton, Mile, Myciel, book graphs, and DJCS graphs.

*Leighton* instances are generated by a procedure proposed by Leighton which constructs graphs with known chromatic number. Leighton instances are all 450 vertex graphs with chromatic numbers ranging from 5 to 25 colors, with average vertex degrees ranging from 11 to 77.

*Myciel* instances are small instances with known optima. These instances are used in the debugging process in our algorithm because of their simplicity.

*Mile graphs* are similar to geometric graphs in that vertices are placed in space and an edge exists between two vertices if the vertices are sufficiently close. These graphs are not random, since the vertices represent a set of United States cities and the distance between them are given by the road mileage from 1947. These graphs are created by D. Knuth and known optimal number is known.

*Book graphs* are created by first assigning a literature character to a vertex. Two vertices are connected by an edge if the corresponding characters encounter each other in the book. Knuth also has created the corresponding graphs for five classic works: Tolstoy's Anna Karenina (anna), Dickens' David Copperfield (david), Homer's Iliad (homer), Twain's Huckleberry Finn (huck), and Hugo's Les Miserables (jean).

Finally the *DSJC* graphs are random graphs created by Johnson et. al.[17] for their studies in simulated annealing to solve graph coloring problems. These graphs range from 125 to 1000 vertices with 0.1, 0.5, and 0.9 edge densities. The optimal colorings of these graphs are not known and obtaining smaller chromatic values in these graphs is exceptionally difficult. Therefore these instances are the main test target by many researchers.

It is generally observed real life instances are somewhat easier to color than random graphs. Also it is reported that the most difficult instances generally have edge densities about 0.7 (dense graphs). Therefore we have concentrated our testes on the seemingly more difficult instances of Leighton and DSJC graphs.

### 3.5.3. Test Parameters

The algorithm that is proposed contains a large number of parameters which directly affect the performance of the algorithm. The parameters can be summarized under three groups

- General Parameters: Non algorithm related parameters.
- Tabu Parameters: Parameters related with Tabu Search.
- Genetic Parameters: Genetic Algorithm related parameters

#### *General Parameters:*

- Number of runs: This defines how many times the algorithm will work. The more the value of this parameter, the more clear the behavior of the algorithm will be. For small instances (ranging from 0-125 vertices) this is set to 50, for medium instances (125-250) this is set to 25, for large instances ranging from 250-500, it is 10, and for very huge instances, the parameter is set to 5. Note that for some algorithms, we have used less or more number of runs than the presented numbers.
- Maximum number of iterations: This defines the maximum of iterations that the algorithm will work. This is actually related with the size of the graph and its difficulty.
- Number of desired colors: This important parameter sets the number of colors to color the graph. The ambition is to find the minimal coloring so subsequently lower values are given to the algorithm as long as a valid solution is found. This parameter is also the size of each individual.

#### *Genetic Parameters:*

- Population Size: This gives the number of individuals in the population. For easier instances this is set to low values around 5, whereas for more difficult values higher values are tested about 10. It should be noted using very large population sizes greatly increases the computation time.
- Crossover type: This sets which crossover method will be used in the algorithm (greedy partition crossover and weighted sets crossover).
- Selection type: There are two types of selection operators implemented. Ranked based selection and roulette wheel selection.
- Crossover rate: This parameter sets the frequency of the crossover rate. Values near 1 were given to this parameter to enforce crossover.
- Mutation rate: Sets the frequency of the mutation rate. We generally enforce the mutation by setting it around 1 just like crossover.

#### *Tabu Parameters:*

- Static Tabu Tenure Length: This defines the number of iterations a tabu move will remain as forbidden. This is generally set around the number of vertices in the graph.
- Dynamic Tabu Tenure: When this parameter is set, the tabu tenure is set according to the penalty function of an individual.
- Number of tabu iterations: This defines how long a tabu search procedure will last. For more difficult instances, this value is set higher values.
- Neighborhood Size: The neighborhood tabu search will operate is defined with a percentage. In the beginning of the algorithm this percentage begins with low values and later increases with the neighborhood increment percentage.
- Neighborhood Increment percentage: The percentage that the neighborhood will expand after a number of iterations.

