Universität des Saarlandes
Max-Planck-Institut für Informatik
AG5

Parallel Stochastic Local Search

Masterarbeit im Fach Informatik
Master's Thesis in Computer Science
von / by

Kaushik Mukherjee

angefertigt unter der Leitung / supervised by

Dr. Rainer Gemulla

betreut von / advised by

Dr. Rainer Gemulla

begutachtet von / reviewers

Dr. Rainer Gemulla
Prof .Dr. Gerhard Weikum

March 2012

Hilfsmittelerklärung

Hiermit versichere ich, die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben.

Non-plagiarism Statement

I hereby confirm that this thesis is my own work and that I have documented all sources used.

Saarbrücken, 01. March 2012,

(Kaushik Mukherjee)

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlichtwird.

Declaration of Consent

Herewith I agree that my thesis will be made available through the library of the Computer Science Department.

Saarbrücken, den 01. March 2012,

(Kaushik Mukherjee)

Acknowledgements

I would like to express my sincere gratitude to Dr. Rainer Gemulla for giving me the opportunity to work under his supervision. Regular discussions with Dr. Rainer Gemulla were helpful in setting up the targets and motivating to further meander through the course of this study. I would also like to thank Prof. Gerhard Weikum for being my reviewer. I would also like to thank the PDB group for organising regular discussion and readings in closely related areas which broadened my spectrum on understanding the topic of the thesis.

I would like to Thank my Parents for loving , understanding and unconditional support. Last but not the least I would like to thank my partner Ekta for believing in me and all her encourgements and love which she had shown.

Abstract

We present an extension of WalkSAT , a stochastic local search algorithm for solving large scale MAX-SAT problems.One important application of WalkSAT is context of Markov Logic Networks is finding its MAP solution.In this thesis , we implement and analyse differnet variants of Parallel WalkSAT algorithm and speedup in its runtime.We argue that instances which arise in context of Markov Logic Networks are significantly sparse and hence parallelism are suitable for them.Finally we evaluate the parallel algorithms on a Markov Logic instance from CORA dataset.

Contents

1 Introduction **4**

1.1 Motivation . . . 4

1.2 Contribution . . . 5

1.3 Outline of the Thesis . . . 6

1.4 Related Work . . . 6

2 Preliminaries **7**

2.1 The MAX-SAT Problem . . . 7

2.2 Stochastic Local Search(SLS) Algorithms for MAX-SAT . . . 8

2.3 Markov Networks . . . 9

2.4 Semantics . . . 10

2.5 Markov Logic Networks . . . 11

2.6 MAP Inference in MLN . . . 14

3 Parallel Stochastic Local Search Algoirthms **22**

3.1 Independent parallel flip . . . 22

3.2 Multiple synchronised flips . . . 24

3.2.1 Parallel MaxWalkSAT : Variable Locking . . . 26

3.2.2 Parallel MaxWalkSAT : Clause Locking . . . 28

3.3 Parallel GSAT . . . 30

4 Experiments **31**

4.1 Convergence Sequential MaxWalkSAT Algorithm . . . 31

4.2 Convergence of Parallel MaxWalkSAT (Independent Flips) . . . 32

4.3 RunTime : Randomly Generated Factor Graph . . . 35

4.4 RunTime : CORA Data Set . . . 36

4.5 Effect of changing density of Factor Graph . . . 36

4.6 Runtime of Parallel Algorithms 38
4.7 GSAT : experiments 38

5 Conclusion and Future Work 40

5.1 Conclusion 40
5.2 Future Work 41

1 Introduction

1.1 Motivation

Satisfiability problems are an important field in computer science. The basic satisfiability problem (SAT) is: given a Boolean expression, is there some assignment to the variables that makes the expression true? SAT is the defining problem in the complexity class of NP-complete problems; all NP-complete problems can be reduced in polynomial time to 3-SAT, and vice-versa. Specifically, general SAT problems can also be reduced to 3-SAT. In addition, many important problems across computer science can be expressed as SAT problems, such as planning and probabilistic planning problems , relational learning , and computer-aided processor design.

In particular, a SAT solver is a strong step towards a MaxSAT solver [BF97] where clauses are given weights, and we seek a solution that maximizes the sum of the weights of the satisfied clauses. MaxSAT is an important part of inference in many important problems, such as [RD06] Markov Logic Networks.

WalkSAT [SKC95] is a stochastic local search algorithm that tries to find satisfying solutions to Boolean expressions.A weighted version of this algorithm called MaxWalkSAT has been used in the past in finding most likely assigment to the variables in Alchemy and Tuffy sofware packages and in many other application relate to inference.

Extending algorithms to run on parallel architectures is increasingly important as chip manufacturers move from increasing clock speeds to increasing the number of cores on a processor. Clock speeds have remained relatively steady on processors for the last few years, as chip manufacturers reach the point where physical constraints cause multiple cores to be a more efficient investment of power than increasing clock speeds. In order for algorithms to continue to improve, they need to exploit this trend by scaling well across multiple processors

In this thesis we extend the MaxWalkSAT algorithm to parallel setting to scake over large weighted SAT instances which also arise in context of Maxkov Logic Networks.We first explore the effects of performing multiple MaxWalk-SAT runs in parallel independently in multiple processors, without synchorni-sation.MaxWalkSAT is particularly well suited to this parallelization; it can be parallelized by running multiple copies, which results in some speedup by taking the fastest of n runs.However this setting has very high memory overhead. We circumvent analyse the algorithm by splitting the MaxWalkSAT into different processers leverging the shared memory model for synchornizing the shared data using locking.

1.2 Contribution

We make the following Contribution in our work :

1. We implement the MaxWalkSAT Algorithm introduced in [SKC95] and study its convergence behaviour and performance from practical point of view.

2. We implement the GSAT Algorithm introduced in [SLM92] and study its performance from a practice point of view.

3. We implement a parallel version of the MaxWalkSAT Algorithm using independent flips and study its convergence behaviour and performance comapared to the sequential algorithm.

4. We implement a parallel version of the MaxWalkSAT Algorithm on shared memory model using synchronisation by locking shared data and compare its performance with the sequential and parallel independent flips algorithms.

5. We also implement a parallel version of GSAT Algorithm and compare its performance with the the sequential algorithm.

1.3 Outline of the Thesis

This thesis is organised as follow : Preliminary concepts that are referred to in the rest of the thesis are elaborated in Sectiom 2.The Parallel Version of the various algorithms are discussed and elaborated in Section 3.Section 4 describes the various experimental settings and provides various tests on performance and behaviour of the algorithms. Section 5 concludes this study and provides directions for future work.

1.4 Related Work

Local Search Algorithms ahave been affectively used in context of most probable explanations (MPEs) in [Par02].Local Search Algorithms have also been applied in finding the most probable explanations (MPEs) [Hut05] in graphical models, such as Bayesion Belief networks, this has shown very promising results.In this thesis we focus on the MaxWalkSAT Algorithm because it has been used for Inference [PD06]in Markov Logic Networks [RD06] and show very promising results.

In particular MaxWalkSAT forms the basis of many MAP inference [RD06] approaches.Scaling such algorithm in context of Markov Logic Networks has recieved special attention recently. The beginning of influx of such algorithms dealing with memory efficieny and runtime was LazyWalkSAT [SD06]. Another approaches which followed the LazySAT algorithm was Cutting Planne Infernce [Rie08].This approaches scales to real world networks. A distributed style of MAP inference based on MaxWalkSAT algorithm has also been used in Tuffy [NRDS11] which had shown vey promising results.

2 Preliminaries

2.1 The MAX-SAT Problem

MAX-SAT can be seen as a generalisation of SAT for propositional formulae in conjunctive normal form in which, instead of satisfying all clauses of a given CNF formula F with n variables and m clauses (and hence F as a whole), the objective is to satisfy as many clauses of F as possible. A solution to an instance of this problem is a variable assignment (i.e., a mapping of variables in F to truth values), that satisfies a maximal number of clauses in F . This definition captures the search variant of MAX-SAT; the evaluation vari- ant and associated decision problems can be defined in a similar way: Given a CNF formula F , in the evaluation variant, the objective is to determine the minimum number of clauses unsatisfied under any assignment; the associated decision problem for a given solution quality bound b is to determine whether there is an assignment that leaves at most b clauses in F unsatisfied.

Weighted MAX-SAT In many applications of MAX-SAT, the constraints represented by the CNF clauses are not all equally important. These differences can be represented explicitly and compactly by using weights associated with each clause of a CNF formula.The clause weights in a weighted CNF formula reflect the relative im- portance of satisfying the respective clauses; in particular, appropriately chosen clause weights can indicate the fact that satisfying a certain clause is considered more important than satisfying several other clauses.The objective is to minimise the total weight of the unsatisfied clauses, rather than just their total number.

