Haplotype Inference by Pure Parsimony by SAT Solver in Distributed Environment

Md SOLIMUL B. CHOWDHURY[I], Md SAKIBUL H.[II], SARDAR A. HAQUE[III]

[I] Faculty of Computer Science and Engineering Department, Sylhet International University, Bangladesh.
[II] Software Engineer, Therap BD, Bangladesh.
[III] Faculty of Computer Science and Information Technology Department, Islamic University of Technology, Bangladesh.

Summary
Recent biological researches have corroborated that gene sequence variants has a rule for the development and progression of common diseases. Some technological constraints restricts us from collecting haplotype data directly, instead we collect genotype data. To infer haplotype data from genotype data, Haplotype Inference By Pure Parsimony which minimizes the number of distinct haplotypes to explain certain number of genotype, is a good option. HIPP can be reduced to equivalent boolean satisfiablity problem. The performance of this approach depends the choice of branching rules and preprocessing steps dramatically. In this paper, we experiment on different combination of preprocessing choices and branching rules of SAT solver. This paper proposes a solution to the HIPP problem, based on this SAT model implemented on a distributed environment. Keeping the complexity of the search problem in mind, we developed SAT solver on distributed environment. And at the upshot of our work, we tested some problem instances under the combination of six branching rules and three pre-processing to give the decision which variant of the SAT model is best for HIPP.

Keywords
haplotype inference by pure parsimony, HIPP, SAT solver, distributed environment.

1. Introduction

In the last few years, a comprehensive search for genetic influences on disease involves examining all genetic differences in a large number of affected individuals. This allows to systematically test common genetic variants for their role in diseases. The next high priority phase of human genomics will involve the development of a full Haplotype Map of the human genome. The hapmap project [2] was initiated to develop a public resource that will help the researchers to find genes associated with human disease. This is the direct result of mankind's avaricious step to thrive for finding the influence of genetic variation in diseases. The achievement of this project's goal stipulates the efficient inference of haplotypes from genotypes.

As Haplotype inference has become the cynosure, ventures were undertaken for finding methods for efficient haplotype inference. There are two major approaches for solving the haplotype inference problem, which are conversant till date: combinatorial methods and statistical methods. Combinatorial methods often follow an optimization criterion [3], whereas statistical methods usually follow a model of haplotype evolution [4].

A well-known combinatorial approach to the haplotype inference problem is called Haplotype Inference by Pure Parsimony (HIPP). The goal is to find a solution to the haplotype inference problem that minimizes the total number of distinct haplotypes used. Current approaches for solving the HIPP problem utilize Integer Linear Programming (ILP) [8] and branch and bound algorithms [9].

This very paper has its contribution in four folds. Firstly, we introduce a plain SAT model for encoding the haplotype inference by pure parsimony problem. Secondly, we develop a parallel algorithm for the SAT solver for implementing the algorithm in a distributed environment for inferring the haplotypes. Thirdly, we introduce search pruning techniques for making the initial model practical for solving existing problem instances in the distributed environment. Lastly, we provide experimental results which guides us about the best combination of the search pruning methods.

We have arranged our paper as follows. The first section pertains to mathematical definition of HIPP, the second section describes SAT based Haplotype Inference and how we encode the HIPP problem instance to SAT problem instance by using plain SAT model by a polynomial time algorithm. Then the

Manuscript received August 5, 2008.
Manuscript revised August 20, 2008.

experimental result section is followed by a compact discussion on DPLL algorithm and distributed SAT solver.

2. Haplotype Inference

"Given a set G of *n* genotypes, each of length *m*, the haplotype inference problem consists in finding a set H of 2**n* haplotypes (not necessarily distinct), such that for each genotype g_i in G there is at least one pair of haplotypes (h_j,h_k), with h_j and h_k in H such that the pair (h_j,h_k) explains g_i ."

The variable *n* denotes the number of individuals in the sample(1<=i<=n). *m* denotes the number of SNP sites(1<=j<=m).. g_i denotes a specific genotype, : g_{ij} denotes a specific site *j* in genotype g_i .

When a nucleotide is altered in a DNA sequence, DNA sequence variation occurs, this DNA sequence variation is called SNP or Single Nucleotide Polymorphism. A SNP site is usually biallelic. That means an SNP site can have only two possible values [5]. In our haplotype inference problem, without loss of generality we assume some semantics. The values of the two possible alleles of each SNP are always 0 or 1.Value 0 represents the wild type and value 1 represents the mutant. A haplotypeis is a string over the alphabet {0,1}. Genotypes may be represented by extending the alphabet used for representing haplotypes to {0,1,2}.With homozygous sites being represented by values 0 or 1, depending on whether both haplotypes have value 0 or 1 at that site. Heterozygous sites being represented by value 2.