### 3.5.4. Test Results

The following results were attained with little a priori knowledge about the graph instance. For a basic setup, all of the miles graph are solved by setting tabu length to 100 iterations after each crossover and mutation operation. The following results depicted in Table 1, are attained after 20 runs. All of the runs returned successful results.

Graph Name	Vertex No	Edge No	Chromatic Number	Time (sec)	Std Dev of Time	Genetic Iterations	Std. Dev of Gen.	Tabu Iterations
miles250	128	774	8	0.06	0.15	1	0	100
miles500	128	2340	20	0.09	0.02	2.3	0.001	230
miles750	128	4226	32	0.13	0.077	4.45	0.005	445
miles1000	128	6432	42	0.68	0.36	38.2	0.004	3820
miles1500	128	10396	73	0.43	0.33	13.45	0.003	1345

Table 1 - Miles1000 Graph with Tabu Length 100 iterations

The book graphs are solved with the same parameter. It should be noted that even for small values of tabu iterations, these graphs are easily solved within one second.

Graph Name	Vertex No	Edge No	Chromatic Number	Time (sec)	Std Dev of Time	Genetic Iterations	Std. Dev of Gen.	Tabu Iterations
anna	138	986	11	0.06	0.01	1	0	100
david	87	812	11	0.05	0.01	1	0	100
homer	561	3258	13	1.46	0.5	1.35	0	135
huck	74	602	11	0.52	0.01	1	0	100

Table 2 - Miles1500 Graph with Tabu Length 100 iterations.

As the Chromatic number of DSJC graphs are not known, the smallest attainable color numbers within a reasonable time are tested. For Leighton graphs le450\_25a and le450\_25b, the exact chromatic number is known and it is found within the algorithm. For DSJC graphs tabu iterations are set to 100 iterations after crossover, mutation. For Leighton graphs they are set to 250 iterations. The results are shown in Table 3.

Graph Name	Vertex No	Edge No	Colors Attained	Time (sec)	Std Dev of Time	Genetic Iterations	Std. Dev of Gen.	Tabu Iterations
dsjc125.1	125	1472	6	0.23	0.12	2.4	0.001	240
dsjc125.5	125	7782	18	12.92	11.44	307.15	0.31	30715
dsjc125.9	125	13922	45	2.6	1.6	77.2	0.07	7720
dsjc250.1	250	6436	9	46.33	20.8	108.5	0.06	10850
le450_25a	450	8260	25	19.14	8.15	11.75	0.006	1175
le450_25b	450	8263	25	5.43	2.81	2.06	0.001	206

**Table 3 - DSJC and Leighton graphs Tabu Length 100 iterations**

The Table 4 represents the effect of tabu tenure on the miles1000 graph. The tabu length is set to 2000 iterations. It is observed that for higher values of tabu tenure, the algorithm converged faster to a solution.

Tabu Tenure	Avg. Time	Std. Time	Avg. Ite	Std. Iter	Success
128 (n)	0.37970	0.51563	2.66	0.00281	100
256 (2n)	0.24968	0.32026	1.98	0.00169	100
512 (4n)	0.18662	0.20172	1.58	0.00100	100
1024 (8n)	0.13438	0.12099	1.34	0.00062	100

**Table 4 - Effect of tabu Tenure on miles1000 graph**

The Table 5 represents the effect of tabu length on the miles1000 graph. The tabu length has a positive effect in terms of solution quality however after a certain level no gains are attained in the population.

Tabu length	Avg. Time	Std. Time	Avg. Ite	Std. Iter	Success
65536	0.19002	0.606194	2699	9.22	98
32768	0.12318	0.310063	1705	4.77	98
16384	0.11436	0.188074	1560	2.81	98
8192	0.10003	0.137581	1367	2.06	94
4096	0.07444	0.072075	953	1.00	94
2048	0.05288	0.039671	657	0.52	94
1024	0.04834	0.029343	600	0.37	66
512	0.03652	0.018592	408	0.15	36

**Table 5 - Effect of tabu length on miles1000 graph**

### 3.5.5. Comment on Test Results

From the test results, it is observed that difficulty of a graph instance mainly depends upon the edge density. For edge densities around 0.7, the graph instances becomes exceptionally difficult to color whereas overly sparse or dense graphs are somewhat easier to color. It is also observed in solving these easy instances, using tabu search gives the fastest results, whereas for more difficult instances, using a genetic diversification strategy on the tabu search will become useful.