Many combinatorial optimisation problems contain logical conditions that have to be satisfied for any feasible candidate solution; these conditions are often called hard constraints, while constraints whose violation does not preclude feasibility are referred to as soft constraints. When representing such problems as weighted MAX-SAT instances, the hard constraints can be captured by choosing the weights of the corresponding CNF clauses high enough that no combination of soft constraint clauses can outweigh a single hard constraint clause. MAX-SAT is an $NP-$hard optimisation problem, since SAT can be reduced to MAX-SAT in a straightforward way.

2.2 Stochastic Local Search(SLS) Algorithms for MAX-SAT

Many SLS methods have been applied to MAX-SAT, resulting in a large number of algorithms for MAX-SAT In this section, we present some of the most prominent and best-performing algorithms, including straightforward applications of SLS algorithms for MAX-SAT categorized primarily into variants of WalkSAT [SLM92] ,Tabu Search[MSG] and Iterated Local Search [SHS03].

Solving MAX-SAT Using SLS Algorithms A SLS algorithm for MAX-SAT is one that keeps track of the incumbent candididate solution and returns it at the end of the search process,provided its solution meets a given target , if such a bound has been specified by the input of the algorithm. GSAT and WalkSAT proposed in [SKC95] can be generalised to weighted MAX-SAT by using the objective function for weighted MAX-SAT - that is , the total weight of the clauses unsatisfied under a given assignment - as the evaluation function based on which the variable to be flipped in each search step is selected. A WalkSAT variant for weighted MAX-SAT with explicit hard and soft constraints was proposed in [JKS95] and had shown promising results in various sets of MAX-SAT encoded Steiner Tree Problemss.In principle WalkSAT is a 2-stage variable selection mechasim , where the variable to be flipped is chosen from randomly choosen unsatisfied clause.On the the other GSAT algorithm is a

1-stage greedy heuristic of variable selection.Apart from WalkSAT and GSAT there has been also many such hill climbing algorithms which has been developed over a period of time and has been successful in many applications.

2.3 Markov Networks

A Markov network is a model for the joint distribution of a set of variables X=(X_1 ,X_2 , . . . ,X_n). It is composed of an undirected graph G and a set of potential functions ϕ_k . The graph has a node for each variable, and the model has a potential function for each clique in the graph. A potential function is a non-negative real-valued function of the state of the corresponding clique. The joint distribution represented by a Markov network is given by

$$\mathbb{P}(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \prod_k \phi_k(x) \tag{1}$$

where Z is the partition function given by

$$\mathbb{Z} = \sum_x \prod_k \phi_k(x) \tag{2}$$

Markov Networks can be represented by *log linear models* replaced by the exponentiated sum of the weighted sum of features of the state , leading to

$$\mathbb{P}(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \exp(\sum_j w_j f_j(x)) \tag{3}$$

In this thesis we focus only focus on binary features $f_j(x) \in 0, 1$. In the most direct translation from the potential-function form (Equation 1) , there is one one feature corresponding to each possible state of the each clique, with its weight being $\log \phi_k(x)$. This representation is exponential in the size of the cliques. However, we are free to specify a much smaller number of features (e.g., logical functions of the state of the clique). allowing for a more compact representation than the potential-function form, particularly when large cliques are present.

2.4 Semantics

A knowledge base (KB) in propositional logic is a set of formulas over Boolean variables. Every KB can be converted to conjunctive normal form (CNF): a conjunction of clauses, each clause being a disjunction of literals, each literal being a variable or its negation. Satisfiability is the problem of finding an assignment of truth values to the variables that satisfies all the clauses (i.e., makes them true) or determining that none exists. It is the prototypical NP-complete problem.

A first-order knowledge base (KB) is a set of sentences or formulas in first-order logic. Formulas are constructed using four types of symbols: constants, variables, functions, and predicates. Constant symbols represents objects in the Domain (e.g , people : `Ana,Bob,Charles` etc) . Variable symbols range over the objects in the domain. Function symbols (e.g., `MotherOf`) represent mappings from tuples of objects to objects. Predicate symbols represent relations among objects in the domain (e.g.,`Friends`) or attributes of objects (e.g., `Smokes`). Variables and constants may be typed, in which case variables range only over objects of the corresponding type, and constants can only represent objects of the corresponding type. For example, the variable x might range over people (e.g., Anna, Bob, etc.), and the constant C might represent a city (e.g, Seattle, Tokyo, etc.)
A term is any expression representing an object in the domain.An atom is a predicate symbol applied to a tuple of terms (e.g , `Friends(x,MotherOf(Anna))`). Formulas are recursively constructed from atomic formulas using logical connectives and quantifiers. If F_1 and F_2 are formulas, the following are also formulas:

$\neg F_1$ (negation), which is true iff F_1 is false; $F_1 \wedge \mathrm{F}_2$ (conjunction), which is true iff both F_1 and F_2 are true; $F_1 \vee \mathrm{F}_2$(disjunction), which is true iff F_1 or F_2 is true; $F_1 \Rightarrow \mathrm{F}_2$(implication), which is true iff F_1 is false or F_2 is true; $F_1 \Leftrightarrow \mathrm{F}_2$ (equivalence), which is true iff F_1 and F_2 have the same truth value; $\forall x \mathrm{F}_1$(universal quantification), which is true iff F_1 is true for every object x in the domain; and $\exists x \mathrm{F}_1$ (existential quantification), which is true iff F_1 is true for at least one object x in the domain. Parentheses may be used to enforce precedence. A positive literal is an atomic formula; a negative literal is a negated atomic formula. The formulas in a KB are implicitly conjoined, and thus a KB can be viewed as a single large formula. A ground term is a term containing no variables. A ground atom or ground predicate is an atomic formula all of whose arguments are ground terms.

2.5 Markov Logic Networks

Example MLN

weight	**formula**	**Description**
1.4	¬`Smokes(x)`	Most people don't smoke.
2.3	¬`Cancer(x)`	Most people don't have cancer.
4.6	¬`Friends(x,y)`	Most people are'nt friends.
1.5	`Smokes(x)` ⇒ `Cancer (x)`	Smoking causes cancer.
1.1	`Friends(x,y)` ⇒ `Smokes(x)` ⇔ `Smokes(y)`	Friends have similar smoking habits.

Figure 1: An example of Markov Logic Network.

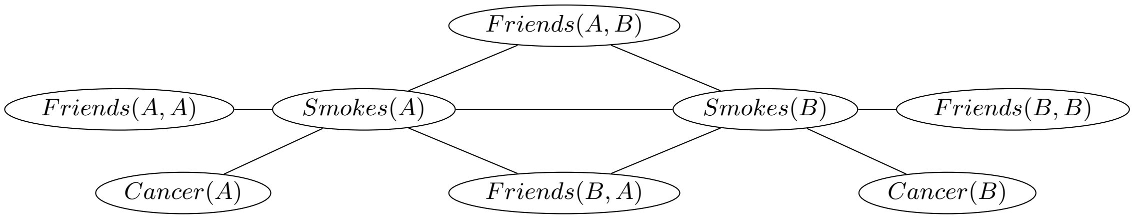
Markov Logic [RD06] is a combination of Markov Network(MN) and First-Order Logic(FOL).A FOL knowledge base(KB) is set of hard constraints on the set of possible worlds: worlds that violates even one formula, have zero probability.Markov Logic is based on the idea that these constraints must be soften: when a world violates one formula in the KB it is less probable, but not impossible. A world is more probable, if it violates fewer formulas.Each formula in Markov Logic has an associated weight that reflects how strong a constraint is: the higher the weight, the greater the difference in log probability between a world that satisfies the formula and one that does not, other things being equal.A set of formulas in Markov Logic is a Markov Logic Network.

A Markov Logic Network (MLN)[RD06] N is a set of pairs (F_i, w_i), where F_i is a formula in first-order logic and w_i is a real number.Together with a finite set of constants C=$c_1, c_2, ..., c_p$ it defines a Markov Network $M_{N;C}$ as follows :

1. $M_{N;C}$ contains one binary node for each possible grounding of each predicate appearing in N. The value of the node is 1 if the ground predicate is true and 0 , otherwise.

2. $M_{N;C}$ contains one feature for each possible grounding of each formula F_i in N.The value of this feature is 1 if the ground formula is true, and 0 otherwise.The weight of the feature is the w_i associated to the F_i in N.Thus there is and edge between two nodes of $M_{N;C}$ iff the corresponding ground predicates appear together in at least one grounding if the formula in N. For example, and MLN containing the formulas: $\forall x$`Smokes(x)` $\Rightarrow$ `Cancer (x)` and $\forall x \forall y$`Smokes(x)` $\wedge$ `Friends(x,y)` $\Rightarrow$ `Smokes(y)`, applied to the constants `Ana` and `Bob` yields the ground Markov Network in the figure.

