American University in Cairo
AUC Knowledge Fountain

Archived Theses and Dissertations

6-1-2004

Variations on particle swarm optimization and their experimental evaluation on maximum satisfiability

Susan A AbdelShahid
The American University in Cairo AUC

Follow this and additional works at: https://fount.aucegypt.edu/retro_etds

 Part of the Computer Engineering Commons, and the Computer Sciences Commons

Recommended Citation

APA Citation
AbdelShahid, S. (2004). *Variations on particle swarm optimization and their experimental evaluation on maximum satisfiability* [Thesis, the American University in Cairo]. AUC Knowledge Fountain.
https://fount.aucegypt.edu/retro_etds/1761

MLA Citation
AbdelShahid, Susan A. *Variations on particle swarm optimization and their experimental evaluation on maximum satisfiability*. 2004. American University in Cairo, Thesis. *AUC Knowledge Fountain*.
https://fount.aucegypt.edu/retro_etds/1761

This Thesis is brought to you for free and open access by AUC Knowledge Fountain. It has been accepted for inclusion in Archived Theses and Dissertations by an authorized administrator of AUC Knowledge Fountain. For more information, please contact fountadmin@aucegypt.edu.

VARIATIONS ON PARTICLE SWARM OPTIMIZATION & THEIR EXPERIMENTAL EVALUATION ON MAXIMUM SATISFIABILITY

SUSAN A. ABDELSHAHID

2004

2004/16

The American University in Cairo

School of Sciences and Engineering

Computer Science Department

Variations on Particle Swarm Optimization and Their Experimental Evaluation on Maximum Satisfiability

A thesis document submitted to

the department of computer science

in partial fulfillment of the requirements of

the degree of Master of Science.

Susan A. Abdelshahid

B.Sc. in Computer Science

Under supervision of **Dr. Ashraf Abdelbar**

May 2004

2004/16

The American University in Cairo

Variations on Particle Swarm Optimization and Their Experimental Evaluation on Maximum Satisfiability

A Thesis Submitted By Susan A. Abdelshahid

to the Department of Computer Science

May/2004

in partial fulfillment of the requirements for

the degree of Master of Science

has been approved by

Dr. Ashraf Abdelbar [redacted]

Thesis Committee Chair

The American University in Cairo.

Dr. Amr Goneid [redacted]

Thesis Committee Examiner

The American University in Cairo.

Dr. Mohamed Gadalla [redacted]

Thesis Committee Examiner

The American University in Cairo.

Dr. Hussein Abbass [redacted]

Thesis Committee Examiner

Australian Defence Force Academy.

[redacted] 27/5/04

Dr. M.N. Mikhail Date

Chair of Computer Science

[redacted] May 31, 2004.

Dr. Fadel Assabghy Date

Dean of Sciences and Engineering

Abstract

Particle Swarm Optimization (PSO) is an evolutionary optimization algorithm based on the idea of imitating swarm life, bird flocks or fish schools.

In this work, PSO was applied on an optimization version of the well-known NP-complete problem, Satisfiability. The algorithm has been implemented with several variations through hybridization with other well-known evolutionary algorithms as well as non-evolutionary techniques.

We have introduced variations on PSO including implementing PSO with instinct-driven particles where particles would stochastically update their status based on an extra component which is meant to represent the particle's innate instinct-level intelligence. We have also introduced a hybrid of instinct-based PSO and stochastic local search where particles perform rounds of local search; in addition, dynamic clause weights was implemented to help escape local minima. Fuzzy PSO is another model where particles would make more use of the information available by their neighbors.

Testing has been conducted on various instances ranging from 100 variables and 900 clauses to 10000 variables and 48000 clauses. Results of different PSO variations have been compared against each other and against the well-known Walksat algorithm.

Acknowledgment

I would like to acknowledge some people, without them I wouldn't have achieved this success and this work wouldn't exist.

First of all, I acknowledge the computer science department at the American University in Cairo, this department has been like a home to me for almost 6 years, ever science I was an undergraduate student. The department with all its faculty, stuff and students is my family. I have great memories with each one of them and in every place.

I would like to give warm thanks to my thesis supervisor, Dr Ashraf Abdelbar, who flooded me with stream of ideas during my thesis work. He was very helpful and supportive. His contribution to this thesis is far beyond being a supervisor.

I would like to thank my committee members, Dr Amr Goneid and Dr Mohamed Gaballa for their cooperation and support. They showed great interest and enthusiasm. I would also like to thank my thesis external examiner for his time and effort in evaluating this work.

I would like to thank my very supportive family. I thank my dad, mom, my sisters and my fiancé Emad Andrews for encouraging me all the time.

Finally, I thank my colleagues, computer science graduate students for helping me through fruitful discussions. They were always there to share with me many ideas and problems that faced me during my work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Particle swarm Optimization, PSO, is a new evolutionary computational paradigm, which involves ideas from several evolutionary computation techniques along with aspects of human and social interactions models. Being a novel algorithm, there is a huge room for improvements. By nature, all evolutionary algorithms are well suited for hybridization with one another and with other non-evolutionary techniques, ideas from one algorithm might enhance the performance of the other, and this was a major work done in this research work.

PSO can be widely applied as a global optimization technique. It has the potential for solving any optimization problem and it has its features, for example multimodality, that gives it an edge over other techniques. Basically, PSO is an algorithm that depends on population of particles each of which represent a potential solution, particles are initialized randomly and are scattered over the search space. Searching for the global minima, particles imitate the behavior of swarms, for example bird groups that learn from the experience of other community members and from their own history.

So far, Local search algorithms have been applied most in optimization and space-search problems, many variations have been suggested and implemented and several algorithms have been developed. However, most of those algorithms have the same bit falls and the major one is local minima. May techniques have been proposed to escape the local minima problem but none of them seem to completely solve the problem. PSO algorithm is designed to avoid this problem since it is a community based algorithm; the algorithm has its own power in escaping local minima; moreover, some modifications have been suggested and tested to deal with such a problem.

In this research work, PSO was applied on the maximum satisfiability problem, Max-SAT. Max-sat is a variation of the satisfiability problem, SAT, which is considered the origin of all NP-complete problems. Max-Sat is an optimization version of the original SAT problem in which

clauses are assigned weights and we seek a solution that achieves maximum weight of satisfies clauses; i.e. the sum of satisfied clauses weights is maximized.

Original PSO algorithm along with other suggested modified algorithms have been applied on weighted max-sat problem. Our modifications include introducing instenct-based particles, integrating local search with PSO and implementing a fuzzy neighborhood model. Results obtained were compared against a famous local search algorithm, Walksat, results are shown and analyzed as well. We discovered that some modifications to PSO could make it competitive to Walksat in many test cases. Those tests were conducted on famous SAT problems publicly available at SATLIB.

Optimization as a large problem domain is discussed in chapter 2. We emphasis on the difference between global and local optimization, and refer to the No Free Lunch theorem. Following that we discuss the concept of Evolutionary Algorithms and the main structure and advantages of such algorithms.

Local search is discussed in chapter 4 along with discussion of the most famous and recent local search algorithms and concepts. We highlight the basic local search pseudo code and then describe the plateau and local minima problems and discuss some techniques for avoiding these problems. We then describe important algorithms like iterated local search, tabu search, GRASP, Walksat and others.

Following that we highlight the problem that we are solving which is Max-sat. We illustrate important features of the SAT problem and famous solving algorithms. We then describe the effort that has been made in Max-sat in particular. Particle swarm optimization technique is described in details in chapter 6. Various algorithm parameters are illustrated and some applications are mentioned. We illustrate how we apply PSO for the max-sat problem describing the particle representation and other data structures. In the following chapter we describe the basic methods and major modifications suggested on the original PSO algorithm and how each affects the solution. The

major approaches are the PSO with instinct, PSO with Local search, PSO with dynamic weights and Fuzzy PSO. Tests and test data are described in chapter 8. Results and analysis are shown in chapter 9. Last but not least we conclude in chapter 10, we show the achievements, obstacles and future work. It is important to note that there is still huge room for research in that area. The PSO model, being very basic, yet explores important ideas and leaves wide space for improvements and modifications. It would be interesting to apply it to different problems and compare against more methods.

2. OPTIMIZATION

2.1 Introduction

Optimization is a computational problem in which the objective is to find the best of all possible solutions. More formally, find a solution in the feasible region, which has the minimum (or maximum) value of the objective function.

Optimization involves determining the values of different parameters so that some measure of optimality is satisfied. Optimization includes both maximization and minimization problems, as the maximization of function f can be seen as the minimization of –f. Some optimization problems involve only linear models and thus called linear optimization, others are non-linear optimization problems.

Optimization is divided into constrained and unconstrained optimization. In constrained optimization, parameters are constrained to be in certain ranges (for example they have to be positive values).

Constrained optimization can be viewed as:

Given: $f(x)$ such that $x \in S \subseteq \mathbb{R}^n$

Where $S = \{x : g(x) \leq 0\}$

$f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$

Find $x^* \in \mathbb{R}^n$ for which $f(x^*) \leq f(x) \forall x \in \mathbb{R}^n$

2.2 Local Optimization

Local optimization is an optimization method in which it is required to find x_B^*, of region B, such that

$$f(x_B^*) \leq f(x), \forall x \in B$$

Where $B \subset S \subseteq \mathbb{R}^n$, and S denotes the search space, note that $S = \mathbb{R}^n$ when dealing with unconstrained optimization. Note also that B is a proper subset of S. The value x_{Bi} will be called local minima.

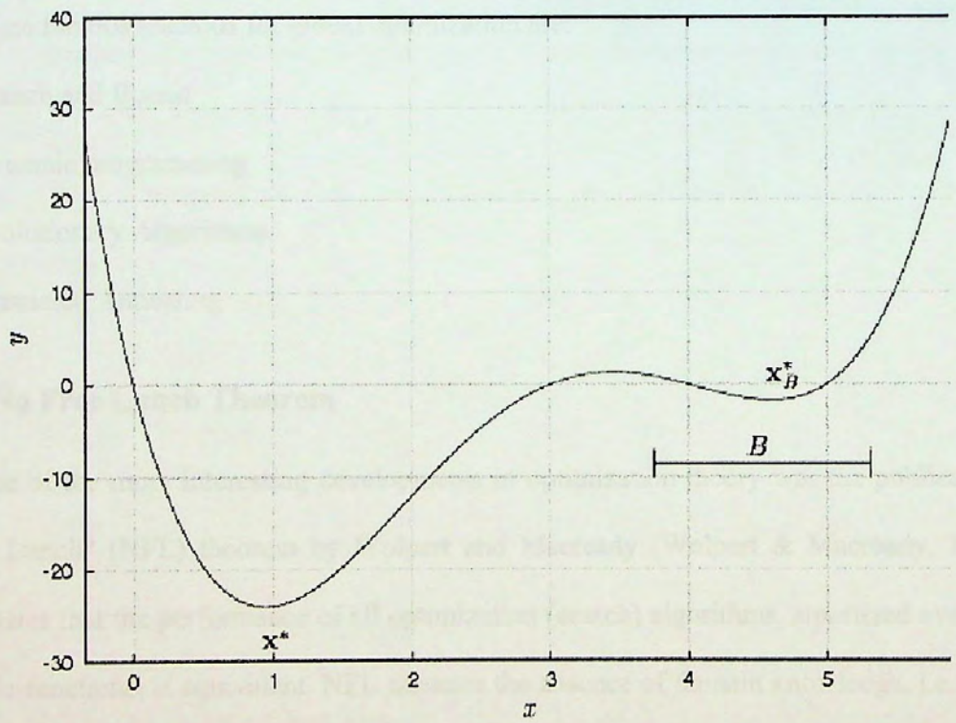

Figure 1: Global minima vs. local minima

2.3 Global Optimization

A global optimizer x^* is defined so that $f(x^*) \leq f(x), \ \forall x \in S$ Where S is the search space, for unconstrained problems, it is common to choose $S = \mathbb{R}^n$. Where n is the dimension of the problem (dimension of x).

Global Optimization (GO) methods can be classified into two main categories: deterministic and probabilistic methods. Most of the deterministic methods involve the application of heuristics; On the other hand, probabilistic methods rely on probabilistic judgments to determine whether or not search should depart from the neighborhood of local minima.

Some famous methods for global optimization are:

Branch and Bound

Dynamic programming

Evolutionary Algorithms

Simulated Annealing

2.3.1 No Free Lunch Theorem

One of the more interesting developments in optimization theory was the publication of the "No Free Lunch" (NFL) theorem by Wolpert and Macready (Wolpert & Macready, 1997). This theorem states that the performance of all optimization (search) algorithms, amortized over the set of all possible functions, is equivalent. NFL assumes the absence of domain knowledge, i.e. algorithms are applied to problems as a black box. The implications of this theorem are far reaching, since it implies that no algorithm can be designed so that it will be superior to a linear enumeration of the search space, or even a purely random search. The theorem is only defined over finite search spaces, however, and it is as yet not clear whether the result applies to infinite search spaces. All computer implementations of search algorithms will effectively operate on finite search spaces, though, so the

theorem is directly applicable to all existing algorithms. Although the NFL theorem states that all algorithms perform equally well over the set of all functions, it does not necessarily hold for all subsets of this set. The set of all functions over a finite domain includes the set of all the permutations of this domain.

Many of these functions do not have compact descriptions, so they appear to be largely "random". Most real-world functions, however, have some structure, and usually have compact descriptions. These types of functions form a rather small subset of the set of all functions. This concern led to the development of sharpened versions of the NFL (Schumacher, Vose, & Whitley, July 2001),showing that it holds for much smaller subsets than initially believed. A more constructive approach is to try and characterize the set of functions over which the NFL does not hold. Christense (Christense & Oppacher, July 2001) Proposed a definition of a "searchable" function, as well as a general algorithm that provably performs better than random search on this set of searchable functions.

3. EVOLUTIONARY ALGORITHMS

3.1 Introduction

Evolutionary Computing, as the name implies, have been designed to simulate the evolutionary process, Evolutionary algorithms are those algorithms that perform search depending on a population of individuals. Typically this population should differ in their fitness, how good the solution represented by each individual, and there should be ways of generating different generations of individual such that each generation inherit features from previous generations.

3.2 Computational Intelligence

By definition, Computational intelligence is "A methodology involving computing that exhibits an ability to learn and/or to deal with new situations, such that the system is perceived to possess one or more attributes of *reason*, such as generalization, discovery, association and abstraction." –David Fogel.

Computational intelligence systems usually comprise a mixture of paradigms such as artificial neural networks, fuzzy systems, and evolutionary computing, augmented with knowledge elements, and are often designed to mimic one or more aspects of carbon-based biological intelligence.

Evolutionary Computing covers approaches to computing based on natural selection, "survival of the fittest." It can be subdivided into such concepts as Genetic Algorithms, Genetic Programming, Evolutionary Strategies and Evolutionary Programming.

Fuzzy Computing deals with vague, incomplete and/or ambiguous data in a precise mathematical way, yet providing solutions based on the human way of thinking. The term *fuzzy* relates to the type of data it processes, not to the technique itself, which is very exact. Fuzzy

Computing is also known as Fuzzy Set Theory or Fuzzy Logic, and covers Fuzzy Control and Fuzzy Expert Systems, for example.

Neural computing covers Artificial Neural Networks, which exist in countless varieties. They are inspired by the human brain. They all use simple mathematical models of brain cells and their power lies in the parallel distributed processing and the ability to learn from their experience.

Apart from techniques from these three areas, combinations frequently occur and one can encounter such fusion systems as neuro-fuzzy systems and evolutionary neural networks.

A system is computationally intelligent when it begins to exhibit:

- Computational adaptivity,
- Computational fault tolerance,
- Reaction speed approaching human-like turn-around, and
- Error rates that approximate human performance.

3.3 History and Applications

Although the ancient Greeks had some rudimentary grasp of the theory of evolution, it was Charles Darwin who first popularized the modern theory of evolution. Credit for this discovery is shared with Alfred Russel Wallace, who independently developed the same theory concurrently. The EC paradigms helped solve problems that were previously considered computationally intractable. Several (somewhat overlapping) categories have been identified to which EC has successfully been applied

- Planning
- Design
- Simulation and identification
- Control

- Classification

3.4 Types of Evolutionary Algorithms

Evolutionary Computation (EC) defines a number of methods designed to simulate evolution. These methods are all population-based, and rely on a combination of random variation and selection to solve problems. Several different approaches exist within the field, including Evolutionary Strategies (ES), Evolutionary Programming (EP), Genetic Algorithms (GAs) and Genetic Programming (GP).

Although similar at the highest level, each of these varieties implements an evolutionary algorithm in a different manner. The differences touch upon almost all aspects of evolutionary algorithms, including the choices of representation for the individual structures, types of selection mechanism used, forms of genetic operators, and measures of performance (Spears, Jong, Back, Fogel, & Garis, 1993).

Evolutionary programming was developed by (L. J. Fogel, Owens, & Walsh, 1966) , the algorithm is as follows, After initialization, all N individuals are selected to be parents, and then are mutated, producing N children. These children are evaluated and N survivors are chosen from the 2 N individuals, using a probabilistic function based on fitness. In other words, individuals with a greater fitness have a higher chance of survival.

Evolutionary strategies were developed by (Rechenberg, 1973). The idea is, after initialization and evaluation; individuals are selected uniformly randomly to be parents. In the standard ES, pairs of parents produce children via recombination, which are further perturbed via mutation. The number of children created is greater than N. Survival is deterministic and is implemented in one of two ways. The first allows the N best children to survive, and replaces the parents with these children. The second allows the N best children and parents to survive.

On the other hand, Genetic algorithms (GAs) were developed by (Holland, 1975). After initializing population, the highest in fitness of those is more likely to be selected as parents. N children are created via recombination from the N parents. The N children are mutated and survive, replacing the N parents in the population. It is interesting to note that the relative emphasis on mutation and crossover is opposite to that in EP (Spears et al., 1993). In a GA mutation flips bits with some small probability, and is often considered to be a background operator. Recombination, on the other hand, is emphasized as the primary search operator. GAs are often used as optimizers, although some researchers emphasize its general adaptive capabilities (De Jong, 1992).

3.5 Advantages of Evolutionary Computing

There are several advantages of Evolutionary computing which have given it the potential for development and growth, here we illustrate some of them as mentioned in (D. B. Fogel, 1997)

- **Simplicity**

The main concept of EAs is simple and easy to comprehend. The always goes as follows: Initialize population randomly, Evaluate fitness, generate new individuals, apply some random variations, select some or all of the individuals to form new generation, then evaluate fitness and so on. All evolutionary algorithms more or less follow this basic flow and hence they are conceptually simple.

- **Wide range of applications**

Evolutionary algorithms can be applied for any optimization problem. It is only required to define the data structure of the expected solution or representation of individuals, the fitness function or the way of evaluating those individuals and the operators that are used for generating new individuals from existing ones.

- **Performance**

Evolutionary algorithms have been shown to outperform many other optimization techniques in a wide range of applications. Real world optimization problems have certain characteristics that make many optimization algorithms fail. First of all, the complexity of the search space as it incorporates many parameters and in most of the cases there exist several local minima

Moreover, the fitness evaluation in most of the statistical optimization methods fails to represent the true fitness of solutions as it depends on the square error between the desired solution and the actual values.

- **Potential of hybridization with other methods**

The framework of evolutionary algorithms leaves a big room for hybridization or mixing the algorithm with other methods. It can be directly hybridized with local search and steepest ascent methods.

- **Parallelization**

Since evolutionary algorithms are basically about having population of individuals then the algorithm can be executed on parallel machines. This gives EAs the potential for solving very large and complex problems and makes a good use of the hardware advancement.

- **Others**

There are several other advantages for Evolutionary algorithms like the ability of self-optimization. Many of the optimization techniques requires setting correct values for their parameters, i.e. requires optimization for their own parameters, evolutionary algorithms have been used extensively for this purpose. For example, Genetic algorithms and particle swarm optimization have been used to optimize training of Neural Networks.

Other advantages include the dynamism of EC and the ability to adapt to changes in the environment and the ability to deal with noisy data.

4. STOCHASTIC LOCAL SEARCH

4.1 Introduction

Local search algorithms are among the standard methods for solving hard combinatorial problems. Local search algorithms search for the optimal solution in the search space. The search basically starts in a random point in the search space of the problem instance and then moves one step at a time. It decides upon which step to take and which direction to move after examining all possible moves from the current location and then choosing the best possible move. Choosing the best possible move at each space depends on the objective function of the problem. The objective function f: $S_\pi \rightarrow \mathbb{R}$ that maps each search space position onto a real (or integer) number in such a way that the maximum (or minimum) number corresponds to the optimal solution depending on whether it is Maximization or Minimization problem.

4.2 Local Search for the SAT Problem

For the satisfiability problem, the search space consists of several points each of which represents a possible truth assignment for the variables of the problem verses the value of the objective function for that candidate solution. For SAT the value of the function is the number of satisfied clauses. For weighted max-sat problem, the value of the function will be the sum of the weights of satisfied clauses by that truth assignment.

Local search algorithms for SAT are incomplete Algorithms i.e. they are not guaranteed to find a solution even if the problem is satisfiable. This is because local search is non-systematic algorithm and can easily be trapped into local minima and plateaus (Holger H. Hoos & Stützle, 1999). On the other hand, systematic SAT algorithms examine the whole search space and thus are guaranteed to find a solution if it exists.

Local Search Algorithm

While TRUE

- **If (enough tries) return fail**
- **Generate initial state**
- **While TRUE**
  - **If (solved) return (Success, state)**
  - **else if enough moves break**
  - **for(each possible step in neighborhood)**
    - **evaluate the move using objective function**
    - **select one of the best moves and update state**

end while

end while

end

Figure 2 Local Search

4.3 Plateau and Local Minima

A local minima is a solution, i.e. a point in the search space, where all neighbors are worse, in SAT context, all neighbors of a local minima satisfy less clauses. The search space typically contains many local minima. The global minimum, on the other hand, is the best solution possible. For SAT there might exist several global minimum, since there might exist more than one solution that satisfy all clauses of the SAT problem instance.

A plateau is basically an area in the space where all solutions have fitness values less tha a desired threshold. A plateau according to (Bohlin, 2002) with respect to a neighbor function N that generate all neighbors from a certain point in the space, and an evaluation function f is a set τ of connected assignments, where all assignments give the same evaluation:

$$plateau_{C,N}(\tau) \equiv connected_N(A) \wedge \forall v,u \in \tau : f(v) = f(u)$$

If no state within the states that form the plateau have a neighbor that achieve better evaluation, then this plateau is a local minima (Hampson & Kibler, 1993). However, if there is an exit from the plateau, i.e. there is a way of achieving better state then it is called a bench or an exit.

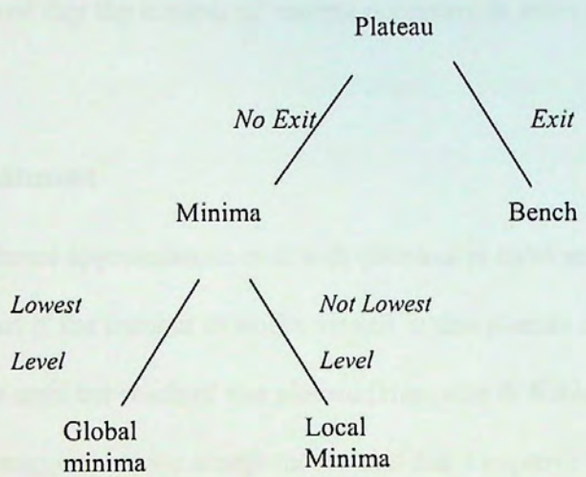

Figure 3 Local Search

4.3.1 Plateau and the Search Space

It was observed that the density of improvable states, i.e. states that can lead to better result, in the plateaus decreases rapidly with depth, so that very little search is needed to find an improvement except for the bottom few plateaus. In other words, the proportions of local minima are high at shallow levels regardless of the ratio of clauses to variables, i.e. the problem of local minima exists even in under constrained problems which imply that even easy problems have local minima (Jeremy Frank, Cheeseman, & Stutz, 1997). Consequently, overall problem complexity is dominated

to a large degree by the complexity of searching the bottom plateaus (Bohlin, 2002; Hampson & Kibler, 1993).

Hampson and Kibler also observed that the number of bottom few plateaus increases exponentially with the number of variables in a SAT problem. Moreover, they discovered an inverse relation between the numbers of possible solutions at those bottom plateaus with the number of variables. It was observed that the number of restarts necessary to solve a problem increases with the number of variables.

4.3.2 Plateau Treatment

There exist different approaches to deal with plateaus in local search. One method suggest to quit a plateau and restart if the number of nodes visited in this plateau exceeds the number of nodes expanded from the start until the reach of this plateau.(Hampson & Kibler, 1993).

Other methods suggest that we accept moves that don't improve the solution or what is called a flat move and hope that a sequence of flat moves would get the search out of the plateau area.

The problem with local minima is that there size can be huge despite the problem difficulty according to (Jeremy Frank et al., 1997) the size can go up to 1000 states.

4.3.2.1 Randomization

Randomization plays great role in local search, basically in two ways. First, the random start of the algorithm and the random restart when the algorithm fails to escape from a plateau or local minima. Another import introduction of Randomization is the Random move (random walk) that the algorithm takes when having to choose between many better moves or flat moves as well. Moreover, some algorithms, like walksat (B. Selman, Kautz, & Cohen, 1994b) introduce a third kind of randomization in which the structure of neighborhood is based on selecting an unsatisfied clause at random then choosing a variable to flip to satisfy this clause.

4.3.2.2 Dynamic weighting

One observation in plateaus is that some clauses are violated more than others (Bohlin, 2002) hence it is possible to change the weights of the clauses dynamically to force the algorithm not to ignore those clauses (in case of weighted max-sat) or assign them weights if they don't have (in case of normal SAT). This approach was first introduced by (Selman & Kautz, 1993).

The weights might be updated either at a plateau or after each iteration. (Frank, 1997) concludes that updating the weights at each iteration is more effective in getting better results, while (Shang & Wah, 1998a) claim that it is better to update weights only at plateaus and local minima.

4.4 Important Local Search Based Algorithms

4.4.1 GSAT

GSAT is a greedy local search for satisfiability problems. it was originally introduced by (B. Selman, H Levesque ,D. Mitchell,1992) for solving the Satisfiability problem (SAT), which is known to be a NP-complete problem. The search begins at a random complete truth assignment. The neighborhood of a point in the search space is defined as the set of assignments that differ from that point by the value assigned to a single variable.

Each step in the search thus corresponds to "flipping" the truth-value assigned to a variable. The basic search heuristic is to move in the direction that maximizes the number of satisfied clauses. According to (Ian Gent & Walsh, 1993) the search process in GSAT is divided into two phases. The first phase most of the flips lead to improvement in result. While in the second phase, search moves (flips) don't improve the state, it only explores more states in what is known as a plateau area.