3. Mathematical Formulation of HIPP

Our approach to haplotype inference, Haplotype inference by parsimony (HIPP) has got an important feature. A solution to this problem minimizes the total number $r = |H|$ of distinct haplotypes used.Though genotypes exhibit a great diversity, experimental results provide support for this approach the number of haplotypes in a large population is typically very small [1] .

Let us consider three genotypes: 2100, 2102, 1221. From the definition of haplotype inference stated in the previous section six distinct haplotypes are expected to explain these genotypesh i.e we are expecting to get six haplotypes which have a congruence with the collected genotypes. But some times we need less than two order of magnitude of genotypes of haplotypes to explain the given genotypes.

Six haplotypes are used here. They are 0100, 1100, 0100, 1101, 1011, 1101.But only four haplotypes are unique. 0100, 1100, 1101, 1011. This observation depicts the strength of haplotype inference by pure parsimony (HIPP).

4. Encoding of HIPP Problem Instance to SAT Problem Instance

We use the same encoding of HIPP problem to SAT instacne as in [1]. In Table 1 we are showing the technique by one simple example created by arbitrary data. Let us take three genotypes, a HIPP problem instance namely A. The genotype data on that problem instance are 101, 201 and 100. Let's take r=3 to explain this genotype data.

After we convert this HIPP problem instance A into SAT problem instance B by our model showed in Table 1, it will render a set of CNFs, in this dissertation we have followed the input format from DIMACS [6].

For this conversion in Table 1 we took r=3, m=3, n=3 and required 9 *h* variables, 18 *s* variables, 2 *g* variables and 18 *v* variables total of 47 variables [1]. The list of variables are as follows:

h^1_{11} h^2_{13} h^3_{13} h^4_{21} h^5_{22} h^6_{23} h^7_{31} h^8_{32} h^9_{33} s^{a10}_{11} s^{a11}_{12} s^{a12}_{13} s^{a13}_{21} s^{a14}_{22} s^{a15}_{23} s^{a16}_{31} s^{a17}_{32} s^{a18}_{33} s^{b19}_{11} s^{b20}_{12} s^{b21}_{13} s^{b22}_{21} s^{b23}_{22} s^{b24}_{23} s^{b25}_{31} s^{b26}_{32} s^{b27}_{33} g^{a28}_{21} g^{b29}_{21} v^{a30}_{11} v^{a31}_{12} v^{a32}_{13} v^{a33}_{21} v^{a34}_{22} v^{a35}_{23} v^{a36}_{31} v^{a37}_{32} v^{a38}_{33} v^{b39}_{11} v^{b40}_{12} v^{b41}_{13} v^{b42}_{21} v^{b43}_{22} v^{b44}_{23} v^{b45}_{31} v^{b46}_{32} v^{b47}_{33}

The subscript numbers indicating the index of the variable and the superscript numbers indicating variable number used in the above table of CNFs.

After execution of the SAT solver we have got the following truth assignment of the variable for the formula to be satisfied:

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

Here variable numbers corresponds to the number on the list of variables in above list. From this truth values we can see the values of the three haplotypes are 101, 100, 001 and these haplotypes are capable to explain the genotypes in our HIPP problem instance.

Some congruence between these truth values and our encoding model [1] will be very relevant to present. Only one value of ***s*** variable in a set pertaining to one genotype is set to 1 in the truth assignment. That was our model's trick to select two