Figure 2: Grounded Markov Network obtained by applying an MLN mentioned above to constants `Ana` or `A` and `Bob` or `B`



Markov Logic Network to Factor Graph

Factor Graph A factor garph [KF09] is a bipartiate graph representing factorization of a function.Given a factorizaion of a function $g(X_1, X_2, ..., X_n)$

$$g(X_1, X_2...., X_n) = \prod_{i}^{m} f_i(S_i) \tag{4}$$

where $S_j \subseteq$X$_1, X_2, ..., X_n$, the corresponding factor graph $F_G = (X, F, E)$ consists of variable vertices $X = X_1, X_2, ..., X_n$, factor vertices $F = f_1, f_2, ...f_m$ and edge E. The edges depend on the factorization as follows : there is an undirected edge between factor vertex f_j and variable X_k , where $X_k \in S_j$.

Consider a function g over some variables $g(X_1, X_2, X_3, X_4)$, if can be factored as,

$$g(X_1, X_2, X_3, X_4) = f_1(X_1, X_2) f_2(X_2, X_3, X_4) f_3(X_4)$$

then we can graphically represent such a factorization by a factor graph shown in the figure.

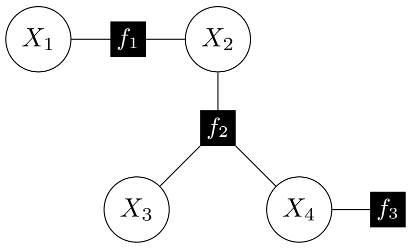


Figure 3: A factor graph

As mentioned in section earlier FOL formulas are represented in Conjunctive Normal Form (CNF). First-order formulas are converted to their clausal form and weight of a formula is equally divided between its clauses.

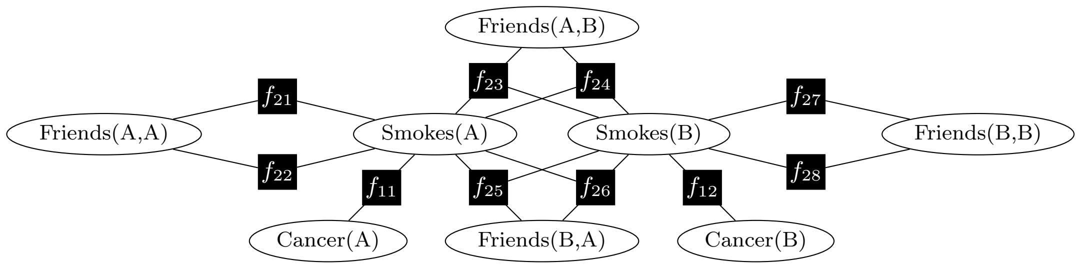
First-Order Logic		Clausal Normal Form	
1.5	`Smokes(x)` $\Rightarrow$ `Cancer(x)`	1.5	$\neg$`Smokes(x)` $\vee$ `Cancer(x)`
1.1	`Friends(x,y)` $\Rightarrow$ `Smokes(x)` $\Leftrightarrow$ `Smokes(y)`	0.55	$\neg$`Friends(x,y)` $\vee$ $\neg Smokes(x)$ $\vee$ `Smokes(y)`
		0.55	$\neg$`Friends(x,y)` $\vee$ `Smokes(x)` $\vee$ $\neg$`Smokes(y)`

Figure 4: First-order logic to Clauses.

Evidence
`Smokes(Ana)`
`Cancer(Ana)`
`Friends(Ana,Bob)`

Figure 5: Sample Evidence data for our example MLN.

Figure 6: Factor Graph



In the figure above each factor encodes a corresponding clause and each node represented a corresponding ground literal in the clause.

2.6 MAP Inference in MLN

Maximum a posteriori (MAP) inference means finding the most likely state of a set of output variables given the state of the input variables. MAP state of an MLN is the state that maximizes the sum of the weights of the satisfied ground clauses.This state can be efficiently found using a weighted MAX-SAT solver. We focus on one of the most popular category of local search algorithm based on a combination of RandomWalk and Greedy Heuristics.Such algorithms have been used used for Inference [PD06]in Markov Logic Networks [RD06] and show very promising results.

Inference using Satisfiability Let us revisit equation 3 which gives the distribution defined by markov logic network.The probability of a state x in a ground Markov network can be written as

$$\mathbb{P}(x) = \frac{1}{Z}\exp(\sum_i w_i f_i(x)) \tag{5}$$

where Z is the normalization constant w_i is the weight of the ith clause f_i=1 if the ith clause is true and f_i=0 otherise.Finding the most probabble state of a grounded MLN given some evidence is thus an instance of weighted satisfibility. If we look at the equation 4 above we have to maximize the exponent which is the sum of weights of all the satisfied clauses which belongs to the class of NP-Hard problems.The MAP problem can be generalised an instance of Optimization Problem.

Optimisation Problem An Optimization Problem can be generally formulated as follows : Given a *solution space* Σ of possible solution σ an *objective function* $f_{obj} : \Sigma \mapsto \mathbf{R}$ that allows to evaluate the quality of each candidate solution.Our aim is then to find the solution which achieves the maximum score $\sigma^* = \max_{\sigma\in\Sigma} f(\sigma)$
For many such problems its hard to approximate the Optimal Solution hence to resort heuristics like Stochastic Local Search.

Search Strategy

Local Search Local Search heuristics don't have any guarantee to find an optimal solution but are often effective in practice.Such search procedure operate over search space, starting at some location of the given search space it subsequently moves from the present location to a neighbouring location in the search space,where each location has only a relatively small number of neighbours, and each of the moves is determined by a decision based on local knowledge only. Typically, local search algorithms are incomplete, that is, there is no guarantee that an existing solution is eventually found, and the fact that no solution exists can never be determined with certainty. Furthermore, local search methods can visit the same location within the search space more than once. In fact, many local search algorithms are prone to getting stuck in some part of the search space which they cannot escape from without using special mechanisms, such as restarting the search process or performing some type of diversification steps. One such diversification step is Random Walk which prevents the algorithms from getting stuck in Local Minima.

Stochastic Local Search Many widely known and high-performance local search algorithms make use of randomised choices in generating or selecting candidate solutions for a given combinatorial problem instance. These algorithms are called stochastic local search (SLS) algorithms, and they constitute one of the most successful and widely used approaches for solving hard combinatorial problems.For a given instance of a combinatorial problem, the search for solutions takes place in the space of candidate solutions. Note that this search space may include partial candidate solutions, as required in the context of constructive search algorithms. The local search process is started by selecting an initial candidate solution, and then proceeds by iteratively moving from one candidate solution to a neighbouring candidate solution, where the decision on each search step is based on a limited amount of local information only. In stochastic local search algorithms, these decisions as well as the search initialisation can be randomised.

GSAT Algorithm The GSAT algorithm [SLM92]was one of the first SLS algorithms for SAT; it had a very significant impact on the development of a broad range of SAT solvers, including most of the current state-of-the-art SLS algorithms for SAT.GSAT is based on a 1-exchange neighbourhood in the space of all complete truth value assignments of the given formula; under this one-flip neighbourhood, two variable assignments are neighbours if, and only if, they differ in the truth assignment of exactly one variable.GSAT uses an evaluation function $g(F, a)$ that maps each variable assignment a to the number of clauses of the given formula F unsatisfied under a.GSAT and most of its variants are iterative improvement methods that flip the truth value of one variable in each search step. The selection of the variable to be flipped is typically based on the score of a variable x under the current assignment a , which is defined as $g(F, a) \longrightarrow$g(F,a$'$) , where a' is the assignment obtained from a by flipping the truth value of x. The basic GSAT algorithm startes from a randomly chosen variable assignment , in each local search it flips a variable with maximum score $score(x)$ defined by the $make(x) - break(x)$. Where $make(x)$ is defined by the total weight of the unsatisfied clauses by variable x which becomes satisfied when x is flipped. Similarly $break(x)$ is defined by the total weight of the satisfied clauses by variable x which becomes unsatisfied when x is flipped . In case there are multiple variables with same score, ties are broken randomly.The iterative best-improvement search underlying GSAT gets easily stuck in local minima of the evaluation function.Therefore, GSAT uses a simple static restart mechanism that re-initialises the search at a randomly chosen assignment every *maxFlips* steps. The search is terminated when the optimal search is reached (i.e the number of unsatisfied clauses is zero) or after *maxTries* times *maxFlips*.

The straightforward implementations of GSAT are rather inefficient, since in each step the scores of all variables have to be calculated.The key to efficiently implementing GSAT is to compute the complete set of scores only once at the beginning of each try, and then after each flip only update the scores of the variables impacted by the flip.