The pseudo code of GSAT algorithm is illustrated in the following figure.

```
procedure GSAT
Input: a set of clauses ff, MAX-FLIPS, and MAX-TRIES
Output: a satisfying truth assignment of ff, if found
begin
for i := 1 to MAX-TRIES
  T := a randomly generated truth assignment
  for j := 1 to MAX-FLIPS
      if T satisfies ff then return T
      p := a propositional variable such that a change in its truth assignment
      gives the largest increase in the total number of clauses of ff that are
      satisfied by T
T := T with the truth assignment of p reversed
  end for
end for
```

Figure 4 GSAT Algorithm

4.4.2 HSAT

HSAT was introduced by Gent and Walash, 1993 and basically it is the same as GSAT except that they way of choosing the variable to be flipped is no longer based on randomly choosing a variable that achieve best result. However, HSAT chooses the variable that was flipped longest ago.

HSAT emerged as Gent and Walsh were performing variations on GSAT in three directions. Greediness, Randomness and Memory. HSAT was the algorithm they developed for keeping a memory during the search process to guide the search to go for unexplored paths through the choice of the variable that was flipped longest ago. Thus if there are variables that were never flipped, these

are more likely to be flipped with time (Walsh & Gent, 1993). HSAT was an indication of how important memory element is for the search process.

4.4.3 Walksat

The walksat algorithm by Selman, B., Kautz, H., and Cohen, B. 1994, is a modification of the GSAT in which the condition for flipping a variable is different. First, a clause in the problem instance that is unsatisfied by the current assignment is chosen at random -- the variable to be flipped will come from this clause. Next, a coin is flipped. If it comes up heads (with a probability that is one of the parameters to the procedure), then a variable that appears in the clause is chosen at random. This kind of choice is called a "random walk". If the coin comes up tails instead, then the algorithm chooses a variable from the clause that, when flipped, will cause as few clauses as possible that are currently satisfied to become unsatisfied. This kind of choice is called a "greedy" move. Note that flipping a variable chosen in this manner will always make the chosen clause satisfied, and will tend to increase the overall number of satisfied clauses -- but sometimes will in fact decrease the number of satisfied clauses. This refinement of GSAT was called "WSAT"' (for "walksat") in (B. Selman, Kautz, & Cohen, 1994a)

The weighted MAX-SAT version of Walksat, shown in Fig. 5, uses the sum of the weights of the affected clauses in computing the greedy moves. The parameter HARD-LIMIT is set by the user to indicate that any clause with that weight or greater should be considered to be a hard constraint.

```
Procedure Walksat(WEIGHTED-CLAUSES, HARD-LIMIT, MAX-FLIPS, TARGET,
MAX-TRIES, NOISE)
M := a random truth assignment over the variables that appear in WEIGHTED-CLAUSES;
HARD-UNSAT := clauses not satisfied by M with weight ≥ HARD-LIMIT;
SOFT-UNSAT := clauses not satisfied by M with weight < HARD-LIMIT;
BAD := sum of the weight of HARD-SAT and SOFT-UNSAT;
        TOP LOOP: for I := 1 to MAX-TRIES do
```

```
    for J := 1 to MAX-FLIPS do
        if BAD ≤ TARGET then break from TOPLOOP; endif
        if HARD-UNSAT is not empty then
            C := a random member of HARD-UNSAT;
        else
            C := a random member of SOFT-UNSAT
        endif
        Flip a coin that has probability NOISE of heads;
        if heads then
            P := a randomly chosen variable that appears in C;
        else
            for each proposition Q that appears in C do
              BREAKCOUNT[Q] := 0;
              for each clause C' that contains Q do
                  if C' is satisfied by M, but not  satisfied if Q is flipped then
                      BREAKCOUNT[Q] + = weight of C'
                  endif
              endfor
            endfor
            P := a randomly chosen variable Q that appears in C and whose
            BREAKCOUNT[Q] value is minimal;
        endif
        Flip the value assigned to P by M;
        Update HARD-UNSAT, SOFT-UNSAT, and BAD;
    Endfor;
endfor
print ``Weight of unsatisfied clauses is", BAD;
print M;
end Walksat.
```

Figure 5 Walksat algorithm

4.4.4 Novelty

The Novelty and the R-Novelty algorithm (McAllester, Selman, & Kautz, 1997) are two of the latest local search algorithms for propositional satisfiability. They combine a focus on unsatisfied clauses like in Walksat. The two algorithms follow HSAT's mechanism to prefer flipping variables

that wasn't flipped for longest time. However, the random walk part of Walksat wasn't used. Instead, the variable with the highest score is selected unless it is the most recently flipped variable in the clause. If the most recent variable has highest score, with probability p Novelty picks the variable with the next highest score instead. R-Novelty also takes into account the difference between the two highest costs.

4.4.5 Iterated Local Search (ILS)

The main idea behind ILS is, instead of the starting with a new random point in the search space at each time search restarts; one can make use of previously visited states to generate better solutions (Lorenc, Martin, & Stutzle, 2002).

The algorithm is designed to enhance the performance of local search by iterating. It also makes use of the fact that local search as a black box always generates a near optimal solution. The algorithm can then be restarted from one of the local minima generated in a pervious call to local search instead of restarting at random.

Thus, the basic structure of Iterated Local Search algorithm is, as shown in Figure 6 Iterated Local Search, to use local search to generate a near optimal solution, add it to the list of near optimal solutions then choose in some way one of the solutions in this list as the new start for the local search and keep iterating. One important element of success in ILS is the way it walks between local optimums. This function, called perturbation, should have some features. It should not jump between neighbor states; otherwise we fall in the same local search problem of being trapped in local minima. Second, the walk should be irreversible. i.e. the algorithm cannot jump from A to B and then from B back to A . Perturbation function makes changes to the local optima achieved in the previous local search iteration based on the history of optimums previously visited. It introduces noise to S* to turn it to new start for the coming iteration of local search. The strength of perturbation is the number of changes the function introduces to the local optima obtained from local search. Too strong

perturbation would be very much like starting from random again as it will change S* tremendously. However, if the perturbation is too small the local search is more likely to fall again in the local minimum S* just obtained. Perturbation function is domain dependant.

procedure *Iterated Local Search*
S_0 = GenerateInitialSolution
S^* = LocalSearch(S_0)
repeat
S^t = Perturbation (S* ,history)
S*t = LocalSearch(St)
AcceptanceCriterion(St , S*t , history)
until termination condition met
end

Figure 6 Iterated Local Search

AcceptanceCriterion is the function that determines whether S*t is accepted or not as the new current solution. AcceptanceCriterion has a strong influence on the nature and effectiveness of the walk. Roughly, it can be used to control the balance between intensification and diversification of that search. A simple acceptance criterion is to accept only better states, i.e. accept a walk that result only in better state.

The potential power of iterated local search lies in its *biased* sampling of the set of local optima (Lorenc, Martin, & Stutzle, 2001) that is, the potential of ILS is in the dependency of states visited during the history.

4.4.6 Tabu Search

Tabu search is a technique first proposed by Glover and Laguna (Fred Glover & Laguna, 1993). The basic idea behind it is to use a memory during the search process; that memory is used to determine which neighborhood state should be chosen from the current state.

Tabu search

- choose an initial solution i in S. Set i*=i and k=0.
- Set k=k+1 and generate a subset V* of solution in N(i,k) such that either one of the tabu conditions tr(i,m) $\in$ Tr is violated (r=1,...,t) or at least one of the aspiration conditions ar(i,m) $\in$ Ar(i,m) holds (r=1,...,a).
- Choose a best j=i XOR m in V* (with respect to f or to the function f~) and set i=j.
- If f(i) < f(i*) then set i*=i.
- Update tabu and apsiration conditions.
- If a stopping condition is met then stop. else go to Step 2.

Figure 7 Tabu Search

TS is an enhancement of the well known hill climbing heuristic as it searches the entire neighborhood for a solution with a cost better than that of the current solution. If a local optimum is encountered, it will accept the solution whose cost deteriorates that of its predecessor the least (Marcus Randall & Abramson, 1999)Therefore, moves that do not improve the solution can be accepted allowing the algorithm to escape local optima. In order to avoid cycling through previous solutions, cycle avoidance is implemented by a memory structure known as the *tabu list,* Glover (Fred. Glover, 1989; Fred Glover, 1990). In this study, we examine the function of the tabu list in the search process.

While the primary purpose of the tabu list is to avoid repeatedly visiting the same (or similar) solutions (Fred. Glover, 1989). There are a number of different forms that the tabu list can take. Glover proposed a tabu list that stores the solution attributes of a move so as to prevent the effect of reversing that move at some later time. This type of tabu list has a finite length and if the capacity of the list is exceeded, the oldest tabu item is replaced by the most recent. However, it can be expensive to reconstruct a given state because it requires re-application of a number of moves. Because of this problem, the most commonly used form of tabu structure, called the *partial attribute list* (Osman, 1995), only stores some of the state information that describes a solution. For example, if a problem uses 2-opt as the transition operator, then the tabu list might only contain the identity of the items that have been swapped. A side effect of implementing a partial attribute tabu list is that it does not uniquely represent each point in search space, and some points may be regarded as tabu when they have not been investigated. Thus, some solution states that could improve on the currently best known solution might be rejected. One way of avoiding this problem is to use an *aspiration function* which accepts the move provided that it is better than any previously recorded.

Tabu search belongs to the class of dynamic neighborhood algorithms where the structure of neighborhood changes from a state to the other. In tabu search, the neighborhood, i.e. the number of possible states, moves, from the current state, is affected by the history, memory, of the search process. This memory helps search to avoid visiting states that it already visited before. Such states get eliminated from the neighborhood(Hertz, Taillard, & Werra, 1995).

4.4.7 Greedy Randomized Adaptive Search Procedure (GRASP)

Greedy Randomized Adaptive Search Procedure is a randomized heuristic for combinatorial optimization. GRASP was first introduced by Feo and Resende in 1995 (Feo & Resende, 1995). GRASP is an iterative algorithm, each iteration consists of two phases a construction phase and a local search phase, each iteration comes up with a solution, the best solution is kept as the result.

In the construction phase, a possible solution is generated. This phase is a sequence of iterations in each iteration a value is assigned to each variable in the solution; the value is determined by a greedy function. The functions measure the benefit of assigning a certain value to that variable, it is adaptive because this value changes and gets updates as the construction goes on from one variable to the other. The probabilistic element in GRASP comes from the fact that an iteration might come up with a solution that is not the best it have seen through the greedy construction function, all solutions that an iteration generates are kept in Restricted Candidate List (RCL) list and the best one should be reported, however with a certain probability, we might not choose the best, but choose a random solution from that list. This technique allows for different solutions to be picked up at each iteration.

The solution generated in construction phase is then refined in the local search phase. Local search is then conducted on a small neighborhood of that solution to come up with an optimal solution in that neighborhood. The process of local search can take exponential time however, that time is reduced since most probably the search is starting from a point so close to the optimal, i.e. the solution previously generated by construction phase should save a lot of time and effort in the local search phase.

4.4.8 Discrete Lagrange-Multiplier (DLM)

DLM, was introduced by Wah and Shang in 1998 (Shang & Wah, 1998b). DLM is arguably the most efficient procedure for local search at present. It is based on dynamic clause weighting and uses tabu search to help in traversing plateaus.

Largrangiant method is used where the search tried to minimize the objective function, while relying on unsatisfied clauses in the constraints to help the search escape local minima

4.4.9 Smoothed Descent and Flood (SDF)

SDF is a recent local search algorithm for the SAT problem, introduced in (Schuurmans & Southey, 2001)The main contribution of the SDF algorithm is the refinement of the cost function, which is done to guide the search when traversing plateaus. The idea is to break ties between equal-valued assignments by taking into account the number of variables that satisfy each clause. Also, multiplicative weight updating is done, in contrast to the common additive updating. The authors of claim that this gives SDF superior performance over other clause weighting approaches when measuring assignments investigated.

5. WEIGHTED MAXIMUM SATISFIABILITY (MAX-SAT) PROBLEM

5.1 Introduction

This chapter describes the basic Satisfiability problem along with its variations among which is weighted max-sat which is the problem we consider solving. We show different problem characteristics along with some of the famous algorithms for solving it.

5.2 Satisfiability problem

In SAT, a problem p contains M clauses $C_1C_2C_3.....C_m$ and N variables $x_1x_2x_3....x_n$. p is a conjunction of clauses $C_1 \wedge C_2 \wedge C_3 \wedge ..... \wedge C_m$ where each clause is a disjunction of literals $X_1 \vee X_2 \vee ..X_{c_j}$ where c_j is the number of literals in clause C . Each literal X_1 is a variable x_i or the negation $\overline{x_i}$ of that variable. It is required to find truth assignment for variables $x_1x_2...x_n$ that satisfies all clauses $C_1C_2..C_m$. The satisfiability problem (SAT) is the basic NP-Complete problem. It is known that all other NP-Complete problems can be mapped to SAT.

Max-sat: in max-sat problem, the objective is to find truth assignment for the variables to satisfy as much clauses as possible (satisfy maximum number of clauses), where each clause is a disjunction of literals each of which is a variable or the negation of that variable.

Weighted max-sat is a variation of Max-sat where there is a weight attached to every clause and it is desired to find truth assignment for the problem variables so as to maximize the weight of satisfied clauses. Max-sat can be viewed as instance of weighted max-sat in which all weights are equal to 1.

5.3 Approximation Algorithms for Max-sat

An approximation algorithm is an algorithm that produces, in polynomial time, a solution F such that:

$F > cF^{*}$ where $0 < c < 1$ and F^{*} is the optimal solution

The first approximation algorithm for MAXSAT is described in (Johnson, 1974) where a $\frac{1}{2}$ approximation algorithm is presented When each clause has at least k literals, then the algorithm becomes a $(1-\frac{1}{2^k})$ approximation. This algorithm is basically assigning a value to each variable to satisfy the maximum number of clauses. Until recently, the best approximation algorithms were those by (Goemans & Williamson, 1994; Yannakakis, 1992)with $c=\frac{3}{4}$ they solved Max-sat through the linear relaxation of the mixed integer linear program that is equivalent to Max-sat problem. Subsequently, (Goemans & Williamson, 1995) described an improved 0.758 -approximation algorithm based on semi definite programming.

5.4 Local Search Algorithms for Max-sat

According to (Stützle, Hoos, Roli, & Dorigo, 2001), most local search algorithms for max-sat use a 1-flip neighborhood relation. i.e. neighbors differ from each other in the truth value of one variable. The fitness function is typically the sum of the weights of satisfied clauses. Since local search algorithms are incomplete, they might fail to find a solution for a satisfiable SAT instance.

5.5 Generating Satisfiability Problems

5.5.1 Randomly Generated Instances

Generating SAT instances of fixed length clauses, K-SAT problems is basically given C clauses N variables and K (which is the desired clause length) each clauses of the M clauses contains 3 randomly chosen variables out of the N vars. Each variable is then negated with probability 0.5.

5.5.2 Difficulty of the Problem

It has been shown (Bert Selman, Mitchell, & Levesque, 1996)that the difficulty of a SAT instance is closely related to the ratio of clauses to variables, M/N. Several tests have been conducted on many randomly generated SAT instances using the Davis Putman procedure for solving SAT, which is a complete search procedure based on backtracking. They measured the difficulty of the problem by the number of recursive calls DP procedure makes. They stated that if the ratio of clauses to variables is low, the problem difficulty is low. At small ratio each variable is participating in small number of clauses thus the problem tends to be under-constrained; consequently, it is easy to find satisfying truth assignment. On the other hand, for high clauses to variables ratio, the number of clauses each particle is participating in tends to be high and thus the problem is over-constrained and it is easy to show that no satisfying truth assignment exists and hence there is no solution. Again, at high ratios the problem difficulty is low.

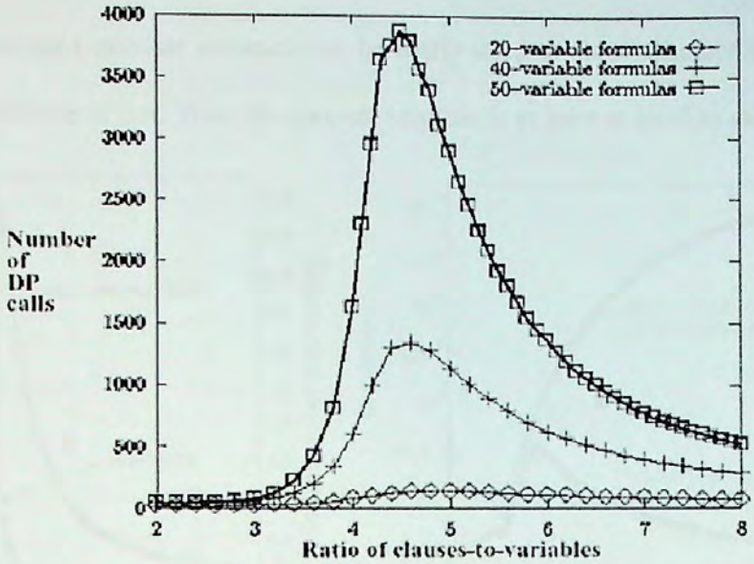

Figure 8 Problem Difficulty VS the ratio of clauses to variables

However, problems that lay in the middle with no high or low ratio are the hardest kind of problems because it is not clear whether they are satisfiable or unsatisfiable.

Some research have been conducted on the distribution of the solutions over the search space and its relation with the ratio of clauses to variables, (Josh Singer, Ian Gent, & Smaill, 2000) state that the Hamming distance between the solutions (they take into consideration not only 100% satisfying solutions but including all solutions that miss up to 5 clauses, which are near optimal solutions) has a inverse relation with m/n ration, i.e. with the increase of clauses to variables ratio, the solutions tend to be clustered together and the hamming distance tends to decrease. Moreover, they concluded that the hamming distance has an effect on the search cost, such that keeping all parameters controlled, there is a linear relation between the hamming distance and the log of the cost.

5.5.3 Difficulty of Max-sat Compared to SAT

According to (Zhang, 2001) the difficulty of maximum satisfiability problem as an optimization problem is much more difficult that satisfiability (SAT) as a decision problem since the

optimal solution of a max-sat instance can be easily used to show whether the corresponding SAT instance is satisfiable or not. Thus the max-sat solution is at least as hard as sat solution.

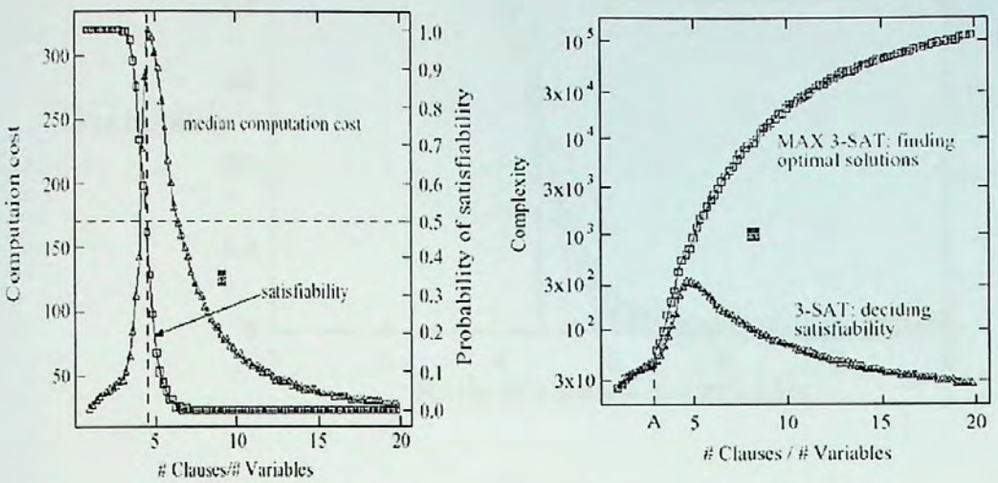

Figure 9 a) phase transition of 3-SAT b) phase transition of Max-sat

As it is shown in figure 9, while the computational cost of 3-SAT decreases after the peak, the difficulty of Max-sat continue to increase exponentially.

5.5.4 Probability of Generating Satisfying Formulas

Selmen, Mitchel and Levesque state in (Bert Selman et al., 1996) that the probability of generating satisfiable formulas decreases with the increase of clauses to variables ratio. They state that it is very likely to find satisfying truth assignment when each variable is participating in small number of clauses; however, it is very likely that when each variable is participating in many clauses that no truth assignment exists that satisfy all clauses. In between the high and low ratios the probability of generating satisfiable problems is 50%. They then conclude that the area where 50 % of the formulas are satisfiable is the hardest set of problems. It was shown that hardest SAT problems are generated with a ratio of clauses o variables equals almost 4.3. i.e. when the number of clauses is 4.3 the number of variables the problem is not obvious whether it is satisfiable or unsatisfiable hence it becomes most difficult.

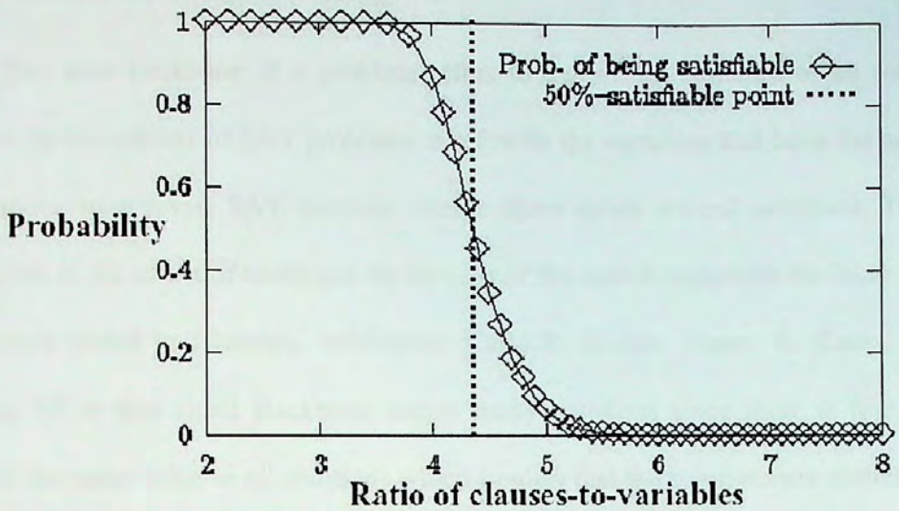

Figure 10 probability of generating satisfiable SAT instances vs. the ratio of clauses to variables.

5.5.5 SAT- Encoded Problems

The other type of SAT problems is SAT problems that are encoding of other domains problems like planning problems

- SAT encoded graph coloring problem
- SAT encoded Quasigroup (Latin square)
- SAT-encoded bounded model checking
- SAT encoding of the Towers of Hanoi problem

Hoos (Hoos, 1999) investigated the effect of the encoding process on the difficulty of the resultant SAT-Encoded problem. He wanted to measure the problem difficulty related to the number of variables used to represent it, he concluded that having few variables and more compact search space doesn't make the problem easier, on the contrary it results in very difficult search space. He also showed that encoding problems and solving them as SAT problems using state of the art SAT solvers proves to perform better than solving problems by their domain specific algorithms.

5.6 Backbone of the Problem

The term backbone of a problem refers to the shared structure of all solutions to a given instance. In the context of SAT problems it refers to the variables that have the same assignment in all solutions to a given SAT instance, incase there exists several solutions. There exist several approaches to the effect of backbone on the cost of the search especially for local search procedures. It has been stated in (Dimitris Achlioptas, Carla P. Gomes, Henry A. Kautz, & Selman, 2000; Johnson, 1974) that small Backbone means many solutions since there is few variables that are assigned the same value in all solutions which implies that the solutions are scattered over the search space and hence easy to find by chance. Also, near 100% Backbone means solutions tightly clustered since all the constraints "vote" in same direction and hence finding a solution is not too difficult. However, SAT instances that have 50% Backbone size have solutions in different clusters subsequently, different clauses push search toward different clusters. Such problems are quite difficult especially for locale search procedures.

A different approach states that the search cost tends to increase rapidly with the increase of backbone size, (Josh Singer et al., 2000) also refers to the concept of backbone-fragility that means that removing few clauses would decrease the backbone size of the problem dramatically, i.e. when there are few number of clauses that contribute to the backbone. He claims that the keeping the backbone size constant, the problem becomes harder when increasing the fragility of the backbone.

5.6.1 Backbone and Clauses to Variables Ratio

The backbone of the problem is closely related to the ratio of clauses to variables (Zhang, 2001). At small ratios, the backbone of the problem tends to be so small. However, as the ratio increase, the backbone tends to increase. Precisely, below clauses-to-variables ratio of 3.6 there tend to be 0 backbone size for the problem, above that the size of backbone tends to increase rapidly.

5.7 Algorithms for solving Max-sat Problem.

5.7.1 Davis Putnam Algorithm

The algorithm by (Davis & Putnam, 1960) is one of the earliest complete algorithms for solving the satisfiability problem. It is basically a back propagation algorithm that is guaranteed to find a solution for any SAT problem if it is satisfiable.

Given a set of clauses S defined over a set of variables V :_

If S is empty, return "Satisfiable".

If S contains an empty clause, return "unsatisfiable".

(Unit-Clause Rule) If S contains a unit clause, assign to the variable mentioned the truth value which satisfies this clause, and return the result of calling DP on the simplified formula.

(Splitting Rule) Select from V a variable which has not been assigned a truth value. Assign it a value, and call DP on the simplified formula. If this call returns "satisfiable", then return "satisfiable". Otherwise, set to the opposite value, and return the result of calling DP on the re-simplified formula.

Figure 11 DP Algorithm

5.7.2 Johnson Algorithm

Algorithm in (Johnson, 1974) was the first approximation Algorithm to solve the Maximum Satisfiability problem. The algorithm is simple and straight forward, it start by all clauses unsatisfied and all variables unassigned and then take a variable at a time and iterate through clauses to discover whether the variable appears more as positive or negative literal in the set of unsatisfied clauses and then decide whether to assign it 0 r 1. after assigning a value to the variable, it is removed from the unassigned variables set and the clauses that it satisfied are removed from the unsatisfied set then the algorithm keeps iterating until all clauses are assigned. The algorithm is a $1-\frac{1}{2^k}$ approximation algorithm, where k is the number of literals per clause.