For small instances (which are less than 150 vertices), the tabu search and hybrid algorithm has given results well within 1 second. For larger instances, the standard deviation in the search time becomes larger and using a hybrid works best. However there are some test instances (some DSJC graphs) where difficulty is not related with the largeness but the instance itself. In order to color those graphs, more extensive experimentation is necessary.

### **3.6. Application of Graph Coloring into Other Problem Types**

The presented HET algorithm can be easily ported to solve scheduling and assignment type problems for more constraints. The conflict type hard constraints can still be represented with a constraint graph. This reduces this type of constraint checking into an AND operation thus it is fairly efficient.

The “can not have” or “should be there” type of constraints can be handled with the introduction of strict tabu lists. These tabu lists have non changing tanu tenures therefore all of the moves set here are always forbidden in the search process. For example this may cover a “such operation can not occur in such time slot” situation where the corresponding tabu list location is set. The constraint then can be checked with only an AND operation. Similary “should be here” type of constraints can be handled by setting tabu rules for moves which sends the specific vertex from the desired color set. This constraint checking is again done with multiple checks.

The algorithm does not offer any good constraint checking mechanisms for multiple set and vertex relations. However, there is not an algorithm that we are aware of that solves this constraint checking problem easily. Therefore this type of constraint checking is handled with somewhat costly operations.

The crossover mechanisms should be improved by setting first the constraint related vertices and then applying crossover. This will give the researcher good feasible solutions.

## **4. Conclusion and Further Work**

In this thesis, we have dealt with the algorithms inspired from the nature. We have given a detailed survey of each algorithm and presented the areas that these algorithms may become useful. Note that there are some algorithm that is not mentioned such as “Quantum and DNA Computing”. Later research can cover these areas also.

We have also presented a hybrid evolutionary tabu algorithm which is competitive for a large number of small test instances. However this algorithm is heavily parameter dependent therefore careful experimentations should be handled to have the optimal coloring for especially a large graph. The next step can then be on finding adaptive parameters which will automatically suit to the problem needs. Also the tabu search procedure can be improved by introducing simple ejection moves to provide a simple look-ahead mechanism.

As it is presented in the previous sections, this algorithm can be applied to real time scheduling problems therefore one may want to adapt it to problem needs.

This page is intentionally left blank.

## References

- [1] Bäck, T., Hoffmeister, F. and Schwefel, H.P., 1991, "A Survey of Evolution Strategies", En R. K. Belew and L. B. Booker (Editores), Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, San Mateo, California, pp. 2-9.
- [2] Battiti, R., Tecchiolli, G., 1994, "The reactive tabu search.", ORSA Journal on Computing, 6(2):126—140.
- [3] Comellas, F. and Ozón, J., 1998, "An ant algorithm for the graph colouring problem." ANTS'98 - From Ant Colonies to Artificial Ants: First International Workshop on Ant Colony Optimization., Brussels, Belgium, October 15-16.
- [4] Costa, D., Hertz, A., Dubuis, O., 1995, "Embedding of a sequential Procedure within an evolutionary Algorithm for coloring Problems in Graphs", Journal of Heuristics Vol. 1, No. 1, 105-128.
- [5] Davis, L., "Handbook of Genetic Algorithms", Van Nostrand Reinhold, New York, 1991.
- [6] Dorigo M., Maniezzo, V. and Coloni, A., 1996, "The Ant System: Optimization by a Colony of Cooperating Agents.", IEEE Transactions on Systems, Man, and Cybernetics-Part B, 26(1):29-4.
- [7] Dorigo M., Di Caro, G. & Gambardella G.M. , 1999, "Ant Algorithms for Discrete Optimization", Artificial Life 5(2):137-172.
- [8] Dorne R., Hao J.K., 1998, "Tabu search for graph coloring, T-Coloring and Set T-Colorings.", In Meta-Heuristics, advances and trends in local search paradigms for optimization. INRIA, Kluwer Academic Publishers.
- [9] Eiben, A.E., van der Hauw, J.K., and van Hemert, J.I., 1998, "Graph Coloring with Adaptive Evolutionary Algorithms", Journal of Heuristics, 4(1):25-46..
- [10] Falkenauer, E., 1996, "A Hybrid Grouping Genetic Algorithm for Bin Packing", Journal of Heuristics 1996;2(1):5-30.
- [11] Fotakis, D., A., Likothanassis, S., D. and Stefanakos, S., K., 2001, "An Evolutionary Annealing Approach to Graph Coloring", p. 120-129, Applications of evolutionary Computing: Proc. EvoWorkshops 2001.
- [12] Galinier, P. and Hao, J.K., 1999, "Hybrid evolutionary algorithms for graph coloring.", Journal of Combinatorial Optimization. 3(4): 379-397.
- [13] Glover F., 1989, "Tabu Search – Part I" , ORSA Journal of Computing Vol. 1, No. 3.