Algorithm 1 GSAT(weighted clauses,maxFlips,maxTries,target)

```
vars ← variables in weighted clauses
for $i = 1 \rightarrow maxTries$ do
    soln ← random truth assignment to vars
    cost ← sum of weights of unsatified clause in soln
    for $i = 1 \rightarrow maxFlips$ do
        if $cost \leq target$ then
            return Success solution is, soln
        end if
        compute DeltaCost(x)
        $x_f$ ← x with highest DeltaCost(x)
        $flip(x_f)$
        soln ← soln with$x_f$ flipped
        cost ←cost+DeltaCost($x_f$)
    end for
end for
return Failure , best assignment is ,best soln found
```

Implementation The implementation of the algorithm is as follows : after the random initialisation we compute scores of all the variables and store it in a Heap data structure ranked based on the score (i.e the variable with the highest score is at the top of the heap). Time complexity of insertion in heap takes $O(logn)$, where n is the number of variables. After each flip we update score of only those variables which have been changed.Updating the also takes $O(logn)$ time. To circumvent the efficiency issue we parallelize this algorithm in the section mentioned section 3.

MaxWalkSAT Algorithm One of the primary challenges in GSAT is getting stuck in local minimum such problem is addressed by natural choice of

Randomness first published in [SKC95].WalkSAT can be seen as an extension of the conflict-directed random walk method that is also used in [Pap91,]. It is based on a 2-stage variable selection process focused on the variables occurring in currently unsatisfied clauses. For each local search step, in a first stage, a clause c that is unsatisfied under the current assignment is selected uniformly at random. In a second stage, one of the variables appearing in c is then flipped to obtain the new assignment. Thus, while the GSAT architecture is characterised by a static neighbourhood relation between assignments with Hamming distance one, using this two-stage procedure, WalkSAT algorithms are effectively based on a dynamically determined subset of the GSAT neighbourhood relation. As a consequence of this substantially reduced effective neighbourhood size, WalkSAT algorithms can be implemented by incrementally updating variable scores and still achieve substantially lower CPU times per search step than efficient GSAT implementations. All WalkSAT algorithms considered here use the same random search initialisation and static random restart as GSAT.

The key ingredients of the algorithm mentioned above are :

1. In the greedy step to compute the the $DeltaCost(x)$ we need to compute the $score(x)$ defined by the $make(x) - break(x)$ where $make(x)$ - total weight of the unsatified clauses which will become satisfied , $break(x)$ - total weight of the satisfied clauses which will become unsatisfied on flip on the variable x.

2. The greedy step allows to maximize the number of satisfied clauses

3. In case variables have same score in the greedy step ties are broken randomly.

4.The random step allows to escape from the local minima.

5. After every MAXFLIPS the local search algorithm is restarted.

DataStructure In the MaxWalkSAT Algorithm a set of Unsatisfied Clauses is always maintained in order to speed up lookup of unsatisfied clause at each

Algorithm 2 MaxWalkSAT(weighted clauses,maxFlips,maxTries,target,p)

```
  $vars \leftarrow$ *variables in weighted clauses*
  **for** $i = 1 \rightarrow MAXTRIES$ **do**
    $soln \leftarrow$ *random truth assignment to vars*
    $cost \leftarrow$ *sum of weights of unsatified clause in soln*
    **for** $i = 1 \rightarrow MAXFLIPS$ **do**
      **if** $cost \leq target$ **then**
        **return Success solution is**, $soln$
      **end if**
      $c \leftarrow$ *a randomly chosen unsatisfied clause*
      **if** $Uniform(0,1) < p$ **then**
        $x_f \leftarrow$ *a randomly chosen variable from c*
      **else**
        **for** *each variable x in c* **do**
          *compute DeltaCost(x)*
        **end for**
        $x_f \leftarrow$ *v with highest DeltaCost(x)*
      **end if**
      $soln \leftarrow$ soln with$\mathrm{v}_x$ flipped
      $cost \leftarrow$cost+DeltaCost($\mathrm{v}_x$)
    **end for**
  **end for**
  **return Failure , best assignment is ,best soln found**
```

step. After every flip the set needs to be updated. We maintain an Indexed Map conatiner of the Unsatisfied Clauses. Complexity : Since updates and lookup in this conatainer is frequent hence we ensure Lookup in $O(1)$ and updates in $O(logn)$, this speeds up the overall performance of the algorithm significantly.

C_1	i_1
C_2	i_2
C_3	i_3
..	..
C_n	i_n

C_1	C_2	C_3	..	C_n

We maintain associative array of the Clause and the index of the clause $< C_n, i_n >$ and the Clauses are also stored in an array $UC[i]$.

$C_1, C_2, C_3, ..., C_n$ are the set of unsatisfied clauses and $i_1, i_2, i_3, ..., i_n$ is the index of the array $UC[i]$

lookup - is done on the array $UC[i]$, since it is done randomly hence its is in $O(1)$

delete - remove an element from $< C_n, i_n >$ has $O(logn)$ and correspondingly update the $UC[i]$ in constant time, hence the total time needed is $O(logn)$ in average case.

insert - insertion of an element in $< C_n, i_n >$ also has $O(logn)$ cost and correspondingly update the $UC[i]$ in constant time ,hence the total time needed is $O(logn)$ in average case.

3 Parallel Stochastic Local Search Algoirthms

There has many attempts to implement efficient Local Search algorithms for different applications.One such initiative is taken by [TH05] which provided a software library with efficient implementation of many such local search algorithms. In this thesis we focus on a having efficient implementations of the algorithms mentioned in the previous section on parallel architecture.As computers become more and more parallelized, scaling our algorithms to work in parallel becomes increasingly important.In this section we discuss issues like contention and deadlocks in parallellizing MaxWalkSAT algorithm and how we circumvent them.We also discuss an efficient implementation of parallel GSAT algorithm.

Parallel Computing A broad category of Parallel Computing is multi-core processor. A multi-core processor is a single computing component with two or more independent actual processors (called "cores"), which are the units that read and execute program instructions. Hence such processors needs some means of synchronising their execution of processes. The communication or synchronisation can be either at process level or data level. Process synchronization refers to the idea that multiple processes are to join up or handshake at a certain point, so as to reach an agreement or commit to a certain sequence of action. Data synchronization refers to the idea of keeping multiple copies of a dataset in coherence with one another, or to maintain data integrity. Here we use Process synchornisation for designing Parallel versions of the algorithms.

3.1 Independent parallel flip

The first approach to address runtime of sequential MaxWalkSAT Algorithms is by having n independent runs of the MaxWalkSAT algorithm. We run the

MaxWalkSAT algorithm independently in n processors with same number of flips as the sequential runs. After the indepedent execution in n different processor the best solution is choosen .This method does not have any communication cost among the runs of the algorithm in different processors.Since each processor runs independtly each of them keep a separate copy of the MAX-SAT instance and hence has very high memory overhead .

Algorithm 3 Slave(weighted clauses,maxFlips,maxTries,target,p)

```
  $vars \leftarrow$ variables in weighted clauses
  for $i = 1 \rightarrow maxTries$ do
    $soln \leftarrow$ random truth assignment to vars
    $cost \leftarrow$ sum of weights of unsatified clause in soln
    for $i = 1 \rightarrow maxFlips$ do
      if $cost \leq target$ then
        return Success solution is, $soln$
      end if
      $c \leftarrow$ a randomly chosen unsatisfied clause
      if $Uniform(0, 1) < p$ then
        $x_f \leftarrow$ a randomly chosen variable from c
      else
        for each variable x in c do
          compute DeltaCost(x)
        end for
        $x_f \leftarrow$ v with highest DeltaCost(x)
      end if
      $soln \leftarrow$ soln with$x_f$ flipped
      $cost \leftarrow$cost+DeltaCost($x_f$)
    end for
  end for
  return Failure , best assignment is ,best soln found
```

Algorithm 4 Master($list < soln >$)

```
  $soln_{best} \leftarrow$ choose the best solution computed by different nodes
return Best Solution
```

3.2 Multiple synchronised flips

Since the Ground Markov Logic Network is potentially very large the independent flips method yields benefit of linear speed up in the number of processors but is practically infeasible.Hence we exploit the possibility of using shared memory model to parallelize the MaxWalkSAT algorithm. The Ground Markov Logic Network encoded in a Factor Graph as mentioned above is shared across all processors.We use Reader-writer(RW) locks to synchronize the flips across the nodes. This is possible because we see in (Figure 7) that if the MaxWalkSAT Algorithms chooses say unsatisfied clause C_2 and in the greedy steep we need to choose a variable between $(v_f and v_1)$ so we need to to compute which we need to compute $score(v_f)$ and $score(v_1)$ for doing so we only need to lock the variables or clauses connected to these variables and not the entire graph. Which means for a Sparsely connected Factor Graph we benefit a lot since the variables not locked can be flipped by other processes running in parallel and having speed-up.In the next subsection we discuss in details what locks we use and in which case.