Table 1: Conversion of HIPP Problem Instance into SAT Problem Instance

HIPP Problem Instance (A)	SAT Problem Instance (B)				
	p cnf 47 140	-36 13 33 0	28 29 0	-40 20 0	-9 -18 0
	1 -10 0	36 -13 -33 0	-28 -29 0	-43 -23 0	-9 -27 0
	1 -19 0	36 16 0	-2 -11 0	-46 -26 0	32 -12 0
	4 -13 0	39 -19 0	-2 -20 0	43 20 -40 0	-32 12 0
	4 -22 0	-39 19 0	-5 -14 0	43 -20 40 0	-35 -15 0
	7 -16 0	-42 -22 0	-5 -23 0	-43 20 40 0	-38 -18 0
	7 -25 0	-45 -25 0	-8 -17 0	43 -20 -40 0	35 12 -32 0
	-2 -10 0	42 19 -39 0	-8 -26 0	46 23 -43 0	35 -12 32 0
	-2 -19 0	42 -19 39 0	3 -11 0	46 -23 43 0	-35 12 32 0
	-5 -13 0	-42 19 39 0	3 -20 0	-46 23 43 0	35 -12 -32 0
	-5 -22 0	42 -19 -39 0	6 -14 0	46 -23 -43 0	38 15 -35 0
	-8 -16 0	45 22 -42 0	6 -23 0	46 26 0	38 -15 35 0
1) 101	-8 -25 0	45 -22 42 0	9 -17 0	1 -12 0	-38 15 35 0
2) 201	3 -10 0	-45 22 42 0	9 -26 0	1 -21 0	38 -15 -35 0
3) 100	3 -19 0	45 -22 -42 0	31 -11 0	4 -15 0	38 18 0
	6 -13 0	45 25 0	-31 11 0	4 -24 0	41 -21 0
	6 -22 0	1 -28 -11 0	-34 -14 0	7 -18 0	-41 21 0
	9 -16 0	-1 28 -11 0	-37 -17 0	7 -27 0	-44 -24 0
	9 -25 0	1 -29 -20 0	34 11 -31 0	-2 -12 0	-47 -27 0
	30 -10 0	-1 29 -20 0	34 -11 31 0	-2 -21 0	44 21 -41 0
	-30 10 0	4 -28 -14 0	-34 11 31 0	-5 -15 0	44 -21 41 0
	-33 -13 0	-4 28 -14 0	34 -11 -31 0	-5 -24 0	-44 21 41 0
	-36 -16 0	4 -29 -23 0	37 14 -34 0	-8 -18 0	44 -21 -41 0
	33 10 -30 0	-4 29 -23 0	37 -14 34 0	-8 -27 0	47 24 -44 0
	33 -10 30 0	7 -28 -17 0	-37 14 34 0	-3 -12 0	47 -24 44 0
	-33 10 30 0	-7 28 -17 0	37 -14 -34 0	-3 -21 0	-47 24 44 0
	33 -10 -30 0	7 -29 -26 0	37 17 0	-6 -15 0	47 -24 -44 0
	36 13 -33 0	-7 29 -26 0	40 -20 0	-6 -24 0	47 27 0
	36 -13 33 0				

haplotypes by using two sets of ***s*** variable in our HIPP problem instance [1]. The encoding model proposed that, ***s*** variables of set A**,** will form three groups; variable number 10, 13, 16, variable number 11,14,17 and variable number 12, 15,18 pertaining to genotype number 1,2 and 3 respectively will form a group under the stipulation that each group will have exactly one variable set to 1. If we observe the values for these three groups of variable in the truth assignment then we will find the congruence with the stipulation. This is true for the other set of s values.

One more obvious congruence is that, the haplotype we will select depends on which **s** variable is set to 1. The **s** variable which is set to 1,the haplotype pertaining to that variable will be used to explain the genotype pertained to the **s** variable [1]. This is the case for both set of values of s. For explaining the first genotype that is for 101, we would need only one haplotype as it does not contain any heterozygous site. In our arbitrary example, from set A of s variable, variable number 10, 13,16 and from set B of s variable, variable number 19, 22, 25 pertaining to genotype number 1 have congruence in their values. Variable number 10 and 19 which are the first variable in both set, is set to 1. As the first variable is set to one in both set of **s** variable, the first haplotype obtained from the truth assignment is selected to explain the first genotype.

But, For genotype number 2 that is for 201, which include heterozygous site, the values of s variables for both sets are not identical. So for explaining these genotype we need different haplotypes. Variable number 11, 14, 17 from set A of s variables and variable number 20,23,26 from set B of s variables, both pertaining to genotype number 2, have different value. From this two sets only variable number 11 and variable number 26 are set to 1 respectively in the truth assignment. Variable number 11 is the first variable on set A of s and 26 is the third variable on the set B of s. So first and third haplotype obtained in the truth statement is selected to explain the second genotype.

5. SAT Solver

5.1 The Algorithm

There are two classes of high-performance algorithms for solving instances of SAT in practice: modern variants of the DPLL algorithm, such as Chaff or GRASP, and stochastic local search algorithms, such as WalkSAT. Here, we are using the general DPLL algorithm that is derived by Davis, Putnam, Logemann and Loveland. With this algorithm, we will use three preprocessing models and six branching rules separately to find out which combination is best for the SAT problem that is derived from a HIPP problem.

The DPLL/Davis-Putnam-Logemann-Loveland algorithm is a complete, backtracking-based algorithm for deciding the satisfiability of propositional logic formulae in conjunctive normal form, i.e. for solving the CNF-SAT problem.

The DPLL algorithm enhances over the backtracking algorithm by the eager use of the following rules at each step:

1) Unit Clause Propagation: Clauses of length one are called unit clauses. Trivially, if a function F includes a unit clause {u}, then every truth assignment f that satisfies F must have f (u) =1.
2) Monotone Literal Fixing: If a propositional variable occurs with only one polarity in the formula, it is called *monotone*. Monotone literals can always be assigned in a way that makes all clauses containing them true. Thus, these clauses do not constrain the search anymore and can be deleted.
3) Subsumption Rule: In the satisfiability problem, if there is such a clause that has a subset clause inside the program then that clause has to be deleted. This rule is known as subsumption rule.