Given a set of clauses C $\{ C_1, C_2, ..., C_m \}$ and set of Variables V $\{x_1, x_2, ..., x_n\}$

1. For each clauses Cj do $M(c_j) = w_j / 2^{C_j}$
2. LEFT = $\{C_1, C_2, ..., C_m\}$
3. for $t = 1$ to n do

   find all clauses $C_1^T, ..., C_r^T$ in LEFT that contain x_t;

   find all clauses $C_1^F, ..., C_s^F$ in LEFT that contain $\overline{x_t}$;

   if $\sum_{i=1}^{r} M(C_i^T) \geq \sum_{i=1}^{s} M(C_i^F)$

   then $x_t = TRUE$; delete $C_1^T, ..., C_r^T$ from LEFT;

   for i = 1 to s do $M(C_i^F) = 2\,M(C_i^F)$

   else $x_t = FALSE$; delete $C_1^F, ..., C_s^F$ from LEFT;

   for i = 1 to r do $M(C_i^T) = 2\,M(C_i^T)$

Figure 12 Johnson Algorithm

6. PARTICLE SWARM OPTIMIZATION

6.1 Introduction

In this chapter we describe the basic Particle Swarm Optimization (PSO) engine. We examine the effect of different algorithm parameters. We show strengths and problems with the current algorithm and some of its major applications. We then describe how PSO can be applied to MAX-SAT.

6.2 History

Particle Swam Optimization is a population based algorithm first proposed by Kennedy and Eberhart (R. C. Eberhart & Kennedy, 1995; J. Kennedy & Eberhart, 2001) to simulate the behavior of organisms "particles" in a bird flock or fish school. It has roots in two main component methodologies, on one hand is artificial life (A-life) in general bird flocking, fish schooling and swarming theory in particular, and Evolutionary Computing on the other hand.

Particle swarm makes use of the important philosophy of collective intelligence. The concept refers in general to the fact that a group's ability to solve a problem is more than the individuals by themselves (Heylighen, 1999). Complex, apparently intelligent, behavior may emerge from the synergy created by simple interactions between individuals that follow simple rules. Although inspecting the behavior of animal groups gives the impression that they are intelligent creatures, the inspection of the behavior of each creature alone shows that they have extremely limited information processing capacities. In that respect, an insect collective behaves like the self-organizing systems studied in physics and chemistry (Bonabeau et al. 1997). On the other hand, collective intelligence still faces some obstacles, which basically lie in the inability of individuals to make use of the information given by other individuals. The problem might be that information arrives at the other

party in noisy or unclear format that the receiver cannot make use of it. Another problem might exist because of the greediness of the individuals themselves, as sometimes they are keen to appear smarter and better than other individuals in the population.

PSO has to do with the study of the collective behavior of those simple individuals in interacting with each other and the environment. Suppose the following scenario: a group of birds are randomly searching for food in an area. There is only one piece of food in the area being searched. All the birds do not know where the food is, but they know how far the food is in each iteration. So what's the best strategy to find the food? The effective one is to follow the bird which is nearest to the food. the basic idea is to have the swarms fly over the search space to figure out the optimal solution. It is then important to protect them from falling into local minima.

As sociobiologist E. O. Wilson (Wilson, 1975) has written, in reference to fish schooling, "In theory at least, individual members of the school can profit from the discoveries and previous experience of all other members of the school during the search for food. This advantage can become decisive, outweighing the disadvantages of competition for food items, whenever the resource is unpredictably distributed in patches." This statement suggests that social sharing of information among the population offers an evolutionary advantage: this hypothesis was fundamental to the development of particle swarm optimization (J. Kennedy, 1998).

The particle swarm concept originated as a simulation of simplified social system. The original intent was to graphically simulate the choreography of bird of a bird block or fish school. However, it was found that particle swarm model can be used as an optimizer.

PSO is a very attractive Global Optimization technique because it is easy to implement and computationally inexpensive since its memory and CPU speed requirements are low (R. C. Eberhart, Simpson, & Dobbins, 1996) and there is no gradient information required.

6.3 PSO Algorithm

In PSO, each particle represents a solution. First particles are flown over the search space (i.e. randomly initialized), and then evaluated as how close each of them to the optimal, each particle would then adjust its position, i.e. take a step, based on the current best particle, i.e. closest to the optimal. Then, the process is repeated.

What is a particle? Any particle must contain information about its current position in the search space, its velocity (the direction in which it is moving), and the best position that that particle have every visited before.

In each iteration, each particle adjusts its position based on three factors, first, the position of the current best particle, second, the position of his own best previous experience, the third factor is this particle's previous velocity. There are factors for each of those three terms that control its effect. Multiplying previous velocity by larger factor than others, for example, will put more emphasis on its value. There are several approaches as to how to look at the best particle. One approach is to consider one global best particle and all particles follow it, i.e. in each iteration a certain particle is chosen as the leader or the best particle and everybody will try to follow him, in that case it is called *gbest*. Another approach is for each particle to have a neighborhood and each particle follows the best particle in his *topological neighbors*, it is called *lbest*.

Each particle update its position based on the value of its adaptive velocity, which is calculated based on the 2 factors mentioned above. It would then update its position based on its velocity.

$$v_{id} = wv'_{id} + \phi_1 r_1 (p_{id} - x_{id}) + \phi_2 r_2 (p_{gd} - x_{id}) \qquad \text{(a)}$$

$$-v_{\max} \le v_{id} \le v_{\max}$$

$$x_{id} = x_{id}' + v_{id} \qquad \text{(b)}$$

Where, w is a constant called the inertia weight, v_{id} is the particle's current velocity, v'_{id} is its previous velocity, ϕ_1 and ϕ_2 are constants usually set to 2.0, r_1 and r_2 are uniform random variables in the range [0,1].

Initially v_{id} is initialized randomly in the range [- v_{max} , v_{max}]. x_{id} is initialized randomly in the domain of the search space.

```
For each particle
    Initialize particle
END
Do
  For each particle
      Calculate fitness value
      If the fitness value is better than the best fitness value (pbest) in history
        set current value as the new pbest
  End
  Choose the particle with the best fitness value of all the particles as the gbest
  For each particle
      Calculate particle velocity according equation (a)
      Update particle position according equation (b)
  End
  While maximum iterations or minimum error criteria is not attained
```

Figure 13 PSO Algorithm

6.4 Binary and Continuous PSO

There are two variations of the PSO algorithm, one is continuous PSO, and the other is binary PSO. In the continuous version of particle swarm, the search space is real numbers and the Dimensions of each particle are real number values. The velocity of each dimension of each particle is computed by the same equation, however, the position is updated using this equation

$$x_{id} = x_{id}' + v_{id}$$

On the other hand, in binary PSO, each dimension of the particle is a binary number i.e. takes the value 0 or 1 and the solution, as well as the search space, is represented in binary numbers. The particle position is then updated according the following equation

$$\text{If } s(v_{id}) > p \; x_{id} = 1; \text{ else } x_{id} = 0;$$

Where $s(v_{id})$ is the sigmoid function of the velocity of that particle dimension..

$$s(v_{id}) = \frac{1}{\exp(-v_{id})}$$

Hybrid particle swarm refers to using both binary and continuous PSO, the reason might be because the parameters of the problems are both binary and real numbers. One example is the explanation facility for a multiple-symptom diagnostic system. In (J. Kennedy & Eberhart, 2001), the authors describe a neural network designed to diagnose several abdominal diseases such as appendicitis and nonspecific abdominal pain.

The PSO method appears to adhere to the five basic principles of swarm intelligence, as defined by (R. C. Eberhart et al., 1996; Millonas, 1994)

(a) Proximity: the swarm must be able to perform simple space and time computations

(b) Quality: the swarm should be able to respond to quality factors in the environment

(c) Diverse response: the swarm should not commit its activities along excessively narrow channels

(d) Stability: the swarm should not change its behavior every time the environment alters

(e) Adaptability: the swarm must be able to change its behavior, when the computational cost is not prohibitive.

6.5 PSO Versus Genetic Algorithms

PSO shares many common characteristics with GA. Both algorithms start with a group of a randomly generated population; both have fitness values to evaluate the population. Both update the population and search for the optimum with random techniques. Both systems do not guarantee success.

However, PSO does not have genetic operators like crossover and mutation. The stochastic acceleration of a particle towards its previous best position, as well as towards the best particle of the swarm (or towards the best in its neighborhood in the local version), resembles the recombination procedure in EC (I. Rechenberg, 1994; Shi & Eberhart, 1998b). Particles update themselves with the internal velocity. They also have memory, which is important to the algorithm.

Compared with genetic algorithms (GAs), the information sharing mechanism in PSO is significantly different. In GAs, chromosomes share information with each other. So the whole population moves like a one group towards an optimal area. In PSO, only gbest (or lbest) gives out the information to others. It is a one -way information sharing mechanism. The evolution only looks for the best solution. Compared with GA, all the particles tend to converge to the best solution quickly even in the local version in most cases. In PSO, all particles even the bad ones survive through all generations while in GA bad individuals do not survive from one generation to the other (Parsopoulos & Vrahatis, 2002.).

6.6 Multimodality

According to (J. Kennedy & Eberhart, 2001), multimodality means that a problem has more than one solution or global optimum. For example the function $x^2 = 25$ has 2 optimal points +5 and –5.

Multimodal problems can be hard for GA's because chromosomes crossover in every generation. Sections of successful chromosomes are mixed together to produce the members of the new generation. This problem is referred to as deception (Goldberg, Deb, & Horn, 1992). Assume for example that we have a problem whose optimal values are 11001 and 10001, and assume in one generation we have the following chromosomes 01001 and 10011 each of them is one bit away from the optimal, assume the crossover result in the chromosome 01011 taking 010 from the first and 11 from the second. This resulting chromosome is 2 steps away from the optimal which is a worse case than his parents. It is the multimodality of the problem that makes crossover ineffective.

6.7 PSO parameters

6.7.1 Inertia weight *w*

The role of the inertia weight w, is considered critical for the PSO's convergence behavior. The inertia weight is employed to control the impact of the previous history of velocities on the current one (Parsopoulos & Vrahatis, 2002.). Accordingly, the parameter w regulates the trade-off between the global (wide-ranging) and local (nearby) exploration abilities of the swarm. A large inertia weight facilitates global exploration (searching new areas), while a small one tends to facilitate local exploration, i.e., fine-tuning the current search area. A suitable value for the inertia weight w usually provides balance between global and local exploration abilities and consequently results in a reduction of the number of iterations required to locate the optimum solution. Initially, the inertia weight was constant. However, experimental results indicated that it is better to initially set the inertia to a large value, in order to promote global exploration of the search space, and gradually decrease it to get more refined solutions (Shi & Eberhart, 1998a). Thus, an initial value around 1.2 and a gradual decline towards 0 can be considered as a good choice for *w* (Parsopoulos & Vrahatis, 2002.).

6.7.2 Constant Factors

The parameters ϕ_1 and ϕ_2, are not critical for PSO's convergence. However, proper fine-tuning may result in faster convergence and alleviation of local minima. An extended study of the acceleration parameter in the first version of PSO, is given in (J. Kennedy, 1998). As default values, $\phi_1 = \phi_2 = 2$ were proposed, but experimental results indicate that $\phi_1 = \phi_2 = 0.5$ might provide even better results. Recent work reports that it might be even better to choose a larger cognitive parameter, ϕ_1, than a social parameter, ϕ_2, but with $\phi_1 + \phi_2 <= 4$

6.7.3 Random Variables

The parameters r_1 and r_2 are two uniform random numbers used to maintain the diversity of the population, i.e. not to keep the effect of local history and global best the same in all iterations and for all particles. The two numbers they are uniformly distributed in the range [0, 1].

6.7.4 Vmax

Vmax is the maximum velocity a particle can have. The maximum velocity allowed serves as a constrained that controls the maximum global exploration abilities PSO can have (Shi & Eberhart, 1998b). By setting a too small maximum velocity, PSO will always favor local exploration no matter what value of inertia weight is.

6.7.5 Neighborhood topology

There are many ways to set the neighborhood topology as illustrated before; the most famous topologies are gbest and lbest. In the typical lbest structure, every particle is connected to a limited number of neighbors, typically two, which are his left and right neighbors. This is sometimes called a circle topology. In the gbest model, global best, all particles are connected to each other as the whole

population is one group and has one global best, sometimes this is referred to as star topology. According to Kennedy (James Kennedy, 1999) the gbest topology leads to faster convergence; however, it is more likely to fall in local minima's since the whole population is following one leader. Particles might also be arranged in a wheel topology (James Kennedy, 1999), i.e. all particles are connected to one particle and in return that particle is connected to all population.

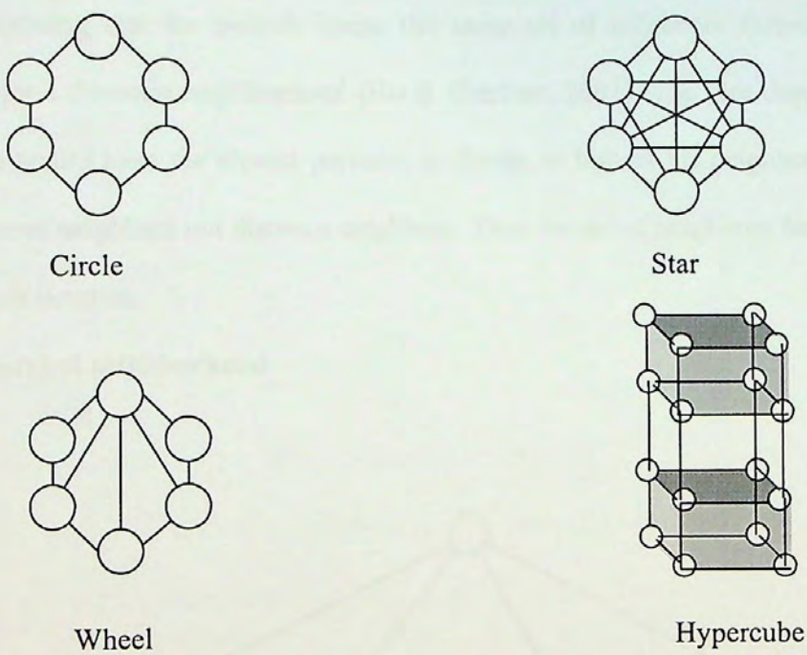

Figure 14 Different neighborhood topologies for particle swarm

Other alternatives also exist like the hypercube topology where particles are arranged in a hypercube, any two particles are considered neighbors if the binary representation of their index differs in one bit. The neighborhood topology has a great impact on the way swarms interact with

each other and the way each particle affect and get affected by his neighbors. Other alternatives (James Kennedy, 1999) describes adding more connections randomly between particles to speed the data communication.

Dynamic Neighborhood

Experiments have shown that it is not recommended that particles keep the same neighborhood during the whole search process. All neighborhood topologies that we have explored are static, meaning that the particle keeps the same set of neighbors forever. There have been suggestions for a dynamic neighborhood (Hu & Eberhart, 2002). The idea described there was that each particle would have the closest particles in fitness to him as his neighborhood. i.e. neighbors would be fitness neighbors not distance neighbors. Thus the set of neighbors for each particle would change in each iteration.

Hierarchal neighborhood

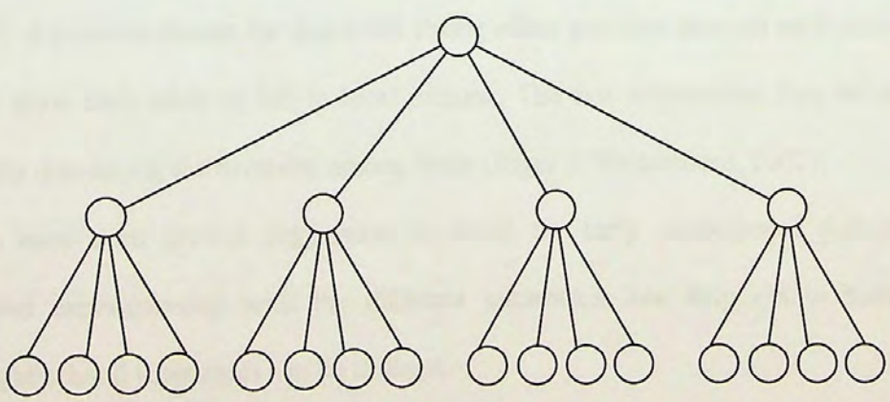

Figure 15 Heirarchial PSO

Hierarchal PSO was introduced in (Jason & Middendorf, 2003) .In H-PSO all particles are arranged in a hierarchy each particle is neighborhooded to itself and his parent. A simple architecture

is to arrange the particles in a tree with particles with highest fitness at the top in order to give them more influence on the group. If the fitness of any particle is greater than his parent's fitness they swap places. Thus the gbest particles will always be at the top of the tree in at most h steps where h is the height of the tree. The velocity that will influence particle x is the velocity of his parent in the tree. Several experiments were conducted by the authors varying the branching degree of the constructed hierarchy. They conclude that at high branching degrees, the algorithm performs better in the early iterations (up to 10000); however, with low branching degree, the algorithm improves slowly at the early iterations but reach better fitness at the end. The authors also experimented the idea of assigned the particles different weights ϕ_1, ϕ_2 according to their level in the tree.

6.8 Problems with PSO

6.8.1 Premature convergence

Premature convergence refers to the case when particles cluster around a solution that is not optimal. A possible reason for this is the strong effect particles have on each other and the fact that they can draw each other to fall in local minima. The fast information flow between particles leads to rapidly decreasing the diversity among them (Riget & Vesterstrom, 2002).

There have been several suggestions to solve the early convergence problem. Ofcourse controlling and experimenting with the different parameters can help obtain better optimizer. Different neighborhood topologies can be utilized.

In their research (Krink, Vesterstrom, & Riget, 2002) suggested a model in which collision between particles can be predicted and avoided (by resetting their positions) in order to avoid clustering, particles having the same values. They investigated bouncing the particles in three methods. Randomly, realistic physical bouncing and simple velocity-line bouncing. In random reset,

particles that are about to collide are simply re-initialized randomly. In physical bouncing the particle positions are slightly changed so that they do not collide. In simple velocity line bouncing particles velocity is speed up or slowed down in an attempt to change directions later.

Other approaches include subpopulations, i.e. dividing the population to groups and limit the interaction between them so that each group follows its own best particle (Lvbjerg, Rasmussen, & Krink, 2001) and giving particles physical extension.

6.9 PSO Applications

6.9.1 PSO of Neural Net Weights

The first real implementation of the particle swarm algorithm was a model that bridges psychological theory and engineering applications. The feed forward artificial neural network is a statistical model of cognition that input vectors of independent variables and outputs estimates of vectors of dependent variables (J. Kennedy & Eberhart, 2001). The network is structured as a set of weights, usually arranged in layers, and the problem is to find values for those weights that make the mapping with minimal error. Normally, feedforward networks are optimized using some sort of gradient descent algorithm, traditionally through back propagation of error.

Evolutionary computation methodologies have been applied to three main attributes of neural networks: network connection weights, network architecture (network topology, transfer function), and network learning algorithms.

The advantage of the EC is that EC can be used in cases with non-differentiable PE transfer functions and no gradient information available. The disadvantages are: 1) The performance is not competitive in some problems. 2) Representation of the weights is difficult and the genetic operators have to be carefully selected or developed.

As in (J. Kennedy & Eberhart, 2001) PSO was tested in training the feed forward NN to solve the XOR problem. The XOR famous function takes 2 inputs and outputs 0 is both are false (0) and 1 otherwise. They deleted the back-propagation training algorithm and substituted particle swarm optimization. The training time required to achieve the same square error dropped from 3.5 hours to about 2.2 minutes, (R. C. Eberhart et al., 1996). Some rules of thumb were developed in using PSO to optimize NN weights. The best value for Vmax was 2.0 which was successful in training NN with sigmoid activation functions. The maximum sum for the 2 acceleration constants ϕ_1 and ϕ_2 was set to 4.0. It was found that a neighborhood size of about 10-20 percent of the population size works well (J. Kennedy & Eberhart, 2001) .

Another simple example of evolving ANN with PSO is the famous iris classification problem (R. C. K. Eberhart, J., 1995). Measurements of four attributes of iris flowers are provided in each data set record: sepal length, sepal width, petal length, and petal width. Fifty sets of measurements are present for each of three varieties of iris flowersh, for a total of 150 records, or patterns.

A 3-layer ANN is used to do the classification. There are 4 inputs and 3 outputs. So the input layer has 4 neurons and the output layer has 3 neurons. One can evolve the number of hidden neurons. However, suppose the hidden layer has 6 neurons. They only evolve the network weights. So the particle will be a group of weights, there are 4*6+6*3 = 42 weights, so the particle consists of 42 real numbers. The range of weights can be set to [-100, 100] (this is just a example, in real cases, one might try different ranges). After encoding the particles, we need to determine the fitness function. For the classification problem, they feed all the patterns to the network whose weights are determined by the particle, get the outputs and compare it the standard outputs. Then they record the number of misclassified patterns as the fitness value of that particle. Now we can apply PSO to train the ANN to get lower number of misclassified patterns as possible. There are not many parameters in

PSO need to be adjusted. They only need to adjust the number of hidden layers and the range of the weights to get better results in different trials.

6.9.2 Other Applications

There are several applications for PSO as a new optimization technique. One of which is reactive power and voltage control. It can be formulated as a mixed-integer nonlinear optimization problem (MINLP). The proposed method expands the original PSO to handle a MINLP and determines an on-line VVC strategy with continuous and discrete control variables such as automatic voltage regulator (AVR) operating values of generators, tap positions of on-load tap changer (OLTC) of transformers, and the number of reactive power compensation equipment. The method considers voltage security using a continuation power flow and a contingency analysis tech

Another application is the detection and classification of human tremor disease. Particle swarm was used in the learning phase of neural network to classify objects as being sick or normal.

6.10 PSO for Max-sat

Since max-sat is an NP-Hard optimization problem it is very challenging to apply particle swarm optimization algorithm to solve it. A big advantage for PSO as well as other evolutionary algorithm is that it can be easily applied to any optimization problem. We are seeking hence a particle to represent a solution to the max-sat instance by which we obtain maximum weight of satisfied clauses.

6.10.1 Particle Representation

To apply PSO to the problem of maximum satisfiability, each particle should represent a potential solution. Hence, each particle would be a representation of the truth values for all the variables in the SAT problem. For example, if we are trying to solve a max-sat instance that contains

ORIGINAL DELIVERED BY CAIRO

100 variables and 500 clauses then each particle in the swarm will contain 100 variables each of which is a mapping of a max-sat variable.

PSO variables will be Boolean variables. i.e. they are allowed to take the values true or false only. We will be applying the binary version of PSO in which the sigmoid of the velocity determines assigning a value of 0 or 1 to the variable.

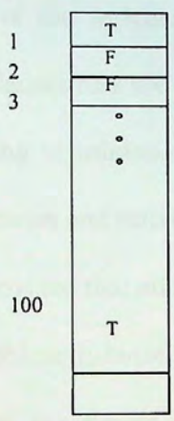

Figure 16 example of particle structure for Max-sat problem

6.10.1.1 Other Data Structures

We keep for each particle

- Velocity: (1D-Array) represents the velocity of each dimension of the particle. In order to calculate current velocity, we need to know the previous velocity for each dimension in each particle.
- Pbest: copy of that particle (represented by 1D-Array) at the best point in its history. pbest represent the best point in the search spaces that this particle visited before.

Problem Instance representation

We use 2D-array to represent the problem instance

Assume we have problem with 100 variables and 500 clauses the array will be 500 x 102 we use 2 extra cells for each clause to save the number of literals in this clause (clause length) since clauses are not of equal length. The second cell is for the weight of that clause.

6.10.2 Fitness Function

The fitness function is the function that is used for evaluating particles and determining whether a solution has been reached. For the weighted max-sat problem, the fitness function is basically the sum of the weights of the clauses that are not satisfied by the current truth assignment represented by this particle. We are trying to minimize this number, thus particles that have less value are better because they miss few clauses and satisfy more. Whenever the result of this function is 0 then this particle missed 0 clauses, provided that all clauses have weights greater than 0, and thus it represents a solution to that max-sat problem instance.

It is very important to note that the fitness function sums the weights of the missed, unsatisfied, clauses. Not the weight of satisfied clauses. If we evaluate the particles based on how much they satisfy, this will not be correct measure because the particle that would satisfy most clauses would be considered a best particle while he could be still breaking a lot of clauses. But since we count satisfied clauses, we wouldn't notice that this particle is actually causing a problem and breaking many clauses. This was a major modification introduced in walksat, (Stützle et al., 2001) highlighted this change in fitness function from counting satisfied to counting unsatisfied clauses as being introduced for the first time in walksat algorithm.

The following is the pseudo-code of the fitness function.

Fitness(Particle P)

- loop over all clauses
  - ➢ set satisfied = false
  - ➢ loop over all literals of that clause
    - if literal = - x and P_x = false set satisfied to true , break
    - if literal = x and P_x = true set satisfied to true , break
  - ➢ if satisfied = false, add weight of clause to fitness value.
- End loop

Figure 17 PSO Fitness Function for Max-Sat

Applying PSO to weighted max-sat can be represented in the following pseudo-code

- loop until max-runs
    - Initialize population randomly (each variable in each particle will be assigned 0 or 1 with 50% probability. Initial velocity for each dimension in each particle will be assigned randomly as well.
    - Loop until you find solution or you reach max tries
        - For all particles
            - Evaluate fitness, how much weight does each particle miss.
            - Choose best particle according to neighborhood structure gbest and determine whether this value is the best in this particles history pbest
            - Calculate velocity for each dimension for each particle based on previous velocity, gbest and local best
        - End for
        - For all particles
            - Update position (determine what new values will be assigned to each variable in this particle based on the sigmoid of the velocity calculated before for this dimension)
        - End for
    - End loop
- End loop

7. VARIATIONS ON THE PSO MODEL

7.1 Introduction

This chapter describes the basic modifications that we have implemented on top of the basic PSO algorithm. The variations include instinct-driven PSO, a hybrid PSO with local search algorithm, what is called Fuzzy PSO and other variations. We also describe new functions and parameters that are used.

7.2 PSO with Instinct-Driven Particles

In an attempt to get better performance, each particle would make use of its instinct to affect the direction in which it will move. In other words, each particle would look at itself, using its natural instinct and intelligence, and try to modify itself to reach a better state.

The instinct function is to basically allow the particle to change affect the value of its velocity. As we have seen, velocity is based on best history, best neighbor and previous velocity values. However there is no way the individual itself can make a choice. The theta, instinct, function is the means by which each particle would think by itself and affect its own velocity. The instinct function can have many forms. It can be a greedy function, i.e. particle would use its instinct to favor changes that lead to better states, or it can be probabilistic function, i.e. each particle would choose probabilistically to push itself in a certain direction. The important thing is to keep the theta as an element of individualism. Possible ideas for instinct function can be driven from other famous optimization techniques.

We implemented a greedy instinct function in which each particle would push itself towards the move that maximizes its fitness. The idea is as follows, after evaluating the fitness function for ac particular particle, a particle would loop over its dimensions try to change the value of each of those

variables (flip it) and re-evaluate its fitness. If the fitness of that particle would increase when flipping this variable, the particle would then assign a theta value to that variable to push it to change to the desired value, i.e. the value that maximize the particle's fitness.

From the nature of the sigmoid function, it is obvious that a negative velocity would cause the result of the sigmoid function to decrease and hence we have higher probability of assigning a 0 to that variable. As a result, when a particle want to flip one of its variables to 0 it is desired to give it negative velocity and visa versa when we need to flip a variable to 1, we need to assign it positive velocity.