Model The instance of the factor graph which encodes a MAX-SAT instance as described in section 2. In the Parallel execution model we store the instance of the factor graph F_G in shared memory. Clauses $C_1, ..., C_5$ encodes a MAXSAT instance. If clause $C_3 = (v_f \vee v_1 \vee v_2); weight = 1.65$ and each of the literal in the clause has an boolean assignment of 0 or 1

We also store a *mutex* for each variable (variable locking scheme) or clause (variable locking scheme) in the shared memory

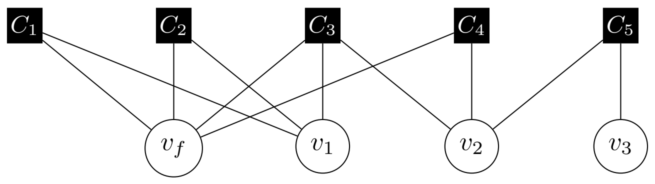


Figure 7: A factor Graph stored in shared memory

A readers-writer lock is like a mutex, in that it controls access to a shared resource, allowing concurrent access to multiple threads for reading but restricting access to a single thread for writes (or other changes) to the resource. One potential problem with a conventional RW lock is that it can lead to write-starvation if contention is high enough, meaning that as long as at least one reading thread holds the lock, no writer thread will be able to acquire it. Since multiple reader threads may hold the lock at once, this means that a writer thread may continue waiting for the lock while new reader threads are able to acquire the lock, even to the point where the writer may still be waiting after all of the readers which were holding the lock when it first attempted to acquire it have finished their work in the shared area and released the lock. To avoid writer starvation, a variant on a readers-writer lock can be constructed which prevents any new readers from acquiring the lock if there is a writer queued and waiting for the lock, so that the writer will acquire the lock as soon as the readers which were already holding the lock are finished with it. The downside is that it's less performant because each operation, taking or releasing the lock for either read or write, is more complex, internally requiring taking and releasing two mutexes instead of one. This variation is sometimes known as a "write-preferring" or "write-biased" readers-writer lock.

Large scale factor graph F_G have extremely sparse dependency (ie each variable to be flipped is connected to only to few other variables). Hence a variable flip does not impact the whole graph but only impacts only a few variables present in its neighbourhood. We leverage this property to acquire Read Locks only the variables in neighbourhood and Write Locks on the variable to be flipped since its we change its state.

Given a varibles v_i its neighbourhood is befined as set $S_j \subset \mathrm{F}_G$ where S_j containes all variables present in the clauses connected to variable v_i.

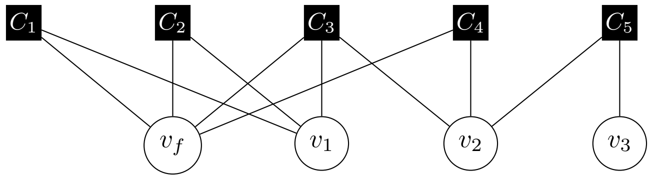


Figure 8: Neighbourhood Variables $S_j = v_1, v_2$

3.2.1 Parallel MaxWalkSAT : Variable Locking

Approach There are two steps involved in the MaxWalkSAT algorithm - selection step and update step. In the selection step random walk has no cost but the greedy step needs to select the best possible variable. This is a read operation on the clauses of the variable the *score* is being currently computed hence having read locks on the only on the variables present in these clauses connected to the variable is enough. In the upadate step one of the selected variables has to be flipped hence we upgrade the read lock on this variable to a write lock.The other variables still retain the read lock.The update step also involves updating the unsatisfied clause list from which the unsatisfied clause will be choosen in the next step. In the randow walk case we directly acquire write lock on the selected variable and read lock on the neighbouring variables.

Problems and Solutions

Deadlocks In the above parallel algorithm there might be following cases due to which deadlocks may arise.

Algorithm 5 PMaxWalkSAT(weighted clauses,maxFlips,maxTries,target,p)

```
$vars \leftarrow$ *variables in weighted clauses*
**for** $i = 1 \rightarrow MAXTRIES$ **do**
  $soln \leftarrow$ random truth assignment to vars
  $Unsatisfiedclauses \leftarrow$ compute all unsatisfied clause in soln
  $cost \leftarrow$ sum of weights of unsatified clause in soln
  **for** $i = 1 \rightarrow MAXFLIPS$ **do**
    **if** $cost \leq target$ **then**
      **return Success solution is**, $soln$
    **end if**
    $c \leftarrow$ a randomly chosen unsatisfied clause
    **if** $Uniform(0, 1) < p$ **then**
      $v_f \leftarrow$ a randomly chosen variable from c
      $N_v \leftarrow$ Neighbourhood variables set
      $R(N_{vs})$ Acquire Read locks on the Neighbours
      $W(v_f)$ Acquire a Write on the Variable to be flipped
      $flip(v_f)$
      $update(Unsatisfiedclauses)$
    **else**
      **for** *each variable v in c* **do**
        $N_v \leftarrow$ Neighbourhood variables set
        $R(N_{vs})$ Acquire Read locks on the Neighbours
        *compute DeltaCost(v)*
      **end for**
      $v_f \leftarrow$ *v with highest DeltaCost(v)*
      $W(v_f)$ Upgrade Read Lock on $v_f$ to Write Lock
      $flip(v_f)$
      $update(Unsatisfiedclauses)$
    **end if**
    $soln \leftarrow$ soln with$\mathrm{v}_f$ flipped
    $cost \leftarrow$costDeltaCost($\mathrm{v}_f$)
  **end for**
**end for**
**return Failure , best assignment is ,best soln found**
```

We illustrate this as follows , Say Thread T_1 and Thread T_2 tries running on two cores on the examples mentioned below

T_1 →a|b

T$_2$ →b|a

Say T_1 try to flip variable a hence acquires Write Lock $W(a)$ and T_2 tries to flip variable b hence tries to acquie a Write Lock b. But when T_1 and T_2 tries to acruire Read on its neighbourhood , it will reach a deadlock. In order to circumvent this we lock acquire the Read Locks on the Neighbourhood variables atomically and the attempt to acquire the Write Lock and flip the variable.

Performance In the MaxWalkSAT Algorithm a set of Unsatisfied Clauses is always maintained in order to speed up lookup of unsatisfied clause at each step. After every flip the set needs to be updated , but there are very few clauses which needs to be updated. Hence instead of locking the entire set of Unsatisfied Clauses we partition the set into k-partitions and only those lock partitions which needs to be updated.

For examples :

If there are 10 clause in the unsatisfied clause set $< C_1, C_2 ... C_10 >$. We partition into 5 disjoint partitions say $< C_1C_2, C_3C_4, C_5C_6, C_7C_8, C_9C_10 >$ and the clauses which needs to be updated are $< C_1, C_3 >$, then we only lock these two partition and rest can still be updated by other threads.

3.2.2 Parallel MaxWalkSAT : Clause Locking

Problem In the variable locking scheme proposed one of the bottle neck is that all the neighbourhood variables have Read Locks hence no such variable can be flipped any other processor.This reduces the search space for other processors to explore

Solution We propose parallelising MaxWalkSAT algorithm based on Clause Locking scheme.Each Clause in the neighbourhood is locked optimistically.This might allow other Threads to explore greater portion of the search space as compared to the variable locking scheme.

Let us take the example of variable locking.If Thread T_1 is flipping variable v_i then clauses C1..C4 cannot be choosen by Thread T_2.

On the other hand if we do lock the clause then we can do so depending on the state and semantics of the clause.We need to lock the unsatisfied clause where variable v_i is present but if the clause is satisfied but not by v_i so it will not have an impact it is flipped so we allow Thread T_2 to flip other variables in that clause. This will increase the search space to be explored by the Thread T_2

For example if the clause C3 and C4 is satisfied and the state the clauses are as follow : $C_3 = v_i \vee \mathrm{v}_1 \vee \mathrm{v}_2$ and the assignment of the variables are $v_i = 0$, $v_1 = 1$ and $v_2 = 1$ then if Thread T_1 flips v_i it will not change the state of the clause C_3 to unsatisfied hence we can allow Thread T_2 to flip other variables in the clause C_3 and hence giveing other Threads more variables to explore in the neighbourhood as opposed to the variable locking mechanism.

Similarly for $C_4 = v_f \vee \mathrm{v}_2$ if the assignment of variables is $v_f = 0$ and $v_2 = 1$, if Thread T_1 flips v_i ,then Thread T_2 can still flip variable v_2.

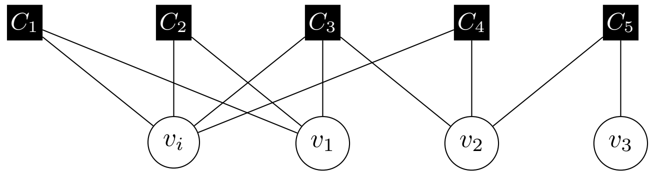


Figure 9: Neighbourhood Factors $S_j = C_1, C_2, C_3, C_4$

3.3 Parallel GSAT

The primary bottle-neck of GSAT is computing the score of all the variables. Partition the variables into k-disjoint sets and each node computes the scores of the given subset independently. The master node aggregates the set of variables ordered by the score which is emitted by the slave nodes. The variable with the highest score is flipped and only the variables affected are updated.Parallelizing the computation of the score yields speed-up but does not increase possibility to explore more search space.