- [14] Glover F., 1990, "Tabu Search – Part II", *ORSA Journal of Computing* Vol. 2, No. 1.
- [15] Glover F, Laguna M, "Tabu Search", 1997.
- [16] Haykin, S, 1994, "Neural Networks", Macmillan College Publishing.
- [17] Johnson D.S., Aragon C.R, McGeoch L.A, and Schevon C., 1991, "Optimization by simulated annealing: An experimental evaluation; part ii, graph coloring and number partitioning.", *Operations Research*, 39(2):378-406, 1991.
- [18] Jordan, M.I, Bishop, C.M., *Neural Networks*, 1996, CRC Handbook of Computer Science, CRC Press, Boca Raton.
- [19] Kennedy, J. and Eberhart, R. C., 1995, "Particle swarm optimization", *Proceedings of IEEE International Conference on Neural Networks*, vol. IV, pp.1942-1948, Piscataway, NJ.
- [20] Kirkpatrick S., Gelatt C.D., Jr., Vecchi M.P., 1983, "Optimization by Simulated Annealing", *Science*, Number 4598.
- [21] Kirovski, D., Potkonjak, M., 1998, "Efficient Coloring of a Large Spectrum of Graphs", *Proc. of ACM/IEEE 35th Design Automation Conference*, pp. 427-432, San Francisco, USA.
- [22] Klotz, W., 2002, "Graph coloring algorithms", *Mathematik-Bericht 2002/5*, TU Clausthal.
- [23] Koza,J.R., 1998, "Genetic programming.", In James G. Williams and Allen Kent, editors, *Encyclopedia of Computer Science and Technology*, volume 39, pages 29-43. Marcel-Dekker.
- [24] Michalewicz Z, Schoenauer M. 1996, "Evolutionary Algorithms for Constrained Parameter Optimization Problems.", *Evolutionary Computation* 1996;4(1):1-32.
- [25] Mitchell, M., "An Introduction to Genetic Algorithms", MIT Press, 1996.
- [26] Paun G, Rozenberg, G, "A Guide to Membrane Computing", *Theoretical Computer Science*, vol 287, nr 1 (2002), pages 73-100.
- [27] Radcliffe N, Surry. P, 1994, "Evolutionary Computing: AISB Workshop", (Ed: T. Fogarty, Springer-Verlag).
- [28] Reynolds R.G., 1994, "An introduction to cultural algorithms," in *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, A V. Sebald and L. J. Fogel, Eds., pp. 131-139, World Scientific, River Edge, NJ.

- [29] Reynolds, R. G., Zannoni, E. and Posner, R. M., 1994, "Learning to understand software using cultural algorithms", Proc 3rd Annual Conf. On Evolutionary Programming, San Diego, CA, Feb 1994 , 150-157.
- [30] González-Velarde J.L. and Laguna M., "Tabu Search with Simple Ejection Chains for Coloring Graphs", To appear in the Annals of Operations Research.