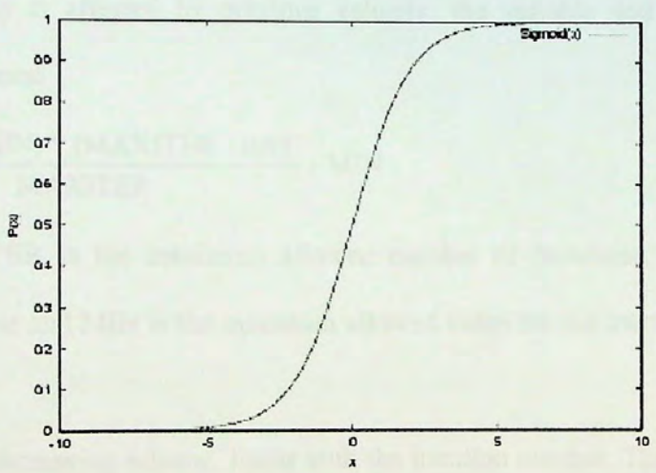

Figure 18 Sigmoid Function

Calculating the theta

Theta value is calculated for each dimension *i* for each particle *x* as follows:

- Flip that variable (if true set it to false and if false set it to true)
- Calculate new particle fitness $g(p_i^{d'})$
- Calculate theta $\theta(x_{id}) = \frac{g(p_i)}{g(p_i) + g(p_i^{d'})} \times (-1)^{\sigma}$

  where $\sigma = 1$ if $x_{id} = 0$ and $g(p_i^{\sim d}) > g(p_i)$ or $x_{id} = 1$ and $g(p_i^{\sim d}) < g(p_i)$

  else $\sigma = 0$

- Flip the variable back (return to original value)
- The velocity equation would then change to

$$v_{id} = wv'_{id} + \phi_1 r_1 (p_{id} - x_{id}) + \phi_2 r_2 (p_{gd} - x_{id}) + \phi_3 \theta (x_{id})$$

Where ϕ_3 is a factor set to control the effect of the theta function.

7.2.1 Parameter Setting

1) *inertia w*

inertia is the factor multiplied by previous velocity, hence it affects the amount by which current velocity is affected by previous velocity, the variable was set to 1.0 with a decreasing scheme

$$w = \frac{\text{(weight-MIN) * (MAXITER - iter)}}{\text{MAXITER}} + \text{MIN}$$

where MAXITER is the maximum allowed number of iterations, iter is the current iteration number and MIN is the minimum allowed value for the inertia *w* which was set to 0.4.

This is linear decreasing scheme, linear with the iteration number. The decreasing inertia allows makes the particle favors global explorations at the beginning and then local explorations at the end.

2) *vmax*

vmax is set to the best value obtained during experimentation, which was 6.0

3) ϕ_1 , ϕ_2

Since a new term was added to the velocity equation, the values of the ϕ_1 , ϕ_2 factors had to be reduced so as not to exceed the maximum velocity and keep it within range. Values of 0.6 and 1.1 were used.

4) Neighborhood topology

Several tests were made on the neighborhood topology, best results obtained were using the hypercube neighborhood as it lies between the two extremes of lbest and gbest. It allows for more data exchange and more communication between particles.

5) ϕ_3

This is the instinct function factor. Since the result of the new theta function is usually a fraction. ϕ_3 was set to 1.0 to keep the value of the term $\phi_3\theta_{id}$ in a reasonable range. It decreases with time according to the following equation. $\phi_3 = \frac{MAXITER - iter}{MAXITER}$

Decreasing the theta effect with time makes the particle more flexible in his moves.

6) Number of particles

Since we were using the hypercube topology, the number of particles should be a power of two. Due to the fact that we are solving large problems with complicated search space, a large number of particles were required; however, too large swarm would cause optimization process to be slow, hence 64 particles were used.

7.3 PSO with Dynamic Weights

As mentioned early, dynamic weights approach have been suggested in (Selman & Kautz, 1993) to overcome the problem of local minima. The idea is basically to change the weights of clauses in an attempt to modify the search space and eliminate the local minima. The idea is, as particles fall in local minima, try to modify the search space so as to raise that point, hence particles would escape that point because they can find better neighborhood states.

To force particles to satisfy all clauses, we increase the weights of those clauses that remain unsatisfied for several iterations. There can be different schemes for increasing weights, you can increase the weight of the unsatisfied clauses linearly, i.e. multiply the weight by a constant factor.

Another alternative is to add constant amount to the weight. However, one should be careful because increasing the weights so much can push particles to leave that local minima to a worse one. If clauses that initially had small weight gained too much weight then the swarm would prefer another solution that lose more clauses that have still small weight. That is why an upper limit was put so that clauses weights do not exceed a certain limit. For example, a clauses weight shouldn't increase to more than double or triple its initial weight. To overcome this situation, chosing the best neighbor would be based on true weights; however the updated weights (dynamic weights) would only be used to evaluate the fitness function used to set the value of the theta function, instinct function, as described previously.

It is worth mentioning that the behavior of the optimization process using this technique is differs in case the problem being solved has a satisfying truth assignment or not, i.e. if the optimal solution is 0.

7.4 PSO with Local Search

We experimented with a combination of local search and particle swarm. The particle swarm algorithm remained the same, with the instinct function, theta. On the other hand, at the end of each iteration, each particle would do k-rounds of local search, in each round, a particle would flip each variable, dimension, it contains and re-evaluate itself, if this would achieve better fitness, the actual value of that variable would then be flipped. This is called 1-opt local search. It is different from the theta function in that when achieving better value, the variable is flipped directly, but in the theta function, it is only given a bias or a push to flip, however the actual value depends on the sigmoid of the whole velocity value.

Local search gives particles the ability for local exploration and neighborhood search as each particle examines all possible neighborhood states that differ only in one variable. It also gives particles a chance for direct change with no randomization. PSO with instinct function and local

search achieved great results that overcome the performance of local search itself when tested against walksat the famous local search algorithm.

7.5 Fuzzy PSO

The current PSO model still suffers from an important defect. Regardless of the neighborhood structure, when calculating velocity, a particle only takes into consideration one best neighbor. That best can be global best or local best with different neighborhood topologies as illustrated before, however, it is still one best neighbor. We thought that a particle should take into consideration a number of neighbors not only one. That best neighbor might be misleading and might lead particles to fall into local minima. That is why a particle would take into consideration the values of a number of neighbors ordered by their fitness. This approach is called Fuzzy PSO.

Fuzzy PSO. In fuzzy PSO, particles don not depend only on one particle as the best however, a particle might look at the best and second best and a number of neighbors rather than one. In the PSO algorithm, according to the velocity equation, particle velocity is composed of 3 components. Those are, Previous velocity, difference between current value and best value in that particles' history, and difference between current value and value if best neighbor. As discussed before, there are several ways to choose the best particle, either the gbest model, lbest or others like hypercube neighborhood structure.

Regardless of the method of choosing the best particle or the neighborhood structure, a particle can look at many best neighbor not only one. Thus according to the neighborhood, particles would be ordered first then second and so on. When a particle calculates its velocity, it doesn't only consider the difference between its value and best particle in the neighborhood, it takes a number of particles into consideration; it might look at the top 2, 3 or more, among its neighbors. Each of the neighbors to be considered is assigned a factor; *ci* that factor depends on the goodness of the solution represented by that neighbor. Those factors are calculated according to the Cauchy function.

Assume a particle is going to look at the best 3 particles in the neighborhood. Assume for particle x these were a, b, c; a being the best, b the second and c the thirst best.

Then the factors for a, b, c will be calculated based on the following Cauchy function.

$$factor(g) = \frac{\alpha}{1+\left(\frac{g-\alpha}{\beta}\right)^2}$$ where α is the best particle in the neighborhood and $\beta = \frac{\alpha}{k}$,

$k \in \{1, 1.1, 1.2, \ldots.\}$

Cauchy distribution is shown in the following diagram, the Cauchy distribution is similar to Gaussian distribution; however, Cauchy distribution is more flat.

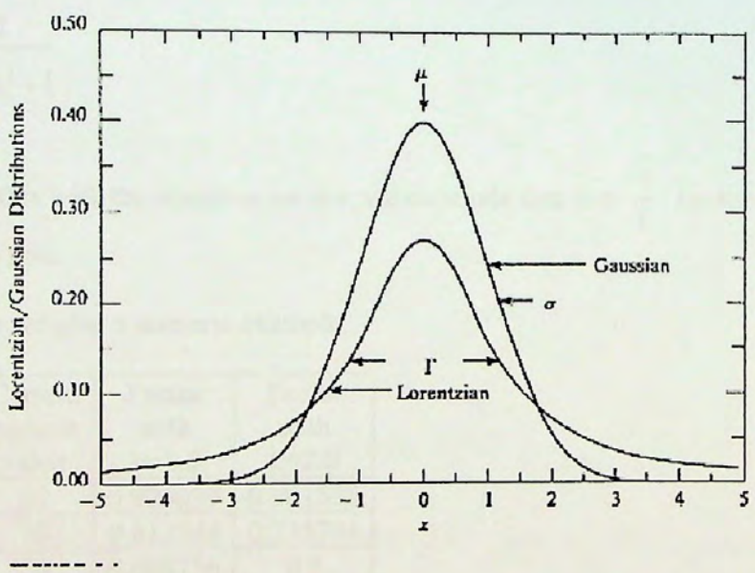

Figure 19 Cauchy Distribution

The Cauchy distribution, also called the Lorentzian distribution, is a continuous distribution describing resonance behavior. Cauchy distribution plays a special role in the theory of statistics because it is a fact that the ratio of any random variables follows the Cauchy distribution (Casella & Berger, 2002).

The Cauchy distribution is often cited as an example of a distribution which has no mean, variance because it has infinite interval.

The original formula for the Cauchy distribution is

$$f(x) = \frac{\Gamma}{x^2 + \Gamma^2}$$

where x_0 is the mode and Γ the full width at half maximum.

Taking $x_0 = 0$ and removing the constants

$$f(x) = \frac{\Gamma}{x^2 + \Gamma^2}$$

$$f(x) = \frac{1}{\frac{1}{\Gamma^2}x^2 + 1}$$

Comparing this with the equation we use, we conclude that $k \propto \frac{1}{\Gamma}$. i.e. the larger k the smaller the graph width gets.

Here we give a numeric example

Best particle value	Current particle value	Factor with k=1.6	Factor with k=2.0
100	90	0.975039	0.961538
100	70	0.812744	0.735294
100	50	0.609756	0.5

On the other hand, The normal Distribution, also called the Gaussian distribution, is a another bell-shaped distribution that can be seen as a general approximation to other distributions in large samples according to the Central Limit Theorem. A normal distribution with mean μ and variance σ^2 is given by

$$f(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-(x-\mu)^2/(2\sigma^2)}$$

In our case, we use μ = Best Value σ =Best value / k

We have used the Cauchy distribution for the distribution of neighborhood factors because it is a flat curve that gives more Varity in value of the variable in reference to the fitness value. Our analysis is similar to that of (Yao & Liu, 1997) where they used Cauchy mutation instead of Gaussian for generating offspring because Cauchy is more likely to generate offspring that is further away from its parents due to its long flat tails, and hence have higher probability from escaping local minima.

Here is a comparison of the factors values obtained using Cauchy vs Gaussian distributions

Best Value	This Particle	Cauchy Factors	Gaussian Factors
100	90	0.975039	0.49812
100	70	0.812744	0.44964
100	50	0.609756	0.36637

The velocity calculation equation would then be

$$v_{id} = wv'_{id} + \phi_1 r_1 (p_{id} - x_{id}) + \sum_{j=0}^{size} f_j \phi_2 r_2 (p_{jd} - x_{id})$$

Where p_j is the j[th] best particle and f_j is the factor calculated using Cauchy distribution based on the goodness represented by p_j

Tests have been made using Fuzzy PSO and compared to the original gbest PSO model better results were obtained. This proves that Fuzzy PSO is a better engine because it results in better communication and information sharing between the particles as particle would look at more neighbors and hence achieve a better benefit from the experience of the rest of the population.

7.6 Other Possible Variations on PSO

7.6.1 k-array d-cube

As explained before, in most of our experiments we used the hypercube structure for neighborhood. In that structure, the number of particles in population should be a power of two 2^n . Each particle would have number of neighbors = n. each neighbor differs in binary representation from that particle in one bit only.

With a k-array d-cube, you would represent the indices of your particles in base-k (starting at 0). For example, with a 4-ary 3-cube, you would have 64 particles (numbered 0..63). You would represent the index number in base 4. Two particles are considered neighbors if their indices differ in exactly one digit. So, therefore, each particle (for k=4, d=3), would belong to 3 different neighborhoods, and each neighborhood would have 4 members.

7.6.2 Simulated annealing

A particle swarm optimizer with instinct, hypercube and simulated annealing was used. Simulated annealing is a famous approach for global optimization. The idea, as the name implies, come from the engineering discipline where materials are heated then cooled according to temperature decreasing schedule. Simulated annealing is an approach where the algorithm might accept a bad move but with a certain probability, this is done to help escape the local minima.

The way the two approaches were hybridized is that after each iteration of PSO, we do k-rounds of simulated annealing. In each round, each particle loop over its variables, flip each of them and re-evaluate its fitness. If it achieve better fitness then it keeps it flipped, if not then it might keep it flipped, but with a certain probability. The algorithm usually starts with a high probability to be more likely to accept different moves, even the ones that result in worst fitness. Then the probability is supposed to decrease with time.

The probability is calculated based on the following equation

$$p = e^{\frac{-\Delta f}{T}}$$

it is desired to set the initial Temperature to a value around 0.8 and then decrease it either linearly or by subtraction i.e.

$T_i = CT_{i-1}$ where $0 < C < 1$

$T_i = T_{i-1} - C$

7.6.3 Randomization

When examining the PSO algorithm, one can notice that there is no way to make a particle change its direction to the opposite. It is true that there are a couple of random variables in the velocity calculation equation (a). However, those random variables affect only the amount by which the values (p_{id} - x_{id}) and (p_{gd} - x_{id}) affect the velocity. However, if the velocity is negative or positive (which will greatly affect the value of the variable to be 0 or 1 , due to the sigmoid function), then there is no way of changing the sign of that velocity, random variables affect only the magnitude or the value of velocity.

That is why the idea of adding an element of randomization to the velocity calculation equation was introduced. There were a couple of alternative methods, either to change the velocity calculation equation to add another term to the equation. That term would be a random value in the range [-0.5 – 0.5]

Another idea is to let the velocity be calculated according to the normal equation and then with a certain small probability change the sign of that velocity (if negative velocity change to positive and vice versa). This could be done to all particles or to certain number (5-10 particles) only.

8. TESTING METHODOLOGY

8.1 Introduction

In this chapter we describe the different max-sat problems upon which we conducted our experiments. Some problems were obtained from the online satisfiability library SATLIB, others were generated by our SAT generator which is also described here.

8.2 Benchmark Problems

8.2.1 SAT-encoded Morphed Graph Coloring Problems

The Graph Coloring problem (GCP) is a well-known combinatorial problem from graph theory: Given a graph $G=(V,E)$, where $V=\{v1,v2,...,v_n\}$ is the set of vertices and E the set of edges connecting the vertices, find a coloring $C: V \rightarrow N$, such that connected vertices always have different colors.

An interesting class of Graph Coloring Problems is obtained by morphing regular ring lattices with random graphs A p-morph of two graphs A=(V,E1) and B=(V,E2) is a graph C=(V,E) where E contains all the edges common to A and B, a fraction p of the edges from E1-E2 (the remaining edges of A), and a fraction 1-p of the edges from E2-E1. The test-sets considered here are obtained by morphing regular ring lattices, where the vertices are ordered cyclically and each vertex is connected to its k closest in this ordering, and random graphs from the well-known class Gnm. The morphing ratio p controls the amount of structure in the problem instances, and by varying p, the behavior of various algorithms depending on the degree of structure and randomness can be studied (I. Gent, Hoos, Prosser, & Walsh, 1999).

SAT encoding

A straightforward strategy for encoding GCP instances into SAT is used, each assignment of a color to a single vertex is represented by a prepositional variable; each coloring constraint (edge of the graph) is represented by a set of clauses ensuring that the corresponding vertices have different colors, and two additional sets of clauses ensure that valid SAT assignments assign exactly one color to each vertex.

It has been shown in (I. Gent et al., 1999) that when the p, morphing ratio, is close 1o 0.01 the problem become much harder for local search procedures and the search cost increases rapidly. The test set was generated using a generator program provided by Toby Walsh and can be obtained from SATLIB. All graphs have 100 vertices and 400 edges. We used 10 problem instances each of which have 100 vertices and 400 edges with 500 variable and 3100 clause and morphing ratio = 2^{-8}.

8.2.2 JNH class

The JNH family of problems has originally been contributed by John Hooker[1] and can also obtained from SATLIB[2]

Jnh belong to the class of randomly generated problems and they follow the constant density model. For an instance with n variables and k clauses, clauses are generated by including a variable with a fixed probability p, and then negating the variable with probability 0.5. Because formulas generated in this way may contain empty clauses or unit clauses. Hence, empty clauses and unit clauses are rejected in the generation process. The resulting clauses are of different length. i.e. they don't contain the same number of literals, hence they belong to the class P-SAT(Bert Selman et al., 1996). Jnh family of problems contain 50 problem instances, of which 10 have 100 variables and 900

1T. Jerome Holleran Professor of Business Ethics and Social Responsibility
Professor of Operations Research, Graduate School of Industrial Administration
Carnegie Mellon University, Pittsburgh, PA 15213 USA.

[2] www.satlib.com

clauses, others have 100 variables and 800 clauses. Only 16 instances are satisfiable. However, according to (Stützle et al., 2001) these instances are no longer challenging for the advanced max-sat algorithms.

8.2.3 Controlled Backbone size problems

As we described previously, the backbone of the problem is the set of variables that have the same truth value in all solutions of a given problem instance and the backbone size of a problem is the number of such variables.

We performed tests on several instances with Controlled backbone size to investigate the effect of that parameter on the performance of various optimization algorithms, the problems were obtained fro SATLIB and were generated as follows:

1. Generate m random k-clauses. For each clause:
    - Select at random k distinct variables from the set of n.
    - Negate each with probability 1/2. These form the literals of the clause.
2. If the instance is unsatisfiable, restart the generation process.
3. Determine the backbone size. If it is not equal to b, restart the generation process.

We tested on 10 problem instances with fixed clause length equal to 3, the number of variables was 100, the number of clauses 423 (in order to get clauses to variables ratio of 4.23, which is according to (Bert Selman et al., 1996)is the hardest region of SAT problems) and the backbone size was 10 and in another set of 10 problems with backbone size equal to 90.

8.2.4 Large Random (LRAN)

This is another class of problems by Bert Selmen and available at the SATLIB. These are randomly generated instances with large sizes, we tested on one of these instances that are 2500 variable and 8000 clause.

8.3 WALKSAT

We tested our PSO methods against the famous Walksat algorithm. As described in 4, Walksat was originally described by (by Selman, B., Kautz, H., and Cohen, B. 1994). Walksat is originally an algorithm for solving the satisfiability problem, SAT. We used a customized implementation, WSATOIP, customized for optimization problems. The program was written by Joachim Paul Walser, and is available for free (http://www.ps.uni-sb.de/~walser/).

Problem instances were converted from CNF format to OIP (Over-constrained linear Pseudo-Boolean) input models. Clauses are of the form

soft: [2] + v371 + v370 <= 0;

where [2] represent the clause weight and soft marks that constraint as a soft constraint so that the search would continue in spite of having clauses unsatisfied. If clauses are hard constraint then they cannot be violated.

Walksat parameters

There are a number of walksat parameters; they are described along with their default values.

Option	Default	Description
-p <prob>	0.010	probability of random move if no improving move is possible
-h <prob>	0.900	probability of selecting a hard vs. soft unsatisfied clause
-z <prob>	0.500	probability of initializing a variable with zero

-m <max> 50K maxflips: number of flips in a try (-1 means infinity)

-r <retries> 1 maxtries: number of tries (-1 means infinity)

-g on turn off history mechanism

-t <size> 1 select tabu memory of <size> flips

-s <seed> 0 initial seed for srandom()

-o <value> none optimal objective value (for termination)

-i <step> 10K interval for reporting progress

-q on (quiet) suppress run-time information from being printed

-d on don't print the variable assignment

-b off display results for benchmarking

-T <dist> 2 maximal distance of triggering FD variables

-L off call CPLEX on linear relaxation for LP bounding

-I off use LP solution for initialization

-w <file.lp> none save problem in CPLEX .lp format to file

(for equivalence the input system must be confined)

-AMPL entry point for interfacing with AMPL

8.4 Large Instance Generator

We developed a weighted max-sat problem generator. This is a random instance generator for max-sat instances with any number of variables and clauses. Clauses can also be of variable or fixed size. The generator generates unique clauses, i.e. no repetitions. Moreover, no variable appear twice in the same clause.

The pseudo code of the generator is as follows

```
Loop For clauses number
Do
Loop for clause length
    Do
        Choose a variable at random
    While(variable exists already in clause before)
While(clause is the same as or subset from previously generated clause)
End loop
```

Figure 20 SAT Generator Pseudocode

9. EXPERIMENTAL RESULTS

9.1 Introduction

This chapter contains the results of experiments done on the famous benchmark problems that were introduced in the previous chapter. Many of these experiments were solved with a number of the proposed PSO methods and approaches. These are mainly pure PSO, PSO with instinct, hypercube neighborhood, PSO with local search, dynamic clauses weights and finally fuzzy PSO. Result analysis and comparisons are also conducted.

The organization of this chapter goes as follows, we start with applying pure PSO algorithm to the weighted Max-Sate problem and show the results obtained on the jnh class of problems. In section 10.3 we then describe the results achieved with instinct-driven particles. Again the experiments show a variation of PSO where instinct-driven particles are introduced as well as a hypercube neighborhood structure. The experiments were conducted on the jnh class.

Section 10.4 illustrates the experiments performed with PSO model with dynamic weights. As clauses remain unsatisfied, their weights start to increase to force particles to give attention to these clauses. The following section, section 10.5, shows the introduction of Local Search as a step performed by swarm particles. We also used static weights. We expanded our benchmark to include encoded graph coloring instances as well as controlled backbone size problems with large and small backbone to show the effect of the instance backbone on the problem difficulty in both Walksat and that PSO approach.

Section 10.6 describes the fuzzy PSO result. The section also contains comparison between Fuzzy PSO, as the comprehensive approach that we have reached and achieved best results, against the other PSO approaches that we have been developing and testing previously, this section also introduces large instance sizes and new randomly generated problems. The section is divided into 5

subsections each showing the results obtained on one problem instance using various fuzzy and non-fuzzy approaches.

9.2 Original PSO for Max-Sat

In this experiment we applied the original PSO algorithm as described in Chapter 6.10. The algorithm was applied on the JNH class of problems. Convergence measurement is achieving 100 iterations with no change in population. PSO algorithm was run a number of times, 100 times, and we calculated the best, average and standard deviation of the 100 results obtained.

We used the standard PSO parameters

- $\phi_1 = \phi_2 = 2$, which is considered the typical value for those parameters
- gbest neighborhood topology
- swarm size 256 paricles

Results obtained on each test problem in each of the 100 runs are shown in table A.1 in appendix. Following is a table describing best, average and standard deviation for each problem.

	Jnh	Jnh 302	Jnh 303	Jnh 304	Jnh 305	Jnh 306	Jnh 307	Jnh 308	Jnh 309	Jnh 310
BEST	443	2024	1289	1550	2099	979	1917	825	1316	1739
AVER	711.20	681.209	718.98	801.79	756.095	704.600	776.405	892.22	803.200	708.919
ST.DE	1741.4	3298.54	2736.04	3260.6	3918.4	2431.97	3272.37	2930.4	2801.59	3525.51
TIME	39271.	38445.9	41493.4	40178.	40973.2	39463.0	38806.1	40423.	43915.8	37773.3

Table 1 Summary of the results obtained with Original PSO on jnh class

Walksat

We now show the results achieved by walksat algorithm on the same jnh class of problems along with the optimal values for each problem. We also show the results achieved by GRASP algorithm described in chapter

GRASP results obtained from (Resende, Pitsoilis, & Pardalos, 1996)

Walksat results obtained with the parameters max-tries set to 100, the rest left with its default values.

Optimal	Walksat	GRASP
0	0	184
395	395	606
351	351	610
321	321	640
742	884	1351
16	16	196
540	540	695
130	130	632
276	276	505
463	463	572

Table 2 Optimal VS results of walksat and GRASP for jnh class

9.2.1 Analysis

The results shown in this section shows that pure PSO is a good optimization engine. It proves that it can be used to find close solutions to the max-sat problem. However, it is not competitive to other well-established algorithms. Local search algorithms like walksat have been extensively used to solve optimization problems and the results obtained show that walksat achieves better performance that PSO. This is what motivates us to enhance the performance of PSO.

9.3 PSO with Instinct-Driven Particles and Hypercube Neighborhood for Max-Sat

These experiments test the PSO engine with instinct-driven particles and hypercube neighborhood instead of gbest model. For the instinct-driven PSO, we had to change some parameters, especially the coefficients ϕ_1 and ϕ_2 because now we added a new term to the velocity equation so we don't want the velocity value to reach Vmax. We decreased the values to 0.6 for ϕ_1 and 1.1 for ϕ_2. The size of he swarm was 64 particles.

Results obtained on problems jnh301-305 are shown in table A.2 in appendix. the following is a summary of best and average result obtained for each problem.

	Jnh301	Jnh302	Jnh303	Jnh304	Jnh305
BEST	64	395	500	321	742
AVERAGE	181.4	820	984	1059	1923

Table 3 summary of results of PSO with instinct and hypercube for jnh class

To prove that the results obtained are certainly due to the instinct function and not due to another change in parameters, we re-did the experiments on the original PSO using the same parameters values that we used with the modified PSO.

Table A.3 in appendix is a table showing the result of Original PSO (no instinct) with $\phi_1 = 1.1, \phi_2 = 0.6$ for jnh301. The experiment shows a worse performance, minimum achieved was 1651. This proves that the change in parameters was not the reason behind the enhancement in the instinct PSO experiments.

To prove that the improvement was not only due to hypercube structure only, we tried the original PSO model with hypercube neighborhood; the results of 100 runs are mentioned in the table A.4 in appendix. Best achieved was 2567.

9.3.1 Analysis

It can be concluded from the comparison between results obtained with pure PSO vs results obtained with instinct-driven and hypercube that the later approach enhances the performance to a great extend. The best achieved with pure PSO for jnh301 for example was 443 and with the instinct and hypercube PSO the minimum was 64. From a quick look at the tables describing the best and average for the 2 methods we can notice the enhancement obtained.