Algorithm 6 Slave(Disjoint Set of variables ds_v)

```
for all $ds_v$ do
$score(v) \leftarrow$ compute the score of each variable in $ds_v$
scoreHeap$\leftarrow$ Maintain a $Heap$ of variables ordered by the score
end for
return Heap of variables ordered by score
```

Algorithm 7 Master (list of Heap of variables ordered by score)

```
$soln \leftarrow$ random truth assignment to vars
$cost \leftarrow$ sum of weights of unsatified clause in soln
$aggrScore \leftarrow$ aggregate the heaps created independtly
for $i = 1 \rightarrow maxTries$ do
  for $i = 1 \rightarrow maxFlips$ do
    if $cost \leq target$ then
      return Success solution is, $soln$
    end if
    $v_f \leftarrow$ choose the Top of the aggregated Heap
    $flip(v_f)$
    $soln \leftarrow$ soln with$v_f$ flipped
    $update(aggrScore)$ update only the scores of the affected variables
    $cost \leftarrow$costDeltaCost($v_f$)
  end for
end for
return Failure , best assignment is ,best soln found
```

4 Experiments

In section 2 and section 3 we introduce the Sequential and Parallel Stochastic Local Search Algorithms respectively. The primary goal of this section is to compare Runtime performance of these algorithm. We implement all the above algorithms in C++. The experiments are done on randomly generated Weighted MaxSAT instances and on publicly available Cora DataSet taken from Alchemy [DJK+09]

We study the following in the experiments on randomly generated Weighted MaxSAT instances :

1. Convergence behaviour of the Sequential MaxWalkSAT Algorithm.

2. Comparism of convergence behaviour of the Sequential MaxWalkSAT Algorithm and Parallel(independent flips) MaxWalkSAT Algorithm.

3. Comparism of Runtime of Sequential MaxWalkSAT ,Parallel MaxWalkSAT (Independent Flips) and Parallel MaxWalkSAT (variable locking and clause locking).

4. Effect of changing the density of the Factor Graph (ie Clause/Variable ratio) on the Runtime of the Sequential MaxWalkSAT Algorithm and Parallel MaxWalkSAT Algorithm (variable locking).

5. Runtime of the Parallel Algorithms.

6. Runtime of GSAT and Parallel GSAT Algorithms.

4.1 Convergence Sequential MaxWalkSAT Algorithm

Here we analyse the solution quality at after each flip and where at which stage the solution converges. Convergence is a constant number of steps for which the solution remains unchanged.

Number of Clauses = 1 million.

Number of variables = 20,000.

Number of flips = 60,000.

Number of restart = 4.

We observe that in the second try the algorithm somewhat gets stuck in a local minima region and then then the restart helps to escape from local minima region and explore better regions of the search space.In the third run the local search performs much better and achieves the best solution among the 4 restarts.It also converges to an optimal solution in the third run where the solution does not change drastically from 55,000 steps to 60,000 steps.

The above mentioned parameters are chosen empirically and may vary based on the structure of the network and the distribution of the weights on the clauses.In this case we create the random Factor Graph with exponential distribution of weights.Restart after every try reinitializes the assignment of the variables.Here we see that the algorithm overall has improvement with increasing number of flips this can be attributed to the choice of greedy flip.However we sometimes notice that the solution may sometimes degrades like in the first run between flips 55000 and 60,000, this can be attributed either to the choice of random flips.

4.2 Convergence of Parallel MaxWalkSAT (Independent Flips)

We run the Parallel MaxWalkSAT algorithm with equal number of Independent Flips in 4 different cores and choose best solution among them.The hope is algorithms on different cores explores different portions of the network and get a better solution by avoiding the random restarts. Hence we also get linear speed-up on the number of processors in the overall runtime time of the algorithm.

We run this on the same network mentioned in the first experiment and

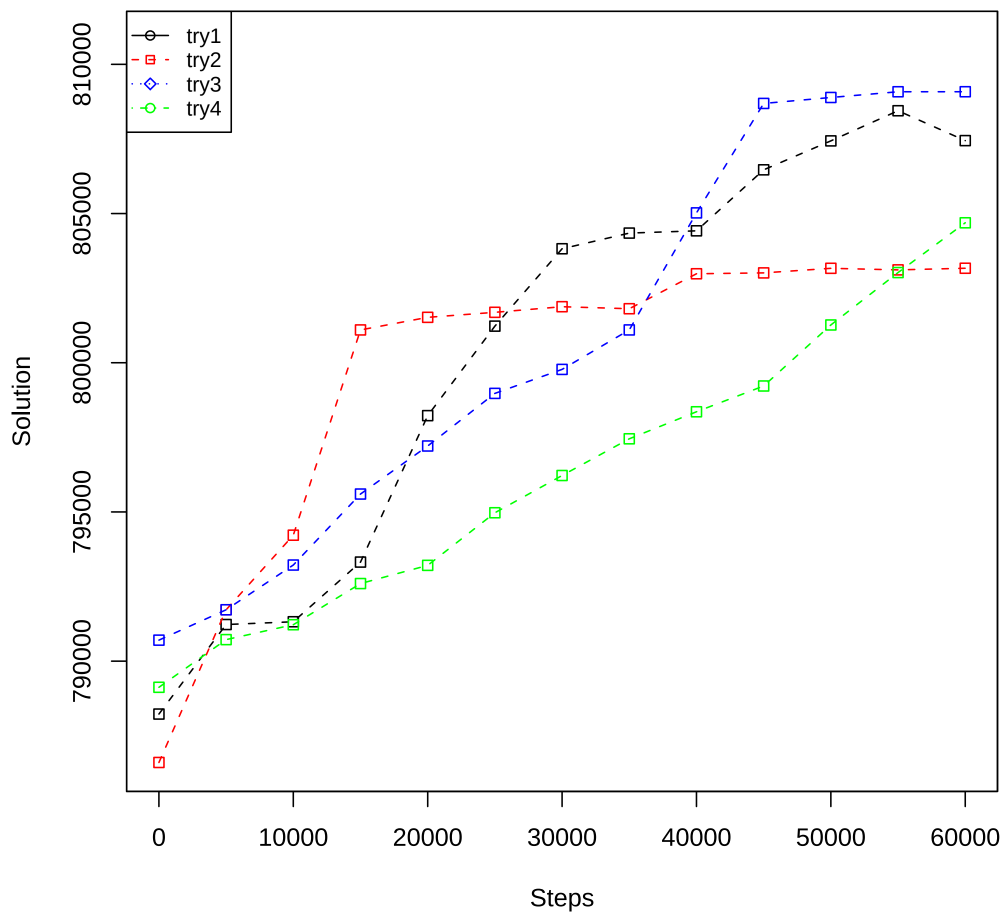


Figure 10: Sequential MaxWalkSAT with four restarts

run it with 60,000 flips on each processor. We observe core3 achieves the best solution and returns approximately solution same as in the sequential case. As in the sequential case we can also see in core2 the algorithm gets gets stuck in a local minima.

Tough this way of parallelism has negligible synchronization cost but it has huge memory overhead as all the processors will have a separate copy of the factor graph and variable assignmente.Hence it becomes infeasible to run this algorithm over larger networks.