9.4 PSO with Instinct-Driven Particles and Dynamic Weights

This is the results obtained on the JNH class of problem using the approach with hypercube neighborhood, instinct and dynamic weights where the parameters are the same as previous experiment. Each time a clause is not satisfied, its weight increases by a factor equal to 1.13 the original weight.

Table A.5 in appendix lists the results obtained on the instances jnh301-310 for 100 runs. The following table describes the best and average obtained for each problem. Average time per run is 1 minute.

jnh301	Jnh302	Jnh303	Jnh304	Jnh305	Jnh306	Jnh307	Jnh308	Jnh309	Jnh310
12	395	516	352	1079	16	580	286	276	651
183.28	838.13	754.96	835.57	1496.6	359.13	789.86	640.16	484.62	1044.1

Table 4 Summary of results of PSO with instinct, hypercube and dynamic weights on jnh class

The following graph shows the performance of PSO for the run that achieved the optimal result (12)

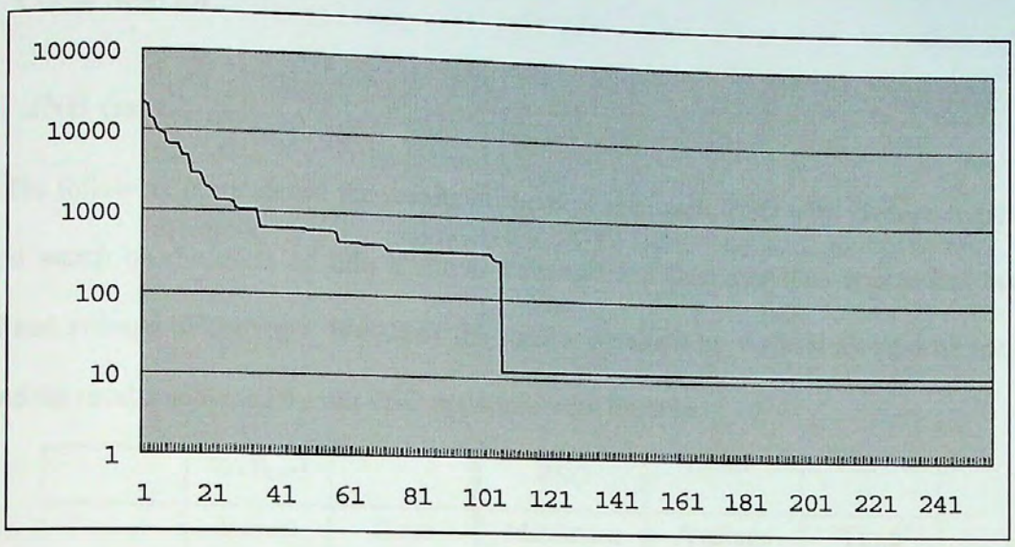

Figure 21 behavior of PSO on Jnh301 problem

9.4.1 Analysis

These experiments show that dynamic weight helps PSO achieve better performance, for example in jnh301, we were able to obtain a better result (12). Dynamic weight helps the engine to overcome the local minima problem as the search space changes dynamically to help particles escape local minima and explore more neighborhoods. However, as mentioned before, one should be careful when using dynamic weights as the performance depends on the problem. With larger sizes, performance can get worse.

9.5 PSO with Instinct-Driven Particles, Hypercube Neighborhood and Local Search

9.5.1 JNH class

The following table shows the results of the new approach, PSO with instinct, hypercube and local search on the class of Jnh, it shows enhancement over previous approaches both in optimal and average of the runs. We show the results obtained by walksat along with the time taken and the results achieved by our PSO approach with its time.

	Walksat		PSO		
	Result	Time	Minimum	Average	Time
Jnh301	0	0	0	59.64	31
Jnh302	395	114	395	557	42
Jnh303	351	113	351	488	32
Jnh304	321	115	321	446	32
Jnh305	884	118	742	1081	40
Jnh306	16	127	16	31	34
Jnh307	540	115	540	628	37
Jnh308	130	115	190	285	33
Jnh309	276	119	276	278	30
Jnh310	463	118	463	595	41

Table 5 results of PSO engine with instinct, hypercube and local search VS walksat for jnh class

9.5.2 Graph Coloring

We show results obtained with PSO with instinct, hypercube and local search on the class of encoded Graph coloring. We chose this set because references show that it is difficult for

walksat to achieve good results in them, we assigned weights randomly to these problems instances. The shaded cells show the problems in which PSO achieved better results than walksat.

WALSAT			PSO		
SOL	Time		BEST	AVERAGE	TIME
45	40		21	132	52
32	41		43	147	52
51	40		31	159	51
45	40		13	137	53
35	40		33	116	50
28	40		63	168	53
50	41		35	141	54
37	40		43	151	53
16	40		39	154	53
27	40		69	186	52

Table 6 Summary of results of PSO with instinct, Hypercube and Local Search VS Walksat for Graph Coloring Problem.

9.5.3 Controlled backbone

Large backbone instances, then small backbone size, it is clear that problems with smaller backbone are easier to solve than the large backbone instances. Small backbone problems were solved in less walksat moves, they were also solved in less PSO time.

Instance name	Best (WALKSAT)	Number of moves (WALKSAT)	Best (PSO)	Average value	Average time
CBS_k3_n100_m423_b90_450	0	43579	0	4.9	9
CBS_k3_n100_m423_b90_451	0	700249	0	11.74	8
CBS_k3_n100_m423_b90_452	0	39261	0	16.6	9.613
CBS_k3_n100_m423_b90_453	0	5605	0	3.29	5.802
CBS_k3_n100_m423_b90_454	0	18506	0	3.22	6.693
CBS_k3_n100_m423_b90_455	0	1246	0	2.6	6.746
CBS_k3_n100_m423_b90_456	0	4899	0	0	2.391
CBS_k3_n100_m423_b90_457	0	5434	0	5.59	7.131
CBS_k3_n100_m423_b90_458	0	105016	0	5.81	10.314
CBS_k3_n100_m423_b90_459	0	61715	0	3.42	7.463
CBS_k3_n100_m423_b10_450	0	226	0	0	1
CBS_k3_n100_m423_b10_451	0	11903	0	0	1.8
CBS_k3_n100_m423_b10_452	0	472	0	0	0.93
CBS_k3_n100_m423_b10_453	0	1646	0	0.16	2.35
CBS_k3_n100_m423_b10_454	0	1152	0	0.65	3.7
CBS_k3_n100_m423_b10_455	0	4884	0	0	1.6
CBS_k3_n100_m423_b10_456	0	1424	0	0	1.89
CBS_k3_n100_m423_b10_457	0	1155	0	0	1.56
CBS_k3_n100_m423_b10_458	0	1155	0	0.05	2.26
CBS_k3_n100_m423_b10_459	0	3897	0	0	1.514

Table 7 Summary of the results obtained by PSO with instinct, hypercube and local search VS Walksat for controlled backbone problems

9.5.4 Analysis

There are two points to note from the previous results. First that hybridizing PSO with local search leads to better performance than PSO only or local search only. For example, on the 1rst, 3rd, 4th and 5th graph coloring problems, the result of the hybrid PSO and local search is better than the results of walksat itself as a local search algorithm. And it is obvious from the

results of the jnh class that the performance of hybrid PSO with local search is better than the results of PSO only.

The second thing to be notes is the effect of the instance backbone on the performance of both local search and hybrid PSO. They both follow the same pattern and small backbone problems were solved in less PSO time and less walksat steps.

9.6 Fuzzy PSO

9.6.1 Fuzzy PSO Results

The PSO model, as explained earlier, incorporates two important parameters, the number of neighbors to be considered and the k in the Cauchy function equation that determines how factors for each neighbor will be calculated. For example, assume we have 3 neighbors and their ranking according to fitness is as follows,

1rst neighbor → BEST

2nd neighbor → 0.75 of BEST

3rd neighbor → 0.5 of BEST

Assume the value of k was 1.0 the factors for these neighbors will be as follows,

1st neighbor = 1.0

2nd neighbor = 0.94

3rd neighbor = 0.8

However, if the value of k was 2.0 the factors will be as follows

1st neighbor = 1.0

2nd neighbor = 0.94

3rd neighbor = 0.8

We conducted 5 basic experiments to show the effect of adding fuzzy approach to PSO. We compare the results of the following approaches on different sets of data

- Basic PSO with Gbest Model.
- Basic PSO replacing Gbest with fuzzy neighbors
- Full PSO
- Full PSO with fuzzy neighbors.

walksat to achieve good results in them, we assigned weights randomly to these problems instances. The shaded cells show the problems in which PSO achieved better results than walksat.

WALSAT			PSO		
SOL	Time		BEST	AVERAGE	TIME
45	40		21	132	52
32	41		43	147	52
51	40		31	159	51
45	40		13	137	53
35	40		33	116	50
28	40		63	168	53
50	41		35	141	54
37	40		43	151	53
16	40		39	154	53
27	40		69	186	52

Table 6 Summary of results of PSO with instinct, Hypercube and Local Search VS Walksat for Graph Coloring Problem.

9.5.3 Controlled backbone

Large backbone instances, then small backbone size, it is clear that problems with smaller backbone are easier to solve than the large backbone instances. Small backbone problems were solved in less walksat moves, they were also solved in less PSO time.

Instance name	Best (WALKSAT)	Number of moves (WALKSAT)	Best (PSO)	Average value	Average time
CBS_k3_n100_m423_b90_450	0	43579	0	4.9	9
CBS_k3_n100_m423_b90_451	0	700249	0	11.74	8
CBS_k3_n100_m423_b90_452	0	39261	0	16.6	9.613
CBS_k3_n100_m423_b90_453	0	5605	0	3.29	5.802
CBS_k3_n100_m423_b90_454	0	18506	0	3.22	6.693
CBS_k3_n100_m423_b90_455	0	1246	0	2.6	6.746
CBS_k3_n100_m423_b90_456	0	4899	0	0	2.391
CBS_k3_n100_m423_b90_457	0	5434	0	5.59	7.131
CBS_k3_n100_m423_b90_458	0	105016	0	5.81	10.314
CBS_k3_n100_m423_b90_459	0	61715	0	3.42	7.463
CBS_k3_n100_m423_b10_450	0	226	0	0	1
CBS_k3_n100_m423_b10_451	0	11903	0	0	1.8
CBS_k3_n100_m423_b10_452	0	472	0	0	0.93
CBS_k3_n100_m423_b10_453	0	1646	0	0.16	2.35
CBS_k3_n100_m423_b10_454	0	1152	0	0.65	3.7
CBS_k3_n100_m423_b10_455	0	4884	0	0	1.6
CBS_k3_n100_m423_b10_456	0	1424	0	0	1.89
CBS_k3_n100_m423_b10_457	0	1155	0	0	1.56
CBS_k3_n100_m423_b10_458	0	1155	0	0.05	2.26
CBS_k3_n100_m423_b10_459	0	3897	0	0	1.514

Table 7 Summary of the results obtained by PSO with instinct, hypercube and local search VS Walksat for controlled backbone problems

9.5.4 Analysis

There are two points to note from the previous results. First that hybridizing PSO with local search leads to better performance than PSO only or local search only. For example, on the 1rst, 3rd, 4th and 5th graph coloring problems, the result of the hybrid PSO and local search is better than the results of walksat itself as a local search algorithm. And it is obvious from the

results of the jnh class that the performance of hybrid PSO with local search is better than the results of PSO only.

The second thing to be notes is the effect of the instance backbone on the performance of both local search and hybrid PSO. They both follow the same pattern and small backbone problems were solved in less PSO time and less walksat steps.

9.6 Fuzzy PSO

9.6.1 Fuzzy PSO Results

The PSO model, as explained earlier, incorporates two important parameters, the number of neighbors to be considered and the k in the Cauchy function equation that determines how factors for each neighbor will be calculated. For example, assume we have 3 neighbors and their ranking according to fitness is as follows,

1rst neighbor → BEST

2nd neighbor → 0.75 of BEST

3rd neighbor → 0.5 of BEST

Assume the value of k was 1.0 the factors for these neighbors will be as follows,

1st neighbor = 1.0

2nd neighbor = 0.94

3rd neighbor = 0.8

However, if the value of k was 2.0 the factors will be as follows

1st neighbor = 1.0

2nd neighbor = 0.94

3rd neighbor = 0.8

We conducted 5 basic experiments to show the effect of adding fuzzy approach to PSO. We compare the results of the following approaches on different sets of data

- Basic PSO with Gbest Model.
- Basic PSO replacing Gbest with fuzzy neighbors
- Full PSO
- Full PSO with fuzzy neighbors.

Aside from comparing those approaches with each other, we compare them against WALKSAT as a famous Local search algorithm.

We describe the basic parameters that were used in each approach before listing the results on each test problem.

Approach	ϕ_1	ϕ_2	Swarm Size
Gbest	2.0	2.0	256
Gbest, Fuzzy	2.0	2.0	256
Full PSO engine	0.6	1.1	64
Full PSO, Fuzzy	0.6	0.5	64

Table 8 Parameters used for different PSO approaches

9.6.2 Jnh Class

9.6.2.1 Gbest

We remind you of the results of pure PSO engine on the JNH class. Those experiments were also described earlier.

	Jnh	Jnh 302	Jnh 303	Jnh 304	Jnh 305	Jnh 306	Jnh 307	Jnh 308	Jnh 309	Jnh 310
BEST	443	2024	1289	1550	2099	979	1917	825	1316	1739
AVER	711.20	681.209	718.98	801.79	756.095	704.600	776.405	892.22	803.200	708.919
ST.DE	1741.4	3298.54	2736.04	3260.6	3918.4	2431.97	3272.37	2930.4	2801.59	3525.51
TIME	39271.	38445.9	41493.4	40178.	40973.2	39463.0	38806.1	40423.	43915.8	37773.3

Table 9 Summary of results for Original PSO for jnh class

9.6.2.2 Fuzzy Gbest

We show the results of the basic PSO engine just replacing the gbest with a number of neighbors each contributes with a certain factor calculated based on the Cauchy distribution. The table describes the result achieved for each value of k and for different neighborhood sizes.

k	Neighborhood size	Jnh301		Jnh302		Jnh303		Jnh304		Jnh305	
		Best	Average	Best	Average	Best	Average	Best	Average	Best	Average
1.2	3	99	717.899963	745	1958.45	740	1627.65	816	1971.9	1714	2543.15
1.4	3	281	708.349915	934	2100.15	610	1727.9	877	1858.8	1646	2371.15
1.6	3	257	907.450012	900	2030.95	610	1474.6	1055	1985.35	1516	2631.2
1.8	3	264	761.450012	1552	2293.4	644	1893.55	944	1761.6	1313	2537.45
2	3	100	717.100098	819	2031.95	1204	1754.8	954	1780	1922	2863.85
1.2	4	105	756.149963	549	1755.9	852	1512.7	1018	1809.4	1477	2452.65
1.4	4	259	846.600098	640	1789.7	1049	1730.85	813	1681.7	1653	2519.85
1.6	4	245	706.899963	888	1861.5	938	1539.05	795	1852.4	1826	2499.8
1.8	4	168	714.100037	867	1822.65	994	1356.8	980	1942.85	1351	2306.9
2	4	173	648.950073	515	1989.35	697	1843.35	860	1754.6	1661	2608.95
1.2	5	247	784.350037	911	1749.65	954	1357	621	1820.7	1666	2340.3
1.4	5	105	835.149963	867	1703.4	897	1346.6	658	1770.65	1422	2320.1
1.6	5	266	690.650085	395	1754.65	857	1517.05	714	1633.4	1824	2556.05
1.8	5	170	619.099976	900	1967.1	900	1432.65	674	1729.3	1186	2485.35
2	5	105	606.150085	912	1902.25	850	1512.15	935	1686.6	1717	2376.6
BEST		99	606.150085	395	1703.4	610	1346.6	621	1633.4	1186	2306.9

Table 10 Summary of Fuzzy Gbest PSO for jnh class 301-305

k	Neighbor-hood Size	Jnh306		Jnh307		Jnh308		Jnh309		Jnh310	
1.2	3	562	1260.55	810	1767	847	1495.35	574	1334.6	1360	2144.45
1.4	3	703	1549.05	771	1656.9	636	1433.55	638	1462	1457	2381
1.6	3	377	1271	1176	1755.8	609	1578	637	1342.65	890	2385.5
1.8	3	212	1153.35	854	1677	599	1359.15	690	1339.85	944	2310.2
2	3	815	1359.25	869	2056.35	759	1550	567	1561.7	1131	2155.7
1.2	4	212	1273.95	1046	1692.7	619	1449.2	276	1361.75	988	2118.7
1.4	4	337	1147	780	1624.7	581	1202.1	631	1196.65	1242	2201.8
1.6	4	225	1094.65	991	1617.55	598	1380.2	548	1262.8	1108	2211.25
1.8	4	262	1266.25	818	1773.45	613	1506.65	600	1338.3	1639	2240.3
2	4	265	1145.3	663	1799.05	407	1323.5	407	1334.5	1135	2008
1.2	5	532	1130.3	1073	1814.5	433	1095.1	574	1281.8	826	2010
1.4	5	307	1050.05	580	1595.4	631	1284.9	518	1149.75	905	1826.9
1.6	5	339	941.4	580	1762.3	404	1108.3	571	1242.4	1276	2057.55
1.8	5	357	1071.65	772	1750.4	521	1199.45	560	1107	918	2016.05
2	5	586	1213.95	663	1672.7	667	1570.45	581	1258.3	1074	1954.1
		212	941.4	580	1595.4	404	1095.1	276	1107	826	1826.9

Table 11 Summary of Fuzzy Gbest PSO for jnh class 306-310

Analysis

Average time taken for each run = 6750.000000 ms

The following is a comparison between the results obtained with the Gbest PSO versus the gbest with fuzzy neighborhood. It is obvious that in all problems the fuzzy approach achieved a better result. The maximum enhancement was in jnh302 where the fuzzy PSO improved by 80%. Fuzzy PSO was able to reach the minimum for 2 problems, which are jnh302 and jnh309.

Gbest PSO	Fuzzy PSO	% Improvement
443	99	77.6 %
2024	395	80.48 %
1289	610	52.67 %
1550	621	59.93 %
2099	1186	43.4 %
979	212	78.34 %
1917	580	69.74 %
825	404	51.03 %
1316	276	79.02 %
1739	826	52.50 %

Table 12 comparison of Pure PSO VS Fuzzy PSO for jnh class

9.6.2.3 PSO with Instinct-Driven Particles, Hypercube Neighborhood and Local Search

We remind you of the results described earlier for the PSO engine with instinct, hypercube and local search on the jnh class of problems.

	Minimum	Average
Jnh301	0	59.64

Jnh302	395	557
Jnh303	351	488
Jnh304	321	446
Jnh305	742	1081
Jnh306	16	31
Jnh307	540	628
Jnh308	190	285
Jnh309	276	278
Jnh310	463	595

Table 13 Results of PSO with Instinct, Hypercube neighborhood and Local Search on jnh class

As we illustrated earlier, full PSO engine achieved optimal in 9 of the 10 jnh problems, which is a good improvement. The only missed problem was jnh308, where the optimal was 130 and we achieved 190.

9.6.2.4 Fuzzy PSO with Instinct-Driven Particles, Hypercube Neighborhood and Local Search

Here we show the results of the same approach as the previous experiment, PSO with instinct-driven particles, hypercube neighborhood and Local search, after adding the fuzzy neighborhood. i.e. we rank the neighbors selected in the hypercube according to their fitness and consider a number of them, each with a certain factor calculated based on the Cauchy function. The following table describes the results obtained with varying neighborhood size and the k parameter in the Cauchy function.

k	Neighbor-hood Size	Jnh301		Jnh302		Jnh303	
		Best	Average	Best	Average	Best	Average
1.2	3	0	60.799999	503	727.65	351	507.75
1.4	3	64	64.599998	395	705.15	516	519.45

1.6	3	64	64.599998	395	718.7001	351	507.75
1.8	3	64	64.199997	395	725.6	351	515.45
2	3	64	64	395	707.4	431	511.75
1.2	4	64	66.800003	395	719.95	516	516
1.4	4	64	64.599991	515	738.35	516	516
1.6	4	64	65.200005	515	747.65	516	516
1.8	4	48	65.599998	515	738.7	351	507.75
2	4	64	64.600006	395	746.3	499	515.15
1.2	5	64	64.599991	395	709.15	516	516
1.4	5	64	70.650009	395	671.5999	516	516
1.6	5	64	64.400009	395	718.85	516	516
1.8	5	64	70.000008	575	736.1	516	520.7001
2	5	64	65	503	728.05	516	516
BEST		0	60.799999	395	671.5999	351	507.75

Table 14 Fuzzy PSO with instinct, hypercube and local search results for jnh 301-303

k	Neighbor-hood Size	Jnh304		Jnh305		Jnh306	
		Best	Average	Best	Average	Best	Average
1.2	3	427	509.05	884	1199.3	16	42.599998
1.4	3	427	543.65	1133	1232.4	16	50.150002
1.6	3	427	512.95	1032	1219.95	16	55.299999
1.8	3	321	484.1	1067	1204.35	16	53.499996
2	3	332	477	1039	1216.15	16	46.25
1.2	4	427	610.85	1133	1229.35	16	74.949997
1.4	4	427	530.8499	1067	1225.5	16	65.700005
1.6	4	427	541.3	1133	1273	16	77.099998
1.8	4	427	506.4	1133	1272.95	16	64.950005
2	4	427	557.2	1133	1233.95	16	76.949997
1.2	5	427	572.65	1133	1241.7	16	70.849998

1.4	5	427	580.85	884	1241	16	69.099998
1.6	5	427	609.35	884	1240.75	16	79.75
1.8	5	427	615.2	884	1214.4	16	69.599998
2	5	427	573.75	971	1228.8	16	67.500008
BEST		321	477	884	1199.3	16	42.599998

Table 15 Fuzzy PSO with instinct, hypercube and local search results for jnh 304-306

k	Neighbor-hood Size	Jnh307		Jnh308		Jnh309		Jnh310	
		Best	Average	Best	Average	Average	Average	Best	Average
1.2	3	540	708.85	286	290.35	276	276	501	655.55
1.4	3	580	748.45	286	308.65	276	282.4	536	758.3
1.6	3	540	719.2	286	308.35	276	288.8	533	697.15
1.8	3	540	716.7501	286	324.25	276	286.75	545	707.05
2	3	540	723.35	286	309.45	276	288.7	545	718.75
1.2	4	540	746.75	286	320.75	276	282.4	501	762.95
1.4	4	540	684.75	286	304	276	282.4	651	722.4
1.6	4	540	686.1	286	299.35	276	288.8	501	733.65
1.8	4	655	775.55	286	305	276	276	501	722.05
2	4	580	780.4999	286	295	276	276	501	716.05
1.2	5	540	746.8	286	328.95	276	276	501	679.5999
1.4	5	540	743.3	286	369	276	288.8	501	773.85
1.6	5	540	729.9	286	344.25	276	295.2	651	760.55
1.8	5	540	754.8	286	332.85	276	282.4	509	728.45
2	5	540	723.7	286	338.75	276	295.2	509	761.3
BEST		540	684.75	286	290.35	276	276	501	655.55

Table 16 Fuzzy PSO with instinct, hypercube and local search results for jnh 307-310

Here, we summarize the results obtained for the 10 problems.

jnh301	Jnh302	Jnh303	Jnh304	Jnh305	Jnh306	Jnh307	Jnh308	Jnh309	310
0	395	351	321	884	16	540	286	276	501

Table 17 Summary of Fuzzy PSO with instinct, hypercube and local search results for the jnh class

9.6.2.5 Walksat for jnh Class

The following table describes the results obtained by Walksat for the problems jnh301-310 along with time taken for each.

	Walksat result	Time (sec)
Jnh301	0	0
Jnh302	395	114
Jnh303	351	113
Jnh304	321	115
Jnh305	884	118
Jnh306	16	127
Jnh307	540	115
Jnh308	130	115
Jnh309	276	119
Jnh310	463	118

Table 18 walksat results on the jnh class

Here walksat didn't achieve the optimal in one instance, jnh305 where the optimal is 742.

Analysis

The results show an improvement from PSO to Fuzzy PSO that reached 80% for the JNH302 instance. Further improvement was achieved using the full PSO approach, incorporating the instinct and local search methods to achieve optimal in all JNH problems except one. Not much improvement was achieved with full fuzzy PSO because the number of runs for the fuzzy model was restricted to 10 runs only (not 100 runs like the other approaches) because the number of experiments was very big.

9.6.3 Testing with LRAN Problem Instance

The four experiments for different PSO approaches were conducted on a large SAT instance obtained from SATLIB (LRAN) containing 2000 variable and 8500 clauses.

9.6.3.1 Gbest

The following table shows the results of the pure PSO algorithm with global best neighbor. We show the result of 10 runs of the program along with the best and average result obtained and average time taken per runs

Runs 1-5	Runs 6-10
38519	38530
38866	38675
38758	39458
38374	37750
39212	38959

Best	37750
Average	38710.1
Average Time	182900

Table 19 Original PSO results and Summary for LRAN

9.6.3.2 Fuzzy Gbest

The following table shows the results of PSO algorithm replacing the gbest neighbor with a set of variable size neighbors. Each of which contributes with a factor. Which we call Fuzzy PSO model. Experiments are done varying the k factor of the Cauchy function and the set of neighbors that have an effect, or what we can call neighborhood size, n. each experiment was conducted 5 times and we record the best and average value obtained.

Neighborhood size	k	Best	Average
2	1	34162	34552.39844
2	1.2	33446	34768.20313
2	1.4	34342	35006.80078
2	1.6	34271	35045.60156
2	1.8	34471	34896.60156
2	2	33952	34618.19922
2	2.2	34388	35057.80078
2	2.4	34419	34668.80078
2	2.6	33566	34550
2	2.8	34482	34887
2	3	34300	35694.59766
3	1	31796	32979.39844
3	1.2	31828	33417.60156
3	1.4	31611	32814.39844
3	1.6	32518	33018
3	1.8	32082	33180.80078
3	2	32706	33172
3	2.2	32006	33395.59766
3	2.4	32340	33575.39844
3	2.6	32744	33017.80078
3	2.8	31754	32800
3	3	32052	33065.39844
4	1	32586	33367.80078
4	1.2	32841	33644
4	1.4	32577	33010
4	1.6	32446	32864
4	1.8	32156	32665.59961
4	2	32937	33270.19922

4	2.2	32618	33265.60156
4	2.4	31866	32986
4	2.6	32751	33660.19922
4	2.8	32512	33010.19922
4	3	32333	33466
5	1	33641	34261.19922
5	1.2	33463	34426.60156
5	1.4	33920	34693.60156
5	1.6	33302	34201.60156
5	1.8	34072	34549.19922
5	2	33208	33640.40234
5	2.2	33471	34005
5	2.4	34024	34609.39844
5	2.6	33371	34179
5	2.8	33762	34134
5	3	33453	34595.59766

Table 20 Fuzzy Gbest PSO results for LRAN

Analysis

Best Result obtained from k = 1.4 and size = 3 which is 31611.