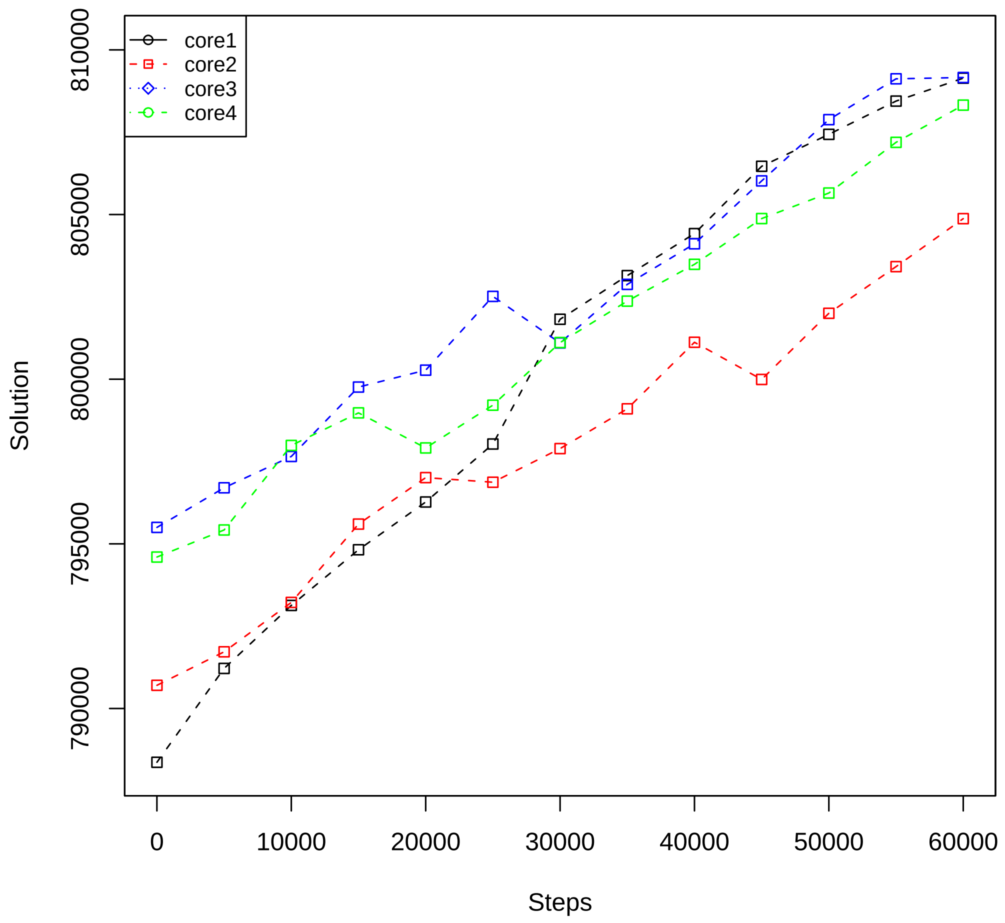


Figure 11: Convergence of Parallel MaxWalkSAT with independent flips on 4 processors

4.3 RunTime : Randomly Generated Factor Graph

We run the Parallel MaxWalkSAT algorithm with variable locking and clause locking.We run each of these algorithms on 4 processors and do synchronisation among the flips based on the shared memory model.We divide the flips equally among each processor, in this case each node runs with 15,000 flips.We compare runtime of the sequential algorithm,the independent flips , variable locking and clause locking.

Sequential Algorithm - 60,000 flips ; 4 restarts.

Parallel Algorithm with independent flips - 4 cores ; 60,000 flips in each processors.

Parallel Algorithm with variable locking - 4 cores ; 15,000 flips in each processors.

Parallel Algorithm with clause locking - 4 cores ; 15,000 flips in each processors.

Runtime MaxWalkSAT (in mins) on Random Factor Graph

Sequential	**Parallel Indepedent Flips**	**Variable Locking**	**Clause Locking**
67.5263	16.8722	23.6042	21.9173

We clearly observe the Parallel Algorithm with independent flips is 4 times faster than Sequential Algorithm. But in case of variable locking and clause locking the performance is promising but not linear in the number of processors. This can be attributed to the synchronisation cost involved in each flip of variable.Since we are using a shared memory model for data synchronisation hence we obviously circumvent the issue memory overhead.

With increase in the size of the neighbourhood of the variable or the density of the network the synchronisation cost may increase and hence the runtime of the algorithm may deteriorate. However for sparse networks this cost will not be too much and hence lock based synchronisation will still have benefit.In all the above algorithms the solution is approximately the same.

4.4 RunTime : CORA Data Set

We performed our experiments also on publicly available CORA DataSet taken from [DJK$^+$09] used for Entity Resolution tasks.In many domains , the entities of interest are not uniquely identified , and we need to determine which records are duplicates.For example, when merging databases we need to determine which records are duplicates.This problem is of crucial importance to many large scientific projects, businesses, and government agencies, and has received increasing attention in the AI community in recent years.The CORA DataSet contains 1295 citations and is extracted from the original Cora database of over 50,000 citations.We learn the weights and ground the network using [DJK$^+$09] and create a compact representation of factor graphs and run the sequential and parallel algorithms mentioned above.The grounded instance of the CORA dataset has 15499188 clause and 69276 variables but we observe the dataset is extremely sparse i.e only few variables are connected to many clauses. We study the runtime behaviour of the algorithm in such a setting.In all the above algorithms the solution is appoximately the same.

Sequential Algorithm - 200,000 flips ; 4 restarts.

Parallel Algorithm with independent flips - 4 cores ; 50,000 flips in each core.

Parallel Algorithm with variable locking - 4 cores ; 50,000 flips in each core.

Runtime MaxWalkSAT (in mins) on CORA

Sequential	**Parallel Indepedent Flips**	**Variable Locking**
197.2412	52.8722	75.3711 height

4.5 Effect of changing density of Factor Graph

Local Search algorithms improves its solution iteratively increasing the density means that in the greedy flip the algorithm has will have to iterate through a larger number of clauses in order to choose the best flip. Maintaining the unsatsfied clause list also becomes more expensive because each will impact a greater number of clauses.We empircally study the impact of changing density of the graph on the Sequential Algorithm We also observe similar behaviour in

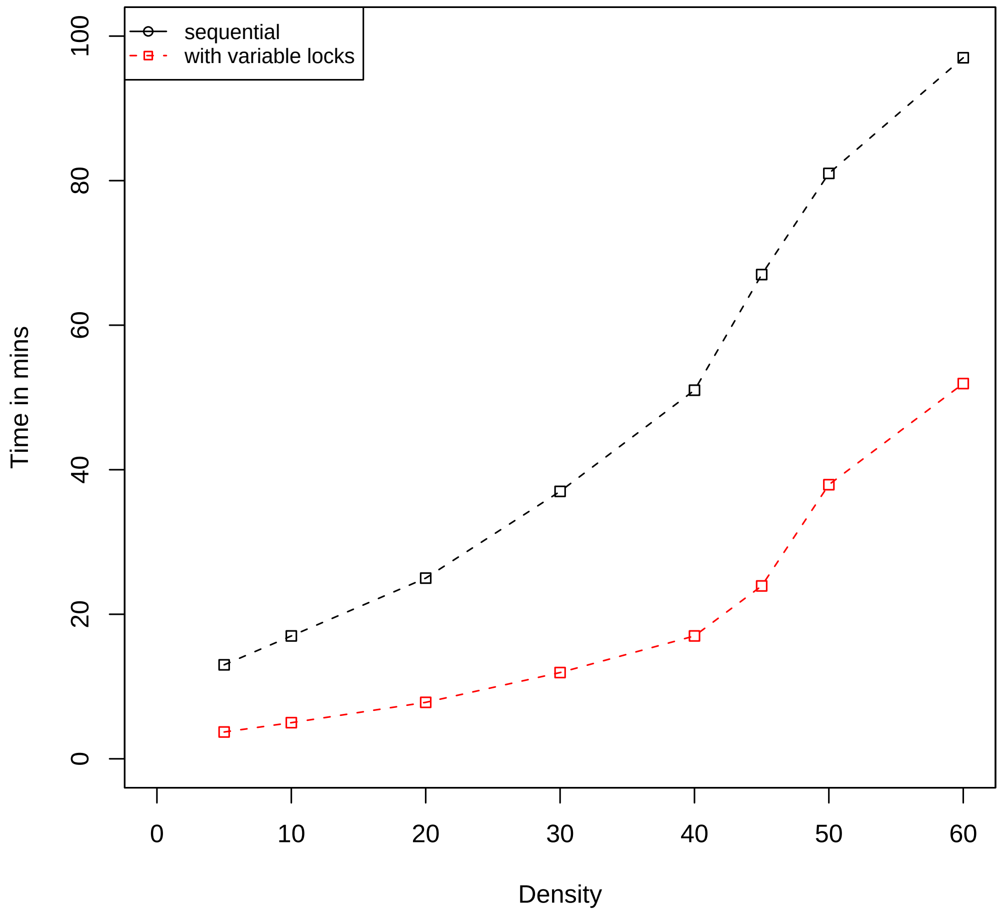


Figure 12: Effect of changing density of Factor Graph on MaxWalkSAT Algorithm

the Parallel algorithm but the overall runtime is also effected because of the contention and deadlock which arise in a multicore environment. But in case of sparse network contention and deadlocks does not arise frequently so the locking still scales to real world instances.

4.6 Runtime of Parallel Algorithms

We compare of the runtime of the Parallel algorithms in 2,4 and 8 processor machines. Here we use the same Random instances with 1 million clause and 20,000 variables. For Independent Flips we use 60,000 flips per processor and for the Parallel variable locking and Clause locking we use 15,000 flips in each processor. We see in the figure above that the independent flips also scales linearly.However the variable locking scheme is also has considerable speed. In large scale instances the Clause locking scheme shows improvement compared to variable locking scheme because we lock the clauses lazily hence reducing contention and deadlocks.We lock the clauses only if its state is impacted by flipping the variable.