Average time per run = 238440 MS

It is obvious that this result obtained, 31611, is better than the minimum result obtained with pure PSO with gbest model which is 37750

9.6.3.3 PSO with Instinct-Driven Particles, Hypercube Neighborhood and Local Search

To make good analysis of the results obtained, we show the results of the same test case on the full PSO approach with no fuzzy function. We show the results of running the program 25 times, we show the best and average value obtained along with average time per run.

Runs 1-6	Runs 7-12	Runs 13-18	Runs19-24
1034	1232	1598	1241
884	1146	869	1137
1048	1153	1014	951
1227	1309	1538	955
1058	1079	1000	929
1051	955	1250	894

Best	869
Average	1106
Average Time	852393 MS

Table 21 PSO with instinct, hypercube and local search results and Summary for LRAN

We can see that the best obtained in this method, 869 is far better than the result obtained in the previous two approaches, 31611, 37750. Which implies that the modifications applied to the PSO engine enhances the performance greatly.

9.6.3.4 Fuzzy PSO with Instinct-Driven Particles, Hypercube Neighborhood and Local Search

Here we show the results of the same approach as the previous experiment, after adding the fuzzy neighborhood. i.e. we rank the neighbors selected in the hypercube according to their fitness and consider a number of them, each with a certain factor calculated based on the Cauchy function. The following table describes the results obtained with varying neighborhood size and the k parameter in the Cauchy function.

k	Neighborhood size	Best	Average
1	2	965	1247.800049
1.2	2	844	1066.799927
1.4	2	1046	1250.400024
1.6	2	1185	1262.400024
1.8	2	992	1153
2	2	790	1164
2.2	2	1005	1202.400024
2.4	2	961	1107
2.6	2	1031	1209.800049
2.8	2	1174	1267.199951
3	2	1041	1151.800049
1	3	850	982.199951
1.2	3	906	1070.800049
1.4	3	798	992.999939
1.6	3	801	1023.200012
1.8	3	874	1041
2	3	877	954.200012
2.2	3	915	1109.800049
2.4	3	851	1022.799927
2.6	3	1018	1062
2.8	3	885	1064
3	3	952	1064.200073
1	4	776	980.399963
1.2	4	959	1080.200073
1.4	4	932	1116.199951
1.6	4	902	991
1.8	4	913	1002.999939
2	4	885	977.200012
2.2	4	860	959.400024
2.4	4	656	981
2.6	4	775	1021.599976
2.8	4	969	1012.400024
3	4	994	1086.800049
1	5	825	1036.199951
1.2	5	964	1111.800049
1.4	5	905	1049.600098
1.6	5	859	946.799988
1.8	5	724	878.799988
2	5	899	972.799988
2.2	5	781	924.599976
2.4	5	808	1034.599976
2.6	5	844	1026
2.8	5	850	1014.799988
3	5	897	1006.799988

Table 22 Fuzzy PSO with instinct, hypercube and local search results for LRAN

Analysis

Best Result obtained from k = 2.4 and size = 4 which is 656 which is the best result achieved on this problem. Introducing the fuzzy neighborhood achieved a 24% improvement.

Average time per run = 862141 MS

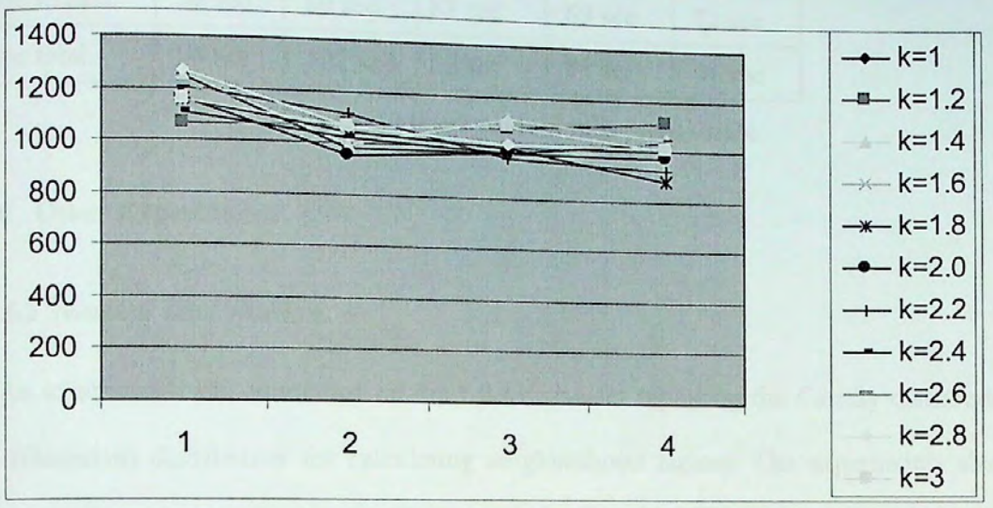

Figure 22 Average Value obtained VS neighborhood for different K

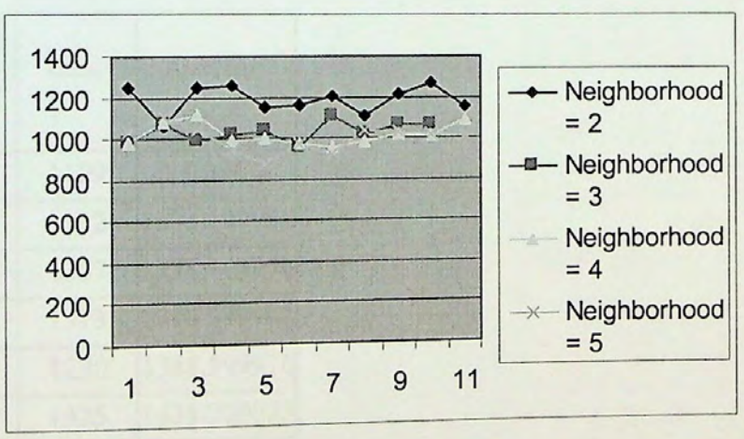

Figure 23 Average Value Obtained VS K for different neighborhood sizes

9.6.3.5 Walksat performance on LRAN Problem Instance

Walksat parameters default values left as is and max-tries was set to 100

Result	1034	980	1147	1984	1558
Moves to best	3046293	3245892	2678249	2276109	3046293
Moves total	5000000	5000000	5000000	5000000	5000000
Time to best	59 sec	80 sec	83 sec	69 sec	72 sec
Time total	96 sec	102 sec	89 sec	91 sec	94 sec

Table 23 Walksat performance on LRAN

9.6.3.6 Other Experiments

9.6.3.6.1 Normal Distribution

An experiment was conducted on the LRAN dataset replacing the Cauchy distribution with Normal (Gaussian) distribution for calculating neighborhood factors. The experiments shows that results achieved using the Cauchy distribution are better, we could reach value of 656 using Cauchy dist. While using Gaussian, the best obtained was 1069

K	Neighbor-hood size	Best	Average
1	2	1199	1376
1.2	2	1132	1373.199951
1.4	2	1307	1378.199951
1.6	2	1313	1398.399902
1.8	2	1230	1388.599976
2	2	1325	1433.200073
1	3	1088	1314.199951
1.2	3	1193	1333
1.4	3	1277	1412.400024
1.6	3	1281	1347
1.8	3	1226	1383.999878
2	3	1308	1373.399902
1	4	1269	1338.799927

1.2	4	1114	1323.599976
1.4	4	1306	1384.199951
1.6	4	1194	1319.200073
1.8	4	1317	1408
2	4	1162	1240
1	5	1251	1382.199951
1.2	5	1288	1422.600098
1.4	5	1198	1336.400024
1.6	5	1079	1324.800049
1.8	5	1120	1332.400024
2	5	1069	1312.800049

Table 24 Fuzzy PSO with instinct-driven particles, hypercube and local search results for LRAN using Gaussian distribution instead of Cauchy

9.6.3.6.2 Performance Enhancement

In order to get better performance, we increased the number of local search rounds performed by the particles at the end of each iteration. In previous experiments, each particle would perform three rounds of local search at the end of each PSO iteration. That number was increased to five rounds. The table below lists the results achieved for different values of k and neighborhood size.

k	Neighborhood Size	Best	Average
1.2	3	640	659
1.6	3	509	659.666687
2	3	683	692.666687
2.4	3	560	657.666626
2.8	3	552	632.666626

1.2	4	624	732.333313
1.6	4	591	658
2	4	638	695.333313
2.4	4	544	603.333374
2.8	4	591	712.666687
1.2	5	556	735.333313
1.6	5	593	631.666687
2	5	722	865
2.4	5	595	663
2.8	5	737	777.333374

Table 25 Fuzzy PSO with instinct-driven particles, hypercube and local search results with increased Local search rounds for LRAN problem instance

Best value achieved was 509 for k = 1.6 and neighborhood = 3.

The results illustrated in the previous table shows good enhancement by increasing the local search rounds. However, the average time taken per run increased due to the time taken for more local search operation performed at each iteration by each particle.

Thus, we gave the same time taken by our Full PSO approach to Walksat by increasing the number of restarts for each run. We got the following result

Result	742	700
Moves to best	46046712	43245892
Moves total	125000000	125000000
Time to best	803sec	798 sec
Time total	2192 sec	2180 sec

Table 26 Walksat performance on LRAN given more time

The result achieved after giving the two approaches equal time shows that PSO (achieving 509) out performs Walksat (achieving 700)

9.6.3.7 Analysis

We want to compare the minimum result obtained by each approach on this problem instance in order to show the quality of each approach and to be able to conclude how different PSO approaches are ranked against each other and against walksat.

1) PSO with instinct-driven particles, hypercube neighborhood and local search with more local search rounds →509
2) Fuzzy PSO with instinct-driven particles, hypercube neighborhood and local search → 656
3) Walksat (125,000,000 moves) →700
4) PSO with instinct-driven particles, hypercube neighborhood and local search →869
5) Walksat (5,000,000 moves) →980
6) Fuzzy Gbest PSO →31611
7) Gbest PSO → 37750

9.6.4 Testing with GEN01 Problem Instance

The four experiments were conducted on a generated SAT instance obtained with the generator described in previous chapter. Instance contains 4000 variable and 32000 clauses. We call this problem GEN01. We show the results of the different PSO approaches along with the results of Walksat.

9.6.4.1 Gbest

The following table shows the results of the pure PSO algorithm with global best neighbor. We show the result of 10 runs of the program along with the best and average result obtained and average time taken per runs

Runs 1-5	Runs 6-10
85531	86655
86344	85010
83378	85588
85392	86918
83150	85646

Best	83150
Average	85361.2
Average Time	1298907 MS

Table 27 Original PSO results and Summary for GEN01 problem.

9.6.4.2 Fuzzy Gbest

The following table shows the results of PSO algorithm replacing the one Gbest neighbor with a set of variable size neighbors. Each of which contributes with a factor. We call this approach Fuzzy Gbest PSO model. Experiments are done varying the k factor of the Cauchy function and the set of neighbors that have an effect, or what we can call neighborhood size. each experiment was conducted 5 times and we record the best and average value obtained.

Neighborhood size	k	Best	Average
3	1.2	75671	77180.39844
3	1.4	74933	76272.60156
3	1.6	76343	77731
3	1.8	77383	78164.20313
3	2	75746	77503.19531
4	1.2	76577	77773.79688
4	1.4	76192	76953.39844
4	1.6	76856	77418.79688
4	1.8	76248	77632.79688
4	2	76376	77333.20313
5	1.2	77618	79156.79688
5	1.4	79436	79927.79688
5	1.6	78761	79244.39844
5	1.8	77398	79158.60156
5	2	76898	78268.39844

Table 28 Fuzzy PSO results for GEN01 problem

Analysis

Best Result obtained from k = 1.4 and size = 3 which is 74933

Average time per run = 1176219 MS

It is obvious that this result obtained, 74933, is better than the minimum result obtained with pure PSO with Gbest model, which is 83150. Other results obtained with this method still shows an improvement over the original Gbest PSO.

9.6.4.3 PSO with Instinct-Driven Particles, Hybercube neighborhood and Local Search

To make good analysis of the results obtained, we show the results of the same test case on the full PSO approach with no fuzzy function. We show the results of running the program 10 times, we show the best and average value obtained along with average time per run.

Runs 1-5	Runs 6-10
3931	2906
4401	3312
3264	3860
4102	3090
4649	3337

Best	2906
Average	3685.2
Average Time	3498765 MS

Table 29 PSO with instinct hypercube and local search results and Summary for GEN01 problem.

We can notice that the result obtained which is 2906 is far better than the minimum achieved in the previous methods, which was 83150 and 74933. This indicates the effect of the modifications to the basic PSO model, which are the instinct, hypercube neighborhood, and particles performing local search.

9.6.4.4 Fuzzy PSO with Instinct-Driven Particles, Hypercube Neighborhood and Local Search

Here we show the results of the same approach as the previous experiment, full PSO, after adding the fuzzy neighborhood. i.e. we rank the neighbors selected in the hypercube according to their fitness and consider a number of them, each with a certain factor calculated based on the Cauchy function. The following table describes the results obtained with varying neighborhood size and the k parameter in the Cauchy function.

Neighborhood	K	Best	Average
3	1.2	949	1099.2
3	1.4	865	1124.4
3	1.6	820	984
3	1.8	880	993.2
3	2	703	1069.25
4	1.2	636	935.5
4	1.4	964	1052
4	1.6	930	1126.75
4	1.8	989	1098.25
4	2	800	865.25
5	1.2	901	1050.2
5	1.4	770	929.5
5	1.6	1015	1141.25
5	1.8	868	1114.75
5	2	974	1198.5

Table 30 Fuzzy PSO with instinct-driven particles, hypercube neighborhood and local search results for GEN01 problem

Analysis

Best Result obtained from k = 1.2 and size = 4 which is 636 which is the best result obtained in the 4 approaches.

Average time per run = 5886815 MS

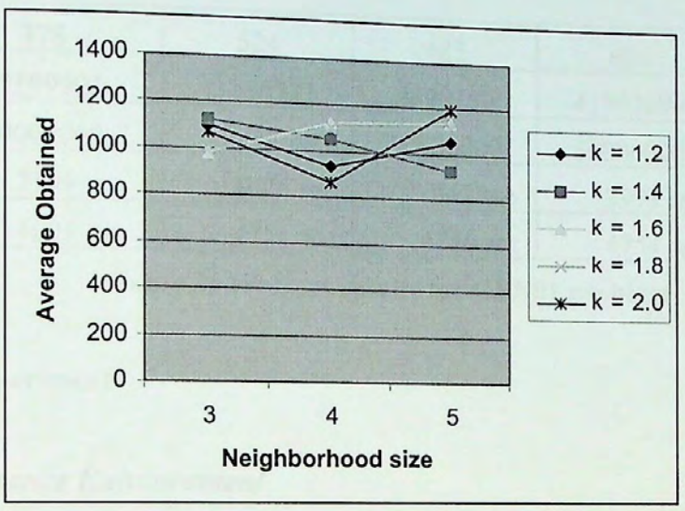

Figure 24 Average Value obtained VS neighborhood for different K

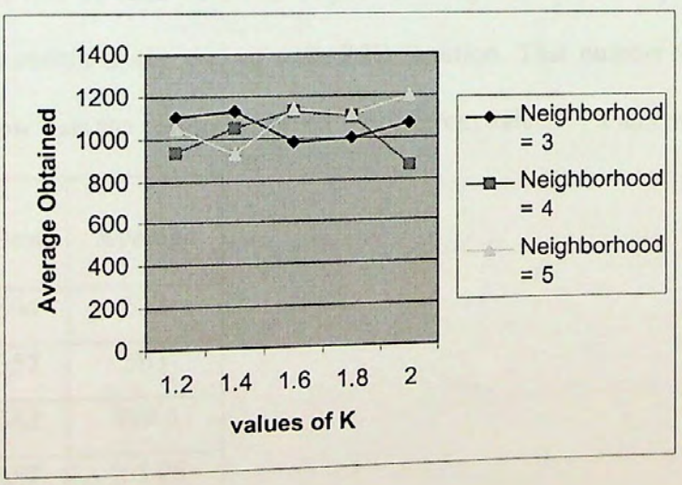

Figure 25 Average Value Obtained VS K for different neighborhood sizes

9.6.4.5 Walksat Performance on GEN01

Here we show the results obtained by running the Walksat algorithm on GEN01 problem for 5 different runs, the best value obtained was 378. This value is better than those obtained by PSO variations.

Result	378	524	485	604	553
Moves to best	23199901	11298701	8898180	41993090	24398468
Moves total	50000000	50000000	50000000	50000000	50000000
Time to best	2199 sec	1076 sec	847 sec	3993 sec	2365 sec
Time total	4693 sec	4751 sec	4746 sec	4754 sec	4796 sec

Table 31 Walksat results for GEN01 problem

9.6.4.6 Other Experiments

9.6.4.6.1 Performance Enhancement

In order to get better performance, we increased the number of local search rounds performed by the particles at the end of each iteration. In previous experiments, each particle would perform three rounds of local search at the end of each PSO iteration. That number was increased to five rounds. The table below lists the results achieved for different values of k and neighborhood size.

k	Neighborhood size	Best	Average
1.2	5	394	532
1.4	5	357	503
1.6	5	382	499.5
1.8	5	387	465.25
2	5	519	566
2.2	5	386	554
2.4	5	318	543.5

2.8	5	320	471.75
1.2	4	323	470.25
1.4	4	269	445.5
1.6	4	246	460
1.8	4	388	522.25
2	4	397	480
2.2	4	286	474
2.4	4	491	540.5
2.6	4	379	497.75
2.8	4	365	469.75
1.2	3	491	534.5
1.4	3	343	452.25
1.6	3	353	442.75
1.8	3	415	505.5
2	3	417	482.25
2.2	3	210	462.25
2.4	3	399	454
2.6	3	200	375
2.8	3	386	522.75

Table 32 Fuzzy PSO with instinct, hypercube and local search results with increased Local search rounds for GEN01 problem instance

Best value achieved was 200 for k = 2.6 and neighborhood = 3.

The results illustrated in the previous table shows good enhancement by increasing the local search rounds. However, the average time taken per run increased due to the time taken for more localsearch operation performed at each iteration by each particle.

Thus, we gave the same time taken by our Full PSO approach to Walksat by increasing the number of restarts for each run. We got the following result

Result	538	485
Moves to best	106347451	87699272
Moves total	125000000	125000000
Time to best	7308 sec	6003sec
Time total	8590 sec	8556 sec

Table 33 Walksat performance on GEN01 problem given more time

The result achieved after giving the two approaches equal time shows that PSO (achieving 200) out performs Walksat (achieving 485)

9.6.4.7 Analysis

We want to compare the minimum result obtained by each approach on this problem instance in order to show the quality of each approach and to be able to conclude how different PSO approaches are ranked against each other and against walksat.

1) Fuzzy PSO with instinct-driven particles, hypercube neighborhood and local search with more local search rounds → 200
2) Walksat →378
3) Walksat (125,000,000 moves) → 485
4) Fuzzy PSO with instinct-driven particles, hypercube neighborhood and local search → 636
5) PSO with instinct-driven particles, hypercube neighborhood and local search →2906
6) Fuzzy Gbest PSO →74933
7) Gbest PSO → 83150

9.6.5 Testing with GEN02 Problem Instance

The four experiments were conducted on a generated SAT instance obtained with the generator described in previous chapter. Instance contains 4000 variable and 32000 clauses. We call this problem instance GEN02. We show the results of the different PSO approaches along with the results of Walksat.

9.6.5.1 Gbest

The following table shows the results of the pure PSO algorithm with global best neighbor. We show the result of 10 runs of the program along with the best and average result obtained and average time taken per runs

Runs 1-5	Runs 6-10
136685	134304
138166	135413
134405	135356
134777	134552
134747	134682

Best	134304
Average	135308.7
Average Time	1897811 MS

Table 34 Original PSO results and Summary for GEN02

9.6.5.2 Fuzzy Gbest

The following table shows the results of PSO algorithm replacing the one Gbest neighbor with a set of variable size neighbors. Each of which contributes with a factor. We call this approach Fuzzy Gbest PSO model. Experiments are done varying the k factor of the Cauchy function and the set of neighbors that have an effect, or what we can call neighborhood size. each experiment was conducted 5 times and we record the best and average value obtained.

Neighborhood Size	K	Best	Average
3	1.2	124301	125483
3	1.4	123644	125332
3	1.6	123219	124469
3	1.8	124080	124873.8
3	2	124340	125916.8
4	1.2	125279	126698.6
4	1.4	124200	125927.2
4	1.6	123312	125353.2
4	1.8	124829	125568.6
4	2	124290	125403.2
5	1.2	126294	127247.8
5	1.4	126751	127144.2
5	1.6	126105	127837.8
5	1.8	126304	128166.2
5	2	125984	127668.6

Table 35 Fuzzy PSO results for GEN02 problem

Analysis

Best Result obtained from k = 2.0 and size = 3 which is 123219

Average time per run = 1761546 MS

It is obvious that this result obtained, 123219, is better than the minimum result obtained with pure PSO with Gbest model, which is 134304

9.6.5.3 PSO with instinct-driven particles, hypercube neighborhood and local search

To make good analysis of the results obtained, we show the results of the same test case on the full PSO approach with no fuzzy function. We show the results of running the program 10 times, we show the best and average value obtained along with average time per run.

Runs 1-5	Runs 6-10
5761	4036
5103	5385
8412	4452
5582	5047
4697	5440

Best	4036
Average	5391.5
Average Time	6217471.1 MS

Table 36 PSO with instinct, hypercube neighborhood and local search results and Summary on GEN02

We can notice that the result obtained which is 4036 is far better than the minimum achieved in the previous methods, which was 134304 and 123219. This indicates the effect of the modifications to the basic PSO model, which are the instinct, hypercube neighborhood, and particles performing local search.

9.6.5.4 Fuzzy PSO with instinct-driven particles, hypercube neighborhood and local search

Here we show the results of the same approach as the previous experiment, full PSO, after adding the fuzzy neighborhood. i.e. we rank the neighbors selected in the hypercube according to their fitness and consider a number of them, each with a certain factor calculated based on the Cauchy function. The following table describes the results obtained with varying neighborhood size and the k parameter in the Cauchy function

Neighbor-hood size	k	Best	Average
3	1.2	2020	2233.25
3	1.4	1846	2195.75
3	1.6	1987	2239.25
3	1.8	2063	2293.5
3	2	1938	2247.75
4	1.2	2067	2236
4	1.4	1947	2242.25
4	1.6	1896	2335.25
4	1.8	1676	2052.75
4	2	2011	2117.75
5	1.2	1649	2174.25
5	1.4	2150	2264
5	1.6	1715	2161.25
5	1.8	1898	2342.25
5	2	2165	2358.25

Table 37 Full Fuzzy PSO result on GEN02 problem

Analysis

Best Result obtained from k = 1.2 and size = 5 which is 1646 which is the best value achieved for this problem over the four PSO approaches.

Average time per run = 6965419 MS

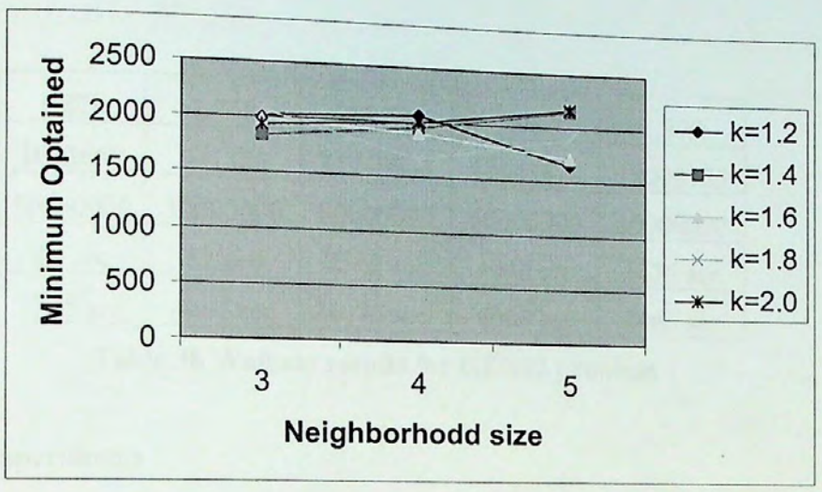

Figure 26 Minimum Value obtained VS neighborhood for different K

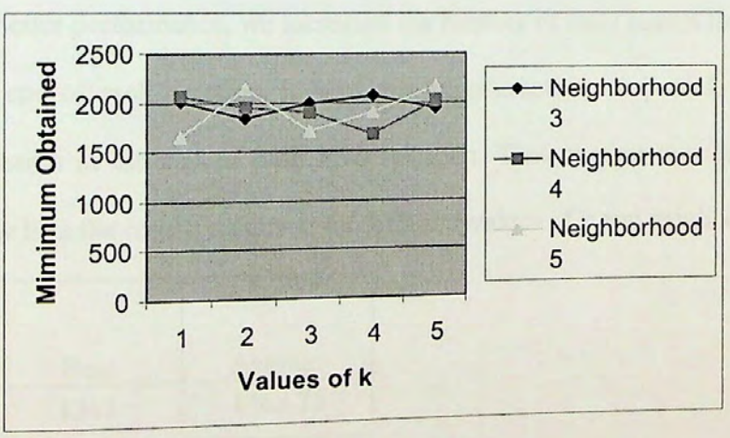

Figure 27 Minimum Value Obtained VS K for different neighborhood sizes

9.6.5.5 Walksat Performance on GEN02

Here we show the results obtained by running the Walksat algorithm on GEN02 problem for 5 different runs, the best value obtained was 1758. This value is worse than those obtained by PSO variations.

Result	1.879	1.758	2.078	1.983	1.892
Moves to best	1092650	941358	31749634	49643674	18099955
Moves total	50000000	50000000	50000000	50000000	50000000
Time to best	98 sec	82 sec	2592 sec	4040 sec	1476 sec
Time total	4235 sec	4067 sec	4076 sec	4069 sec	4066 sec

Table 38 Walksat results for GEN02 problem

9.6.5.6 Other experiments

9.6.5.6.1 Performance Enhancement

In order to get better performance, we increased the number of local search rounds performed by the particles at the end of each iteration. In previous experiments, each particle would perform three rounds of localsearch at the end of each PSO iteration. That number was increased to five rounds. The table below lists the results achieved for different values of k and neighborhood size.

k	Neighborhood Size	Best	Average
1.2	5	1313	1368.75
1.4	5	1408	1562.25
1.6	5	1137	1271.5
1.8	5	1086	1273.75
2	5	816	1144.75
2.2	5	1093	1330.75

2.4	5	1162	1256.75
2.6	5	1435	1527
2.8	5	1096	1298.75
1.2	4	888	1135
1.4	4	1151	1293.75
1.6	4	1052	1352.25
1.8	4	1108	1261.5
2	4	867	1224.25
2.2	4	1123	1243.75
2.4	4	1085	1216.5
2.6	4	987	1112.5
2.8	4	1051	1259.75
1.2	3	856	1216.75
1.4	3	1142	1281.75
1.6	3	991	1115.75
1.8	3	1167	1291
2	3	1106	1244.5
2.2	3	1135	1183
2.4	3	909	1057.75
2.6	3	1110	1244.5
2.8	3	1230	1329.5

Table 39 Fuzzy PSO with instinct, hypercube and local search results with increased Local search rounds for GEN02 problem instance

Best value achieved was 816 for k = 2 and neighborhood = 5.

Average Time = 20657694 MS

The results illustrated in the previous table shows good enhancement by increasing the local search rounds. However, the average time taken per run increased due to the time taken for more local search operation performed at each iteration by each particle.

Thus, we gave the same time taken by our Full PSO approach to Walksat by increasing the number of restarts for each run. We got the following result

Result	1699	1839
Moves to best	116399051	95749910
Moves total	125000000	125000000
Time to best	8505 seconds	6999 seconds
Time total	9134 seconds	9136 seconds

Table 40 Walksat performance on GEN02 problem given more time

The result achieved after giving the two approaches equal time shows that PSO (achieving 816) out performs Walksat (achieving 1699)

9.6.5.7 Analysis

We want to compare the minimum result obtained by each approach on this problem instance in order to show the quality of each approach and to be able to conclude how different PSO approaches are ranked against each other and against walksat.

1) Fuzzy PSO with instinct, hypercube and local search with more local search rounds → 816
2) Fuzzy PSO with instinct, hypercube and local search → 1646
3) Walksat (125,000,000 moves) → 1699
4) Walksat → 1758
5) PSO with instinct, hypercube and local search →4036
6) Fuzzy Gbest PSO →123219
7) Gbest PSO → 13430

9.6.6 Testing with GEN03 Problem Instance

These are the results obtained on large instances generated with the generator described in previous chapter. The instance contains 10000 variables and 80000 clauses. We call this instance GEN03

9.6.6.1 Gbest

The following table shows the result of Pure PSO on that problem GEN03. The algorithm was run 5 times.

Result
236435
239367
239152
239633
238536

BEST	236435
Average	238259.67
Average Time	3068970

Table 41 Original PSO results and Summary for GEN 03 problem

9.6.6.2 Fuzzy Gbest

We show the improvement achieved when introducing the fuzzy approach, we show the result of 44 runs with different values of k and different neighborhood sizes.

Neighborhood	k	100000000	Average
2	1	223744	226853
2	1.2	228741	230592.9844

2	1.6	231758	232884
2	1.8	231769	231989.3438
2	2	228748	230886.3281
2	2.2	232373	232887.9844
2	2.4	232221	232607.3438
2	2.6	230656	231444.3281
2	2.8	230165	231548.3438
2	3	228451	231201.6719
3	1	227630	228283
3	1.2	226006	227669.3281
3	1.4	225600	227476.6719
3	1.6	226671	228006.9844
3	1.8	226075	226944.3281
3	2	226497	227376
3	2.2	226730	228668.3281
3	2.4	226629	229001
3	2.6	226233	227920.3281
3	2.8	227081	227815.9844
3	3	227894	228738.6719
4	1	224331	226388.6719
4	1.2	226635	227385.3281
4	1.4	226721	227916
4	1.6	227293	228386.6719
4	1.8	228786	229673.3438
4	2	227122	228945.6563
4	2.2	227157	229337.3438
4	2.4	225550	227196.6719
4	2.6	227300	228457.6719
4	2.8	224715	227411

4	3	227029	229487.3281
5	1	226722	228764.3281
5	1.2	228487	230407.6563
5	1.4	226756	231602
5	1.6	230388	231285.6719
5	1.8	229654	230858.6563
5	2	229066	230293.3281
5	2.2	230105	232201.0156
5	2.4	230883	231574
5	2.6	228026	229021.6563
5	2.8	233284	233764
5	3	230681	231405.6563

Table 42 results of Fuzzy PSO on GEN03 problem

Analysis

Best 223744 for k = 1 and neighborhood size = 2.

Average time per run = 597900 MS

The value obtained after adding fuzzy neighborhood to PSO is better than the minimum obtained with pure PSO which was 236435 .

9.6.6.3 PSO with Instinct-Driven Particles, Hypercube Neighborhood and Local Search

To make good analysis of the results obtained, we show the results of the same test case on the full PSO approach with no fuzzy function. We show the results of running the program 10 times; we show the best and average value obtained along with average time per run.

Result
7739
10443
8932
7274
7277

Best	7274
Average	8156.5
Average Time	12467100 MS

Table 43 PSO with Instinct, Hypercube and Local Search results and Summary on GEN03 problem

We can notice that the result obtained which is 7274 is far better than the minimum achieved in the previous methods, which was 223744 and 236435 . This indicates the effect of the modifications to the basic PSO model, which are the instinct, hypercube neighborhood, and particles performing local search

9.6.6.4 Fuzzy PSO with Instinct-Driven particles, Hypercube Neighborhood and Local Search

Now we show the results with the best PSO configurations, fuzzy PSO with hypercube, instinct-driven particles and local search. The best value achieved is very good compared to any of the above results.

k	Neighborhood	Best	Average
1	3	3248	3709.199951
1.2	3	2581	3646.799805
1.4	3	3627	3746

1.6	3	3442	3743.399902
1.8	3	3676	3923.400146
2	3	3408	3894.400146
2.2	3	3655	3806.799805
2.4	3	3471	3641.600098
2.6	3	3428	3659.200195
2.8	3	3529	3817.800049
3	3	3588	3715.399902
1	4	3173	3886.599854
1.2	4	3187	3558.799805
1.4	4	3128	3605.800293
1.6	4	3146	3686.800049
1.8	4	2880	3462.600098
2	4	3781	3996.400146
2.2	4	3450	3616.199951
2.4	4	3280	3560.599854
2.6	4	2822	3626.199951
2.8	4	3319	3699.399902
3	4	3470	3719.199951
1	5	3392	3796.400146
1.2	5	3222	3581
1.4	5	3641	3790.999756
1.6	5	3462	3772.999756
1.8	5	3709	3894
2	5	3513	3913.599854
2.2	5	3555	3866.199951
2.4	5	3633	3963.399902

2.6	5	3564	3848.399658
2.8	5	3535	3928.400146
3	5	3515	3741

Table 44 Fuzzy PSO with Instinct Hypercube and Local Search results for GEN03

Analysis:

Best value achieved at k = 1.2and size =3 which is 2581

Average time per run = 17640121 MS

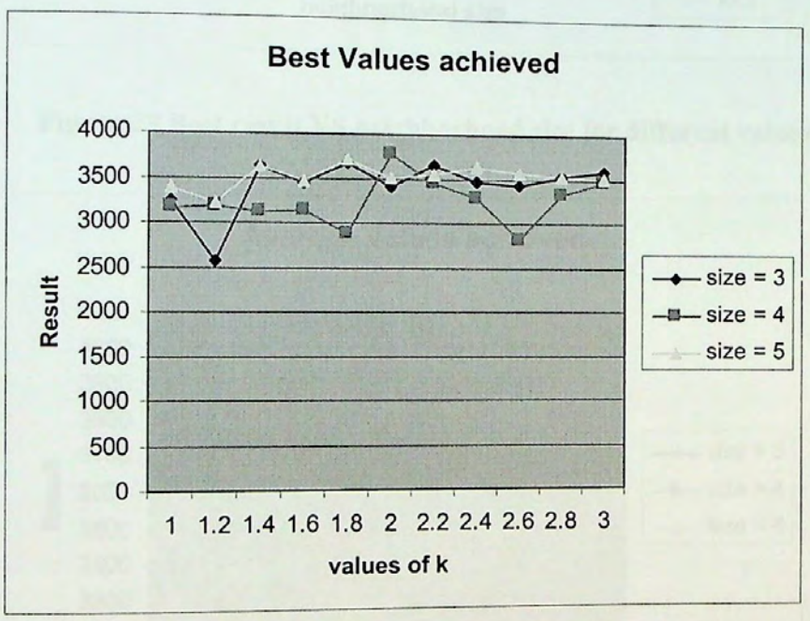

Figure 28 Best Result VS different values of K for different neighborhood sizes

Figure 29 Best result VS neighborhood size for different values of k

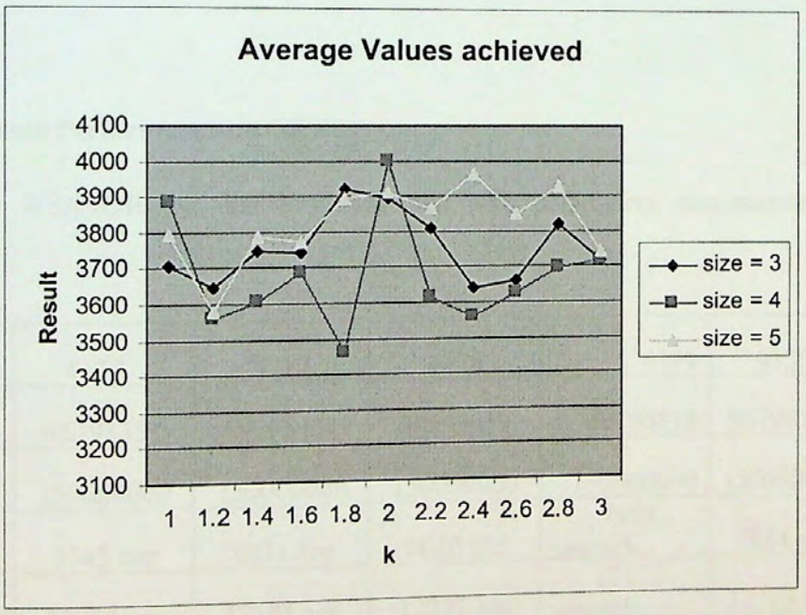

Figure 30 Average Value achieved VS k for different neighborhood sizes

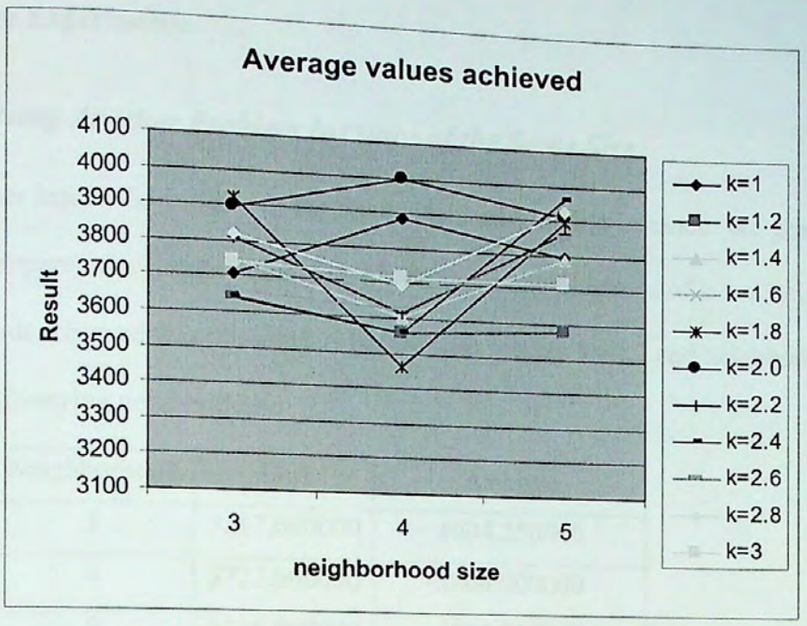

Figure 31 Average Value achieved VS neighborhood size for different values of k

9.6.6.5 Walksat Performance on GEN03

We ran WSATOIP on that large instance with parameters max-moves=100000, max-tries=1500.

Result	4139	4217	4276	4199	4139
Moves to best	46193131	99298441	68599913	60796518	96796182
Moves total	150000000	150000000	150000000	150000000	150000000
Time to best	3743 sec	8071 sec	5610 sec	4954 seconds	7854 sec
Time total	12035 sec	12185 sec	12227 sec	12173 seconds	12169 sec

Table 45 Walksat result for GEN03 problem

9.6.6.6 Other Experiments

9.6.6.6.1 Solving Another Problem Instance of the Same Size

To further insure the validity of the results achieved on that instance. We generated a second instance and compared the results obtained with different combinations of k and neighborhood size, still the best result achieved was the same as before with setting k =1.2 and neighborhood size = 3.

Result of varying neighborhood

k	Neighborhood	Best	Average
1.2	3	3217.000000	4004.250000
1.2	4	3722.000000	3864.000000
1.2	5	3335.000000	3713.750000

Result of varying k

K	Neighborhood	Best	Average
1.0	3	3318.000000	3716.000000
1.2	3	3217.000000	4004.250000
1.4	3	3338.000000	3677.000000

9.6.6.6.2 Performance Enhancement

In order to get better performance, we increased the number of local search rounds performed by the particles at the end of each iteration. In previous experiments, each particle would perform three rounds of local search at the end of each PSO iteration. That number was increased to five rounds. The table below lists the results achieved for different values of k and neighborhood size.

1.2	3	2230.000000	2490.000000
2	3	1898.000000	1993.000000
2	4	2184.0000000	2307.000000

Table 46 Fuzzy PSO with Instinct, hypercube and local search results with increased Local search rounds for GEN03 problem instance

Best value achieved was 1898 for k = 2.0 and neighborhood = 3.

The results illustrated in the previous table shows good enhancement by increasing the local search rounds. However, the average time taken per run increased due to the time taken for more localsearch operation performed at each iteration by each particle.

Thus, we gave the same time taken by our Full PSO approach to Walksat by increasing the number of restarts for each run. We got the following result

Result	4406	4273
Moves to best	71799586	240598100
Moves total	250000000	250000000
Time to best	5565 sec	18741 sec
Time total	19354 sec	19474 sec

Table 47 Walksat performance on GEN03 problem given more time

The result achieved after giving the two approaches equal time shows that PSO (achieving 1898) out performs Walksat (achieving 4273)

9.6.6.7 Analysis

We want to compare the minimum result obtained by each approach on this problem instance in order to show the quality of each approach and to be able to conclude how different PSO approaches are ranked against each other and against walksat.

1) Fuzzy PSO with instinct, hypercube neighborhood and local search with more local search rounds →1898
2) Fuzzy PSO with instinct, hypercube neighborhood and local search → 2581
3) Walksat → 4139
4) Walksat (250,000,000 moves) → 4273
5) PSO with instinct, hypercube neighborhood and local search →7274

6) Gbest Fuzzy PSO →223744

7) Gbest PSO → 236435

9.7 Result Analysis

For all the problem instances that we have seen an improvement in the total weights of satisfied clauses was achieved through variations on the PSO model. The instinct-driven particles along with hypercube neighborhood and local search, or what is called the Full PSO approach achieved improvement was in order of 100% better than pure gbest results.

Fuzzy PSO also improved the performance by creating more links between particles since it is not longer one best neighbor; it is a set of neighbors. It provides better means of communication between the swarm. Enhancement was up to 70 %

We have also seen a big improvement over Walksat for the instances of large size where it is known that local search techniques do not give optimal results.

There was also a variation of the runtime for different approaches. For example for the GEN01 problem the time taken for each method is described here

Gbest	21.64845 min
Fuzzy Gbest	19.60365 min
Full PSO	58.31275 min
Fuzzy Full PSO	98.11358 min
Walksat	78.21667 min

The parameters for the fuzzy PSO are the k factor and the neighborhood size. Experiments show that there is no best value for k or the problem size. We can consider them problem dependent. Here are the best values recorded for those parameters over the different problems.

Problem	Local Search Rounds	Best k	Best neighborhood size
LRAN	3	2.4	4
	5	1.6	3
GEN01	3	1.2	4
	5	2.6	3
GEN02	3	1.2	5
	5	2	5
GEN03	3	1.2	3
	5	2	3

Most of the problems recorded 1.2 as the best value for k. the neighborhood size seems to depend on the size of the problem. For the largest problem, the best value for neighborhood size was 3 particles.

10. CONCLUSIONS AND FUTURE WORK DIRECTIONS

10.1 Conclusion

We now conclude the research presented in this thesis. We highlight important ideas that lead to enhancing the performance of particle swarm algorithm and we emphasis some important results achieved. We give our expectations to other possible work that can be done in the same field.

PSO proved through the experiments to be a very interesting optimization algorithm. It is very close to the human life and way of learning and gaining experience. One advances in life through his own experiences as well as those of others. The instinct function can be thought of as the moments when a person makes up his mind to change his life just because he feels it will make him better regardless of his experience before or anybody else's advice. Particles also would try to reason and use their intelligence through evaluating themselves and examining possible changes. They can choose a greedy choice, or they can just decide to change themselves randomly.

To further model social behavior of living creatures, PSO can be hybridized with local search. This hybridization is best sought at the end of each iteration of algorithm. Local search helps particles explore their neighborhood and physically change their positions accordingly.

Making particles perform local search rounds enabled the particles to explore their neighborhood and make some greedy moves.

In an attempt to enhance communications between the population and enhance information sharing, we introduced the idea of Fuzzy PSO, this approach makes particles not only look at the history of one best particle, they would take into consideration the history of best 3 or 4 particles in their neighborhood. The fuzzy PSO approach was tested on the gbest neighborhood architecture and the hypercube architecture and in both cases it enhanced the performance.

From the results shown in the previous chapter, we can see that performance of PSO is very good compared to other optimization techniques. Tests have been made on several test cases of various sizes. It can be seen that there are certain problems, SAT encoded that Walksat performs worse in them. As a general remark, Walksat doesn't perform well on very large problems. It is so because Walksat, as well as other local search algorithms, starts at a random point and it could be the case that it starts very far away from optimal result. The probability of such cases increases when we deal with very large problems as the complexity of the search space increases.

10.2 Future work

The more experiments we do, the more we discover possible areas for work and experimentation. There are many possible optimization problems that PSO can be applied on. So far there have not been a lot of applications of PSO, major work focus on modifying the algorithm itself. There is still room for experimenting with the many parameters that PSO have. Each of them can present a search line by itself. The number of particles, i.e. swarm size can be varied to reflect the effect of it on performance. The neighborhood topology, especially the dynamic neighborhood architectures represent a potential area for research. The factors ϕ_1, ϕ_2 are also important for the convergence of the algorithm, they seem to be problem dependant; however, good research can be done to discover the relation between those factors and the problem to be solved. The value of inertia weight that affect the effect of previous velocity on next velocity value along with its decreasing scheme is another area for exploration. Those are all future work that can be conducted even on the pure PSO model.

Further research can be conducted with the instinct function. Possible functions could be tried. Random functions, I believe, could be tested to add element of variance and more exploration ability.

There is also possibility to work with the fuzzy model. Other alternatives could be explored to choose and rank among neighbors, other distributions (other than the Cauchy distribution) could be used to calculate factors for the functions.

Appendix A: Tabular Results

A.1 Results of Pure PSO algorithm on jnh class.

This table shows the results of 100 runs for each problem.

Jnh301	Jnh302	Jnh303	Jnh304	Jnh305	Jnh306	Jnh307	Jnh308	Jnh309	Jnh310
2293	3300	3081	3145	4574	2165	2221	4780	2674	2718
1022	3090	2889	2940	4461	2317	3182	2868	3319	4152
1377	4491	3208	3098	2949	2387	3728	2425	2473	3410
625	4614	3128	3741	4240	2350	2245	3726	1840	2686
1326	3195	2854	2940	4758	1561	3576	2755	2873	3299
1264	2602	2863	4521	2861	2059	2491	2403	3124	3429
1119	2610	2847	3129	4540	3774	3323	2430	3124	3764
1624	3914	2082	1550	3834	3504	2440	3611	2297	3537
1430	3042	3001	2514	2637	3643	2752	1983	2875	2659
2011	2810	3522	3437	3903	2246	4286	1974	3452	4350
1046	3402	1973	4619	2922	2596	4587	2200	2718	3988
793	3499	2644	2433	4866	2010	4537	4102	3406	3712
1697	3255	4337	2660	3304	1003	3098	3804	2081	3311
2062	3987	3365	4198	3436	1591	2939	825	3157	4151
3545	3530	1815	4137	5379	2143	3364	6248	2144	3615
2263	2918	2713	4084	3930	2397	2826	3281	4009	2805
1594	3157	1617	2343	2890	2555	2565	2682	2283	2691
1506	3926	3642	3170	3015	2694	2923	3941	2903	2694
2755	3207	2120	2727	3541	1823	4128	2547	2156	2802
2652	2610	3816	4348	4267	3014	3491	2860	3357	3376
826	2765	2420	5499	4454	3483	3391	2897	2215	2804
1657	4137	2602	3713	3849	2952	3369	2086	2792	4243
1373	3230	3885	4184	4272	2638	3259	2214	2520	3782
1270	2490	1558	3975	3350	2830	3683	3090	2938	2764
2076	4613	1402	3735	3751	2526	1917	2521	2434	3890
1569	3196	2855	3684	3809	3104	2722	3982	1363	3331
1430	2424	2237	4044	4632	1847	3678	1882	1858	3270

2672	4174	2670	2803	3481	2317	3066	2720	4277	2630
2040	3109	2259	3717	2976	2542	2841	3235	1316	2483
1488	3052	3163	2927	3150	2218	2519	1932	3251	4141
715	3379	3075	3236	3702	3757	3423	3833	2808	4131
989	4768	3240	2527	5600	1956	3376	3921	3894	1739
3608	2576	2786	3779	4198	1044	5006	3513	4280	3237
1794	2570	2525	3454	4707	1895	2891	2427	3538	2844
1768	3423	4157	2204	4789	2354	3176	3277	3207	4946
2000	2383	2346	3196	4862	2264	3158	3721	2208	3013
3601	3602	2579	3566	4970	2576	2123	3873	2818	4779
1392	3277	2680	2936	2951	1810	2121	1156	1589	3203
2621	3414	4296	2929	3483	2091	4939	2459	2711	2634
2185	3187	3732	2776	4728	1984	2919	3030	1544	3675
2255	3100	1834	2216	3902	3019	3611	3298	2637	4160
3137	3227	2477	4771	3659	2274	2853	3764	4722	4589
2391	3663	2965	4722	3139	2256	3526	4280	2976	3786
1805	3356	1787	2010	5451	1897	3549	3391	3125	3738
1834	4481	3693	2255	4210	1350	3287	3043	1785	3675
1306	3323	2179	3868	3977	2342	3825	3183	2410	3887
3273	4480	1876	1779	3852	1640	3639	1329	2642	3449
1754	2908	3424	2262	3529	2143	2867	2835	2802	3687
1071	2615	2509	2556	2978	2420	3773	2336	3931	3159
2144	3861	2439	2882	3649	2883	2897	3462	1884	3519
443	3355	3343	3350	3516	1954	2590	3171	3722	5485
495	3016	2719	3358	3262	2049	3308	3995	4320	3593
3304	2825	2070	4098	4949	1173	3769	1797	3077	4212
1134	3298	3380	2889	4886	3308	3820	1837	2914	3685
1880	2024	1987	3260	5661	1830	3171	2564	3932	2905
1603	3180	2525	2294	3198	979	5713	3375	2528	3152
949	4549	2359	2321	3170	2765	2485	2833	3798	2969
2906	3148	3516	3971	4547	2323	2644	2885	1891	3311
1485	3294	3212	3202	5073	2411	3284	2171	2250	3791
1763	3753	3511	3549	4947	2085	3214	3957	4829	2231

2293	3700	1796	2533	4494	2925	4566	1422	1419	2286
1221	3539	1660	2479	4066	2032	4314	2369	3205	2448
821	3925	3101	2747	2993	2442	3128	4250	1623	4433
1229	2149	3364	2893	4530	2874	2783	1862	2227	4115
1304	3433	3304	3046	3131	2561	3268	2840	2867	3251
2030	3748	2488	2893	4100	1547	2193	2421	3420	3149
2881	3252	1648	2660	3773	2639	3128	4092	2623	3650
879	3492	2694	4163	3526	3863	2865	2633	2854	4058
1259	3398	2478	2744	3373	1243	4641	1884	1968	4565
2451	4194	1896	4480	5226	1226	2241	3506	2964	3608
1530	3908	2499	2793	3763	3120	1925	3763	3189	3631
1615	2621	3731	3837	4743	3440	4393	4218	3464	2907
848	2961	1693	4305	3414	1859	3395	3955	2219	4387
2146	2993	3347	2787	3597	3260	2465	2707	2400	4052
1166	2309	3403	2076	3051	2035	3542	3834	2943	3333
1413	3545	2705	2133	2099	3088	2421	2034	1862	3456
1905	2865	3552	3722	5771	2477	2599	3161	2808	2449
1634	3478	3694	1632	4769	1315	3886	2171	1654	2971
1232	3148	2162	3944	3876	2581	2202	3781	2000	3313
2388	2478	3360	3740	4057	1982	4743	3406	3907	3894
1721	3123	2365	3480	4247	3083	2293	2955	1745	2681
799	3003	1659	3336	4293	2076	3252	2844	3189	5257
1837	2797	3655	3817	3054	2641	3644	2455	2589	3161
986	5892	1462	3242	3796	3199	4280	2696	3131	3212
824	2720	1842	5244	3893	2900	3247	3040	1982	3483
2095	4486	2640	3581	2784	3859	2416	2551	4069	3766
1831	2216	3216	1743	4159	2941	1921	2492	2699	3025
1522	2387	2565	3699	4421	2625	2698	1560	2762	2631
1204	4431	2650	2956	3072	2559	4349	2990	1861	4121
2512	2858	2051	3345	3075	3587	3843	3698	3460	4522
2326	3200	3302	3283	3961	1984	3490	2968	4018	3842
1349	2561	2466	2098	4163	3948	3641	3062	2951	4000
706	4394	3810	3931	3187	4197	4325	3032	2134	2742

1692	2177	3295	3215	3586	2459	2692	1165	2640	3373
1776	2997	2536	3782	4066	2622	4622	2702	2576	5459
2458	3235	1289	2469	□010	2246	2811	4032	1996	4563
2825	2797	2610	3089	3434	1755	2437	1245	1872	4212
1444	2817	2801	3592	3700	3186	2702	4158	4077	3802
1792	3690	3685	3904	3122	1199	4153	2220	2150	3544
2188	2553	1341	4413	3889	2501	3619	2149	4641	3528

A.2 Results of PSO with instinct and hypercube on jnh class

Jnh301	Jnh302	Jnh303	Jnh304	Jnh305
151	1732	770	1887	1216
287	561	555	2427	1038
105	864	1486	1924	799
64	925	811	1569	1045
114	1024	697	1319	810
141	600	1106	1929	1363
131	1957	1309	1961	774
100	503	1191	1561	824
105	395	697	1861	1347
662	959	752	1222	651
105	1355	516	2728	773
100	870	1185	1423	969
151	1767	752	1586	1238
178	731	881	1496	486
228	503	1246	1342	991
153	503	1414	2270	1372
287	395	1310	1404	509
68	993	841	1607	805

64	395	962	1618	1373
105	566	729	2067	638
64	946	811	1914	751
105	1440	1055	1286	994
141	441	954	2665	779
209	441	832	1569	369
312	1180	1363	2222	544
152	503	846	2055	1312
141	789	1212	1814	634
105	855	555	1942	1147
271	503	811	1537	1048
139	515	829	1837	1081
100	1080	1004	2130	523
76	911	1357	2200	993
274	1176	1005	1676	1124
607	900	962	2081	351
105	1014	773	1738	427
64	1742	1557	2475	1555
310	515	593	2303	1271
76	722	1314	1568	953
181	726	995	1689	427
154	1252	1106	1686	714
100	503	752	2715	1402
332	686	1009	1895	1374
104	503	1529	2402	1340
711	503	708	1797	976
151	566	640	1773	988
435	515	674	2139	1458
171	981	1509	1882	561

170	684	785	1613	978
179	920	1075	2385	1157
100	1180	846	2316	1690
68	1397	1606	1488	979
105	985	1591	1649	1187
228	503	1495	1357	680
279	662	1540	1381	531
286	1302	622	1367	1140
68	503	1119	1720	1255
321	1164	610	1266	1017
151	395	834	1954	1287
147	503	1350	1979	914
64	550	1122	1753	1336
105	503	621	1919	838
141	395	858	1407	761
114	395	1292	1918	843
290	395	500	2680	714
68	654	1159	2634	1488
249	395	1030	2052	838
184	406	654	1429	1121
100	395	1396	1270	321
234	1051	753	2006	824
141	1470	516	1627	799
198	395	631	2168	486
480	566	955	2161	785
105	959	626	1951	1449
141	615	1349	1497	787
105	395	653	1494	691
100	1180	942	1123	761

122	1230	985	1429	486
141	1419	1327	1827	810
76	1249	835	1915	721
105	925	1507	2046	588
100	503	1001	1370	1208
427	1209	1135	1842	561
76	395	752	1477	648
105	686	627	1795	799
189	1307	1159	2189	829
247	664	1960	2111	561
281	395	845	1612	507
154	503	666	1614	761
330	503	1446	1837	760
141	768	787	1702	1126
196	783	763	1427	1315
332	732	847	2284	1024
206	532	725	2084	351
105	1257	1044	3207	1206
319	1759	811	1544	921
154	1348	610	1528	818
168	732	1115	1755	1422
105	1544	690	1740	902
100	656	804	2076	1234
152	445	1450	1586	656
64	395	500	1123	321

A.3 Results of Original PSO with $\phi_1 = 1.1, \phi_2 = 0.6$ for jnh301 problem.