4.7 GSAT : experiments

We evaluate the performance of the sequential GSAT algorithm and Parallel Algorithm on Randomly Generated Factor Graphs. As mentioned earlier one of the primary bottlenecks of the Sequential Algorithm is to compute the score of all the variable and choose the one which has maximum benefit and update the scores affected.Computing score of all the variables is computationaly very expensive hence we choose a smaller instance for this algorithm.

Number of clauses - 100,000

Number of Variables - 10,000

number of flips - 30,000

Runtime - 27.18 mins(sequential)

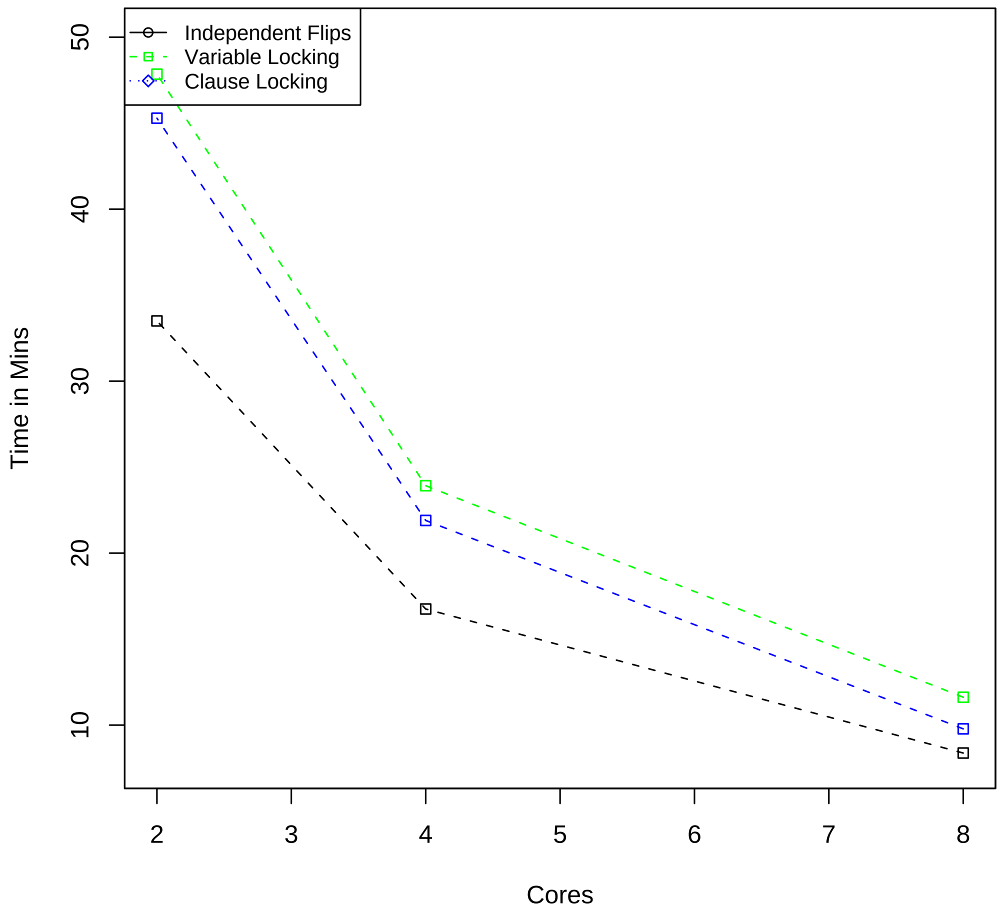


Figure 13: Runtime of Parallel MaxWalkSAT Algorithms

In the parallel setting the variable set is partitioned into 4 partitions (i.e 2500 variables are given to each processor)¿each processor computes the scores of the variables independently this reduces the runtime linearly. However the updating Heap on fliping the variable is still considerably expensive.

Number of Processors - 4

Runtime - 7.51 mins (parallel)

In both cases we contruct the Random Factor Graph with exponential distribution of weights .Both the algorithm also return approximately the same solution.

5 Conclusion and Future Work

5.1 Conclusion

Satisfibilty are hard to approximate but metaheuritsics like MaxWalkSAT extremely fast and yields reasonable solutions for large instances both random and real world (CORA).We propose a generic way to parallelize Local Search Algorithm where the each variable is dependent on few other variables in the network. The efficient implementation of variable locking and clause locking helps to parallelize a single chain and has runtime improvement of in order of 75-80 percentage of n processors in other words in case of using 4 processor we have approximately 3 times faster than the sequential MaxWalkSAT Algorithm. The Independent flip has linear speedup in order of number processor but may be practically infeasible because of the high memory usage. We also propose a parallel implementation of GSAT Algorithm which also has almost linear improvement compared to sequential GSAT. Comparing Solution of GSAT (sequential and parallel) with MaxWalkSAT(sequential and parallel) , GSAT solution quality is better than MaxWAlkSAT but not enough to pay of the high Runtime of GSAT compared to MaxWalkSAT. Hence we see empirically that Parallel MaxWalkSAT (variable locking or clause locking) has best RunTime and feasible in large scale sparse networks like CORA

5.2 Future Work

Our work can be further expanded to implement these algorithms in a distributed setting on a partitioned Factor Graph. It will be also interesting to combine the clause locking and variable locking scheme and analyse its performance. An runtime evaluation of the Parallel MaxWalkSAT in context of inference in MLN for MCSAT algorithm [PD06] might be very useful. An empirical runtime comparism of the MAP inference results of the Parallel MaxWalkSAT algorithm on networks with frameworks like Tuffy.

References

[BF97] Brian Borchers and Judith Furman. A two-phase exact algorithm for max-sat and weighted max-sat problems. *Journal of Combinatorial Optimization*, 2:299–306, 1997.

[DJK$^+$09] Pedro Domingos, Dominik Jain, Stanley Kok, Daniel Lowd, Lily Mihalkova, Hoifung Poon, Matthew Richardson, Parag Singla, Marc Sumner, and Jue Wang. http://alchemy.cs.washington.edu/, 2009.

[Hut05] Frank Hutter. Efficient stochastic local search for mpe solving. In *In Proc. of IJCAI-05*, pages 169–174, 2005.

[JKS95] Yuejun Jiang, Henry Kautz, and Bart Selman. Solving problems with hard and soft constraints using a stochastic algorithm for max-sat, 1995.

[KF09] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques.* MIT Press, 2009.

[MSG] Bertrand Mazure, Lakhdar Sa, and Eric Gregoire. Tabu search for sat. In *In Proceedings of AAAI?97*.

[NRDS11] Feng Niu, Christopher Ré, AnHai Doan, and Jude W. Shavlik. Tuffy: Scaling up statistical inference in markov logic networks using an rdbms. *CoRR*, abs/1104.3216, 2011.

[Pap91] Christos H. Papadimitriou. On selecting a satisfying truth assignment (extended abstract). In *Proceedings of the 32nd annual symposium on Foundations of computer science*, SFCS '91, pages 163–169, Washington, DC, USA, 1991. IEEE Computer Society.

[Par02] James D. Park. Using weighted max-sat engines to solve mpe. In *in AAAI*, pages 682–687. AAAI/IAAI, 2002.

[PD06] Hoifung Poon and Pedro Domingos. Sound and efficient inference with probabilistic and deterministic dependencies. In *AAAI'06*, pages –1–1, 2006.

[RD06] Matthew Richardson and Pedro Domingos. Markov logic networks. *Mach. Learn.*, 62:107–136, February 2006.

[Rie08] Sebastian Riedel. Improving the accuracy and efficiency of map inference for markov logic. *Network*, page 468?475, 2008.

[SD06] Parag Singla and Pedro Domingos. Memory-efficient inference in relational domains. In *Proceedings of the 21st national conference on Artificial intelligence - Volume 1*, AAAI'06, pages 488–493. AAAI Press, 2006.

[SHS03] Kevin Smyth, Holger H. Hoos, and Thomas Sttzle. Iterated robust tabu search for max-sat. In *In Proc. of the 16th Conf. of the Canadian Society for Computational Studies of Intelligence*, pages 129–144, 2003.

[SKC95] Bart Selman, Henry Kautz, and Bram Cohen. Local search strategies for satisfiability testing. In *DIMACS SERIES IN DISCRETE MATHEMATICS AND THEORETICAL COMPUTER SCIENCE*, pages 521–532, 1995.

[SLM92] Bart Selman, Hector Levesque, and David Mitchell. A new method for solving hard satisfiability problems. In *AAAI*, pages 440–446, 1992.

[TH05] Dave A. D. Tompkins and Holger H. Hoos. UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT. In Holger H. Hoos and David G. Mitchell, editors, *Theory and Applications of Satisfiability Testing: Revised Selected Papers of the Seventh International Conference (SAT 2004, Vancouver, BC, Canada, May 10–13, 2004)*, volume 3542 of *Lecture Notes in Computer Science*, pages 306–320, Berlin, Germany, 2005. Springer Verlag.