Jnh301	3715	1774	3771	3139

4297
3063
4604
3590
3123
6098
3585
1651
3043
5114
3350
2636
3930
4592
3381
4245
3835
3465
2549
2659
2672
3381

2602
3135
3710
3194
2819
2972
5282
4813
4853
3733
3530
2770
3997
3959
4652
3870
4490
3766
3368
2052
4172
2968

4009
3621
3980
4336
4489
4569
2741
3715
1774
3771
3139
4297
3063
4604
3590
3123
6098
3585
1651
3043
5114
3350

2636
3930
4592
3381
4245
3835
3465
2549
2659
2672
3381
2602
3135
3710
3194
2819
2972
5282
4813
4853
3733
3530

2770
3997
3959
4652
3870
4490
3766
3368
2052
4172
2968
4009
3621
3980
4336
4489
4569
2741
2280

A.4 Results for original PSO algorithm with hypercube neighborhood for jnh301 problem instance.

Jnh301	4384	3049	3533	3621

4083
3691
3900
4208
3401
2567
4699
4525
4004
3761
4021
4286
4328
3068
4283
3105
4204
4333
3818
3419

3519
3860
2957
3535
3303
3453
3735
3142
4108
3914
4517
3677
3811
4483
4536
4021
3342
4278
3398
3149

4231
4109
3778
3619
3775
4083
4310
3988
4567
3671
4576
4344
4420
4567
4525
2638
4042
4413
3079
4340

4380
4789
3724
3950
3897
4488
3736
4015
3613
3928
3780
3649
3678
3737
4042
3123
4385
3540
3812
3442

3553
3302
2844
2620
3972
4638
2985
4261
4260
2685
4476
3779
4034
4397
4170
3568

A.5 Results of PSO with hypercube, instinct and dynamic weights for the jnh class of problems, 100 runs for each problem.

Jnh301	Jnh302	Jnh303	Jnh304	Jnh305	Jnh306	Jnh307	Jnh308	Jnh309	Jnh310

60	867	610	938	1222	212	663	933	571	943
253	867	899	849	1222	461	580	620	402	943
253	1088	610	960	1251	225	771	705	469	1156
64	395	610	841	1674	466	663	693	561	937
279	843	697	740	1353	262	900	730	572	1213
76	901	610	829	1354	479	1057	286	462	1134
68	867	610	643	1721	288	1193	604	404	883
166	732	763	561	1979	582	885	782	571	943
105	819	900	954	1953	272	1440	644	462	1237
105	732	772	876	1353	849	580	286	457	1378
141	711	610	785	1641	533	771	311	571	944
64	1008	807	860	1602	307	854	527	631	1583
64	867	610	908	1737	162	667	961	496	967
247	899	610	785	1355	323	771	286	404	1105
76	899	610	876	1354	297	627	747	407	850
158	575	610	799	1267	154	950	666	431	1390
302	1197	775	886	1259	212	580	609	532	979
105	901	930	801	1222	393	580	738	418	1475
253	911	610	1018	1574	212	718	610	418	739
64	1180	610	785	1561	212	667	630	404	1034
105	1008	972	969	1814	592	929	608	521	1154
163	729	653	1046	1385	320	580	595	525	943
302	686	610	902	1222	242	1227	594	404	1078

236	1187	950	785	2080	592	810	541	462	885
177	867	943	507	1371	212	810	492	470	1086
166	732	859	936	1348	128	1048	454	431	943
141	1235	994	861	1452	610	580	593	570	943
208	1254	862	785	1188	309	1024	639	404	980
347	867	842	783	1561	349	655	666	572	953
228	686	888	954	1625	212	771	578	445	1314
257	867	610	852	1079	212	810	577	413	1365
76	503	848	832	1816	851	771	743	521	883
64	911	866	1005	1266	128	667	630	532	790
154	686	516	427	1309	212	580	647	532	1108
105	899	864	882	1344	225	580	632	404	739
168	868	989	714	1745	527	580	614	528	1053
260	729	640	954	1493	610	663	544	525	1220
337	632	698	1031	1493	297	580	679	457	943
302	732	850	785	1244	403	580	774	572	782
105	867	610	714	1441	212	780	893	418	883
343	566	610	1031	1600	307	848	705	431	883
68	900	610	1177	1879	705	663	765	521	1130
377	756	842	427	1764	307	580	676	404	991
100	503	848	1120	1631	513	1029	723	525	1012
228	395	555	643	1313	212	580	708	521	1247
12	867	610	892	1601	183	580	577	675	1052

284	1362	610	921	1347	236	663	381	571	817
224	1020	900	785	1148	139	782	497	431	943
337	395	655	674	1472	446	848	774	505	1557
100	867	913	740	1260	288	857	704	521	967
76	686	610	852	1624	339	780	693	525	2012
269	861	858	944	1735	592	823	466	404	687
152	911	848	714	1615	718	897	749	404	1240
64	503	1007	665	1561	225	580	845	431	1275
141	686	610	885	1222	212	855	651	404	896
247	732	610	952	1574	368	655	623	404	1037
174	549	1032	1101	1705	354	819	547	392	754
105	911	610	1138	1462	196	1029	884	521	933
171	732	848	935	1260	297	889	466	431	937
331	1453	1022	926	1253	212	667	772	325	953
247	686	977	352	1328	307	810	705	587	933
105	732	610	643	1259	212	1037	689	521	1143
105	837	858	740	1368	307	1097	711	521	937
64	913	610	1179	1107	339	874	632	404	953
105	1063	850	954	1587	700	958	672	418	953
64	686	610	960	1800	307	771	801	679	752
288	503	972	963	2458	307	780	577	431	1069
141	1110	610	783	1342	536	780	526	431	969
64	1008	610	1332	1989	288	823	550	431	1164

141	686	610	735	1735	560	849	738	521	1179
202	515	911	714	1477	162	823	654	521	1105
240	993	610	860	1717	515	823	577	525	1128
105	912	610	886	2015	123	1165	723	521	850
76	1018	937	799	1270	225	733	520	404	651
314	732	944	1120	1666	878	848	711	404	1353
295	728	890	799	1431	389	1003	846	566	1076
331	1235	610	694	1472	144	663	679	598	1771
105	729	850	643	1561	212	780	632	525	1058
300	993	610	954	1322	497	663	286	469	1194
105	867	859	876	2123	403	580	837	418	1730
280	981	610	544	1259	837	1105	672	521	1154
100	769	914	783	1590	212	580	286	521	811
430	578	783	840	1585	449	1004	602	548	955
170	913	610	643	1561	446	897	729	518	937
196	911	997	1065	1222	242	678	615	532	651
367	566	763	437	1320	288	823	565	404	679
388	884	890	799	1452	592	580	608	571	910
260	900	1332	832	1463	632	580	581	521	1136
317	884	566	954	1561	16	823	768	404	1169
331	899	610	740	1354	212	580	878	571	1065
247	785	610	672	1313	212	810	704	404	1047
64	728	610	643	1381	631	580	707	566	1224

64	1234	610	1048	1313	295	772	677	521	1166
194	964	610	785	1414	259	771	933	525	651
159	901	943	714	1222	259	843	689	470	782
183	911	950	876	1408	562	885	577	532	945
100	867	830	926	1453	582	843	745	548	850
233	711	610	588	1555	193	880	540	457	819
100	711	610	816	1259	193	878	286	276	1108
141	911		926	1993	397	848	721	462	1191

A.6. Results of Full PSO algorithm on jnh class of problems.

Jnh3 01	Jnh3 02	Jnh3 03	Jnh3 04	Jnh3 05	Jnh3 06	Jnh3 07	Jnh3 08	Jnh3 09	Jnh3 10
64	575	351	427	884	162	771	286	276	501
64	515	516	427	1259	16	540	286	276	651
64	575	516	427	884	16	540	286	276	651
64	503	516	427	1222	16	540	286	276	501
64	575	516	427	1000	16	540	286	404	651
64	575	351	427	1148	16	540	286	276	536
64	575	516	427	1203	16	580	286	276	509
64	395	516	321	1032	128	540	286	276	542
64	503	516	427	1079	16	540	286	276	651
64	395	516	523	1222	16	540	286	276	719
64	395	503	332	1203	16	540	286	276	746
64	657	516	427	1222	16	771	286	276	501
64	867	516	427	1079	16	771	286	276	501
64	578	431	427	884	16	580	286	276	651
64	503	351	427	742	16	580	286	276	501
64	515	516	427	884	16	771	286	276	651
64	395	516	427	1133	16	771	286	276	501

64	395	516	643	1203	16	580	286	276	670
64	622	516	427	1222	16	580	233	276	536
12	395	351	427	1073	16	580	286	276	501
64	740	516	427	1133	16	540	286	276	883
64	575	516	321	884	16	771	286	276	719
64	395	516	321	1000	16	540	298	276	501
64	575	351	588	1222	16	771	286	276	651
64	740	516	427	938	128	733	286	276	501
64	515	516	427	1133	16	655	286	276	501
64	575	503	588	884	16	771	286	276	501
64	669	516	427	1067	128	771	286	276	560
64	575	516	352	854	16	771	286	276	553
64	503	351	427	1188	16	540	286	276	651
64	503	503	486	1262	16	771	394	276	501
64	395	516	427	884	16	540	286	276	651
64	395	516	561	884	16	540	233	276	670
64	503	351	427	884	71	655	286	276	848
64	395	516	583	1133	16	846	286	276	687
64	395	516	321	1130	16	540	233	276	577
64	867	516	729	1222	71	580	286	276	501
64	503	516	427	1268	16	580	286	276	542
64	575	516	352	1000	16	655	286	276	651
64	867	516	427	1172	16	580	286	276	536
64	575	516	427	1133	16	655	286	276	501
64	395	503	427	1133	16	771	286	276	651
64	575	351	427	1259	71	715	466	276	651
64	395	516	427	1133	16	540	286	276	542
64	395	516	427	884	128	771	286	276	536
0	503	516	427	1107	16	580	190	276	501
64	575	516	427	1251	71	540	286	276	651
64	664	516	427	1067	16	540	286	276	670
64	395	516	427	1133	16	771	286	276	670

64	395	516	561	1130	16	771	286	276	651
64	503	516	427	1222	71	620	233	276	651
64	664	516	427	1133	16	655	233	276	501
64	575	516	427	1079	16	567	286	276	662
64	669	516	427	1032	16	540	286	276	501
64	740	516	637	884	16	655	286	276	553
64	867	516	427	1110	88	771	286	276	545
64	503	351	427	884	16	660	286	276	651
64	575	516	321	1222	16	540	286	276	545
64	395	516	321	1222	16	540	286	276	501
64	729	516	486	1032	16	655	286	276	501
64	395	516	427	1195	16	771	286	276	651
64	657	516	427	1133	16	771	233	276	501
0	395	516	427	1000	16	771	286	276	736
64	395	516	646	1079	16	540	286	276	687
0	669	516	427	1133	128	771	441	276	501
64	515	516	427	884	16	655	233	276	651
64	575	516	321	1251	16	540	286	276	651
64	740	351	427	1222	16	540	286	276	651
64	867	516	729	1222	16	540	286	276	651
64	575	516	588	1079	16	771	286	276	501
64	740	516	427	884	16	655	286	276	687
64	395	351	691	1124	16	540	286	276	501
64	575	516	427	1079	88	540	286	276	651
64	395	516	427	1183	16	771	286	276	501
64	664	516	427	1133	16	655	286	276	525
64	669	351	427	1128	16	655	286	276	463
64	395	516	427	1262	16	580	286	276	542
64	867	516	507	1270	71	540	286	276	501
64	867	516	574	1222	128	580	286	276	501
64	395	351	427	1268	16	540	286	276	651
64	503	516	427	884	16	580	286	276	651

64	686	351	427	884	16	540	286	276	542
64	575	516	321	1130	16	771	286	276	501
64	664	351	427	1000	16	540	286	276	542
64	503	516	427	884	88	655	286	276	806
64	740	488	427	742	16	655	286	328	651
64	578	516	427	1222	16	580	286	276	536
12	515	516	427	1032	16	540	286	276	687
64	575	516	486	1133	71	540	286	276	560
64	503	516	321	1000	16	771	286	276	739
64	395	516	427	1133	16	580	286	276	651
64	395	516	427	884	16	580	286	276	687
64	575	516	427	884	16	540	286	276	651
64	395	516	427	1203	16	655	286	276	501
64	686	410	588	1118	16	603	286	276	553
64	395	516	427	1222	16	540	286	276	719
64	395	516	427	884	16	655	233	276	651
64	729	431	427	1133	71	540	233	276	651
64	740	516	427	1079	16	540	286	276	501
64	575	516	427	1133	88	540	286	276	501

Bibliography

Bohlin, M. (2002). *Constraint satisfaction by local search*: Swedish Institute of Computer Science.

Casella, G., & Berger, R. L. (2002). *Statistical Inference* (Second ed.): Thomson Learning.

Christense, S., & Oppacher, F. (July 2001). What can we learn from No Free Lunch? A First Attempt to Characterize the Concept of a Searchable Function. *Proceedings of The Genetic and Evolutionary Computation Conference*, San Francisco, USA, 1219-1226.

Davis, M., & Putnam, H. (1960). A Computing Procedure for Quantification Theory. *Journal of Association for Computing Machinery*, 7, 201-215.

De Jong, K. A. (1992). Are genetic algorithms function optimizers? *Proceedings of The Second International Conference on Parallel Problem Solving from Nature*, 3-14.

Dimitris Achlioptas, Carla P. Gomes, Henry A. Kautz, & Selman, B. (2000). Generating Satisfiable Problem Instances. *Proceedings of 17th National Conference on Artificial Intelligence*, 256-261.

Eberhart, R. C., & Kennedy, J. (1995). A New Optimizer Using Particle Swarm Theory. *Proceedings of international symposium on Micro Machine and Human Science*, Nagoya, Japan, 39-43. Piscataway, NJ: IEEE Service Center

Eberhart, R. C., Simpson, P., & Dobbins, R. (1996). *Computational Intelligence PC Tools*: Academic Press.

Eberhart, R. C. K., J. (1995). Particle Swarm Optimization. *Proceedings of IEEE Intl. Conference on Neural Networks*, Perth, Australia, 1942-1948.

Feo, T. A., & Resende, M. G. C. (1995). Greedy randomized adaptive search procedures. *Global Optimization*, 6, 109-133.

Fogel, D. B. (1997). The Advantages of Evolutionary Computation. *Bio-Computing and Emergent Computation*, 1-11. World Scientific Press.

Fogel, L. J., Owens, A. J., & Walsh, M. J. (1966). Artificial Intelligence through Simulated Evolution. New York: Wiley Publishing.

Frank, J. (1997). Learning short-term weights for GSAT. *Proceedings of International Joint Conference on Artificial Intelligence*, 384-391. Morgan Kaufmann Publishers, San Francisco, CA, USA

Gent, I., Hoos, H., Prosser, P., & Walsh, T. (1999). Morphing: Combining Structure and Randomness. *Proceedings of Sixteenth National Conference on Artificial Intelligence AAAI-99*, Orlando, Florida, 654-660.

Gent, I., & Walsh, M. J. (1993). An Empirical Analysis of Search in GSAT. *Artificial Intelligence Research*, 1, 47-59.

Glover, F. (1989). Tabu Search - Part I. *ORSA Journal on Computing*, 1(3), 190-206.

Glover, F. (1990). Tabu Search - Part II. *ORSA Journal on Computing*, 2(1), 4-32.

Glover, F., & Laguna, M. (1993). Tabu Search. In C. Reeves (Ed.), *Modern Heuristic Techniques for Combinatorial Problems*. Oxford, England: Blackwell Scientific Publishing.

Goemans, M., & Williamson, D. (1994). A new 3\4 approximation algorithm for the maximum satisfiability problem. *SIAM Journal on Discrete Mathematics*, 7, 656-666.

Goemans, M., & Williamson, D. (1995). Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of Association for Computing Machinery,* 42, 1115-1145.

Goldberg, D. E., Deb, k., & Horn, J. (1992). Massive Multimodality, Deception and Genetic Algorithms. *Parallel Problem Solving From Nature*, 2, 37-46.

Hampson, S., & Kibler, D. (1993). Plateaus and Plateau Search in Boolean Satisfiability Problems: When to Give Up Searching and Start Again. *Workshop Notes: 2nd DIMACS Challenge.*

Hertz, A., Taillard, E., & Werra, D. d. (1995). A Tutorial on Tabu Search. *Proceedings of Giornate di Lavoro AIRO'95 (Enterprise Systems: Management of Technological and Organizational Changes)*, Italy, 13-24.

Heylighen, F. (1999). Collective Intelligence and its Implementation on the Web: algorithms to develop a collective mental map. *Computational and Mathematical Organization Theory*, 5(3), 253-280.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, Michigan: The University of Michigan Press.

Hoos, H. (1999). SAT-Encodings, Search Space Structure, and Local Search Performance. *Proceedings of Sixteenth International Joint Conference on Artificial Intelligence, IJCAI*, Stockholm, Sweden, 296-303.

Hoos, H. H., & Stützle, T. (1999). Local Search Algorithms for SAT An Empirical Evaluation: Technical Report, Department of Computer Science, University of BC, Canada.

Hu, X., & Eberhart, R. (2002). Multiobjective Optimization Using Dynamic Neighborhood Particle Swarm Optimization. P*roceedings of the IEEE Congress on Evolutionary Computation (CEC 2002)*, Honolulu, Hawaii USA,1677-1681.

Jason, S., & Middendorf, M. (2003). A Hierarchal particle swarm optimizer. *Proceedings of IEEE Congress on Evolutionary Computation*, Canberra, Australia, 770-776.

Jeremy Frank, Cheeseman, P., & Stutz, J. (1997). When Gravity Fails: Local Search Topology. *Journal of Artificial Intelligence Research*, 7, 249-281.

Johnson, D. (1974). Approximation Algorithms for Combinatorial Problems. *Journal of Computer and Systems Sciences*, 9, 256-278.

Josh Singer, Ian Gent, & Smaill, A. (2000). Backbone Fragility and the Local Search Cost Peak. *Journal of Artificial Intelligence Research*, 12, 235-270.

Kennedy, J. (1998). The Behavior of Particles. *Evolutionary Programming*, 7, 581-590.

Kennedy, J. (1999). Small Worlds and Mega-Minds: Effects of Neighborhood Topology on Particle Swarm Performance. *Proceedings of Congress of Evolutionary Computation*, 1931–1938

Kennedy, J., & Eberhart, R. C. (2001). *Swarm Intelligene*: Morgan Kaufmann.

Krink, T., Vesterstrom, J. S., & Riget, J. (2002). Particle Swarm Optimisation with Spatial Particle Extension. *Proceedings of the Fourth Congress on Evolutionary Computation (CEC 2002)*, 1474-1479.

Lorenc, H. R., Martin, O., & Stutzle, T. (2001). A Beginner's Introduction to Iterated Local Search. *Proceedings of the Fourth Metaheuristics International Conference*, Porto, Portugal, 1-6.

Lorenc, H. R., Martin, O., & Stutzle, T. (2002). Iterated Local Search. In G. Kochenberger (Ed.), *Handbook of Metaheuristics* (pp. 321-353). Norwell, MA: Kluwer Academic Publishers.

Lvbjerg, M., Rasmussen, T., & Krink, T. (2001). Hybrid particle Swarm Optimiser With Breeding and Subpopulations. *Proceedings of Genetic and Evolutionary Computation Conference (GECCO)*, 469-476

Marcus Randall, & Abramson, D. (1999). An Empirical Study of State Encoding in Tabu Search: Technical Report, School of Information Technology, Bond University.

McAllester, D., Selman, B., & Kautz, H. (1997). Evidence for Invariants in Local Search. *Proceedings of the Fourteenth National Conference on Artificial Intelligence AAAI-97*, Providence, Rhode Island, 321-326.

Millonas, M. M. (1994). Swarms, Phase Transitions, and Collective Intelligence. In T. Fukuda (Ed.), *Computational Intelligence: A Dynamic System Perspective* (pp. 137-151). Piscataway, NJ: IEEE Press.

Osman, I. (1995). Heuristics for the Generalised Assignment Problem: Simulated Annealing and Tabu Search Approaches. *OR Spektrum*, 17, 211-225.

Parsopoulos, K. E., & Vrahatis, M. N. (2002.). Recent Approaches to Global Optimization Problems Through Particle Swarm Optimization. *Natural Computing*, 1, 235-306.

Rechenberg. (1973). Evolutions Strategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution. Frommann-Holzboog,Stuttgart.

Rechenberg, I. (1994). Evolution Strategy. In C. Robinson (Ed.), Computational Intelligence: Imitating Life (pp. 147-159). Piscataway, NJ.: IEEE Press.

Resende, M. G. C., Pitsoilis, L. S., & Pardalos, P. M. (1996). Approximate Solutions of Weighted MAX-SAT Problem using GRASP. Murray Hill, NJ: AT&T Research.

Riget, J., & Vesterstrom, J. S. (2002). A Diversity-Guided Particle Swarm Optimizer (No. EVALife Technical Report no.2002-02). Aarhus C, Denmark: EVALife Project Group, Department of Computer Science Aarhus Universitet.

Schumacher, C., Vose, M. D., & Whitley, L. D. (July 2001). The No Free Lunch and Problem Description Length. *Proceedings of the Genetic and Evolutionary Computation Conference*, San Franscisco, USA, 565-570.

Schuurmans, D., & Southey, F. (2001). Local Search characteristics of incomplete SAT procedures. *Artificial Intelligence*, 132(2), 121-150.

Selman, & Kautz, H. A. (1993). Domain-Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems. *Proceedings of International Joint Conference on AI*, 290-295.

Selman, B., Kautz, H., & Cohen, B. (1994). Noise Strategies for Local Search. *Proceedings of 12th National Conference on Artificial Intelligence*, AAAI'94, Seattle/WA, USA, 440-446.

Selman, B., Mitchell, D., & Levesque, H. (1996). Generating Hard Satisfiability Problems. *Artificial Intelligence*, 81, 17-29.

Shang, Y., & Wah, B. (1998). A Discrete Lagrangian Based Global Search Method for Solving Satisfiability Problems. *Journal of Global Optimization*, 12(1), 61.

Shi, Y., & Eberhart, R. C. (1998). Parameter selection in Particle Swarm Optimization. *Evolutionary Programming*, VII, 611-616.

Shi, Y., & Eberhart, R. C. (1998). A modified Particle Swarm Optimizer. *Proceedings of IEEE Conference on Evolutionary Computation*, 69-73.

Spears, W., Jong, K. D., Back, T., Fogel, D., & Garis, H. d. (1993). An Overview of Evolutionary Computation. *Proceedings of European Conference on Machine Learning*, 442-459.

Stützle, T., Hoos, H. H., Roli, A., & Dorigo, M. (2001). A review of the literature on local search algorithms for MAX-SAT. Technical Report AIDA-01-02, FG Intellektik, FB Informatik, TU Darmstadt, Germany

Walsh, T., & Gent, I. (1993). Towards an Understanding of Hill-climbing Procedures for SAT. *Proceedings of AAAI-93*, 28-33.

Wilson, E. O. (1975). Sociobiology: The new synthesis. Cambridge, MA: Belknap Press.

Wolpert, D. H., & Macready, W. G. (1997). No Free Lunch Theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 4, 67-82.

Yannakakis, M. (1992). On the Approximation of Maximum Satisfiability. *Proceedings of the Third ACM-SIAM symposium on Discrete Algorithms*, 1-9.

Yao, X., & Liu, Y. (1997). Fast Evolution Strategies. In P. J. Angeline, R. G. Reynolds, J. R. McDonnell & R. Eberhart (Eds.), *Evolutionary Programming*, VI, 151-161.

Zhang, W. (2001). Phase Transitions and Backbones of 3-SAT and Maximum3-SAT. *Proceedings of 7th International Conference on Principles and Practice of Constraint Programming (CP2001)*. Springer, 153-167.

AMERICAN UNIV. IN CAIRO LIBRARY
3 8534 01039 1609