5.2 Best Variant of DPLL Algorithm for HIPP Problem

Each recursive call of DPLL may invoke a choice of a literal u; algorithms for making these choices are referred to as *branching rules*. Several of the branching rules commonly used in DPLL. Among them, we used the following rules.

1) Jeroslow-Wang (1 sided) Rule
2) Jeroslow-Wang (2 sided) Rule
3) Minlen Rule
4) DSJ Rule
5) C-SAT Rule
6) Maximum Occurrence Rule

All branching rules except Maximum Occurrence Rule are well known [11]**.** As a branching rule, Maximum Occurrence Rule can be defined as follows ”branch on the variable which has the most occurrence in the working formula”. Choosing the variable which has maximum occurrences might be a good rule. We applied this rule to justify this statement.

The DPLL algorithm has three possible ingredients:

1) Unit Clause Propagation
2) Monotone Literal Fixing
3) The Subsumption Rule

The three ingredients listed here yield 2^3 different versions of what might be called the Davis-Putnam kernel (DP_Kernel) [11]. One of these eight versions, using none of these three ingredients, reduces DPLL to implicit enumeration. Each additional ingredient that is incorporated into the Davis-Putnam kernel may reduce the size of the DPLL tree, but it adds to the time spent at each node of the tree. Without experimental results, it seems hard to predict which of the combinations is likely to be most efficient.

Comparing algorithms by experiments requires a measurement of algorithmic performance. The running time is an obvious choice. However, it is not a good choice on the criterion of reproducibility, because it is

affected by the data structure, the programming language, the compiler, and the machine used; even the programmers may introduce biases through the coding process.

A measurement of algorithmic performances that is independent of these factors is the size of the DPLL tree. For an unsatisfiable formula, it is one plus twice the number of the times that the DPLL construct the branching rule. So, it seems perfect to use the size of DPLL tree for measuring algorithmic performance.

6. Distributed SAT Solver

In the previous section, at the time of finding the measurement of the algorithmic performance, we omit the run time of the program because of its instability of giving same result for different platform. But runtime can not be omitted totally for solving HIPP problem instance. Because, when HIPP problem instance is converted into the SAT problem instance the problem size increases by large magnitude, for the use of some extra variables like 'g' variables, 's' variables in the encoding algorithm which itself dilate the problem instance. The justification of the involvement of grid environment has its existence on the gigantic complexity of HIPP problem instance after converted to SAT problem instance. The parallel algorithm for SAT solver is supposed to be more convenient for HIPP because it can handle the increased complexity in a formidable way comparing to the to the traditional serial SAT solver. As we know, each branch of the DPLL tree is mutually exclusive from others for computing, so if we compute the branches in parallel and after computing if we can merge the result, then it will reduce the run time of the total execution. This idea thrived us to develop a parallel algorithm for the SAT solver and we found our idea useful.

For constructing the distributed version of the SAT Solver, we use the Alchemi [7] software and C# coding language. *Alchemi*, a .NET-based grid computing framework provides the runtime machinery and programming environment required to construct desktop grids and develop grid applications. It allows flexible application composition by supporting an object-oriented grid application programming model in addition to a grid job model.

There are four types of distributed components (nodes) involved in the construction of Alchemi grids and execution of grid applications: Manager, Executor, User & Cross-Platform Manager. The run time of the execution is proportion to the inverse of the number of the executer. That is, if we increase the number of the executer, than the run time will be decreased.

7. Experiment Result

7.1 Obtaining Real Genotype Data for SAT Solver

We obtained our real genotype data for our SAT solver from the hapmap project [10]. Genotype data that we had obtained from this source was in the form of base pair in a loci. Data from two populations (Han Chinese in Beijing, China and Japanese in Tokyo, Japan) are selected for our project. The encoder of HIPP problem instance, described on section 5, will not take this biological data as input, rather it will take a problem instance with alphabet {0,1,2}. So, we have converted the real genotype data into a data that is convenient for our encoder. p1.txt and p2.txt shown in Table 2 are the converted files.

Table 2: Files Convenient for The Encoder

Original Name of the File	Converted File Name
genotypes_ENm014.7q31.33_nonrs_CHB.txt	p1.txt
genotypes_ENm014.7q31.33_nonrs_JPT.txt	p2.txt

7.2 Experiment results

The following table, Table 3 shows the experiment result obtained by executing distributed sat solver on the above mentioned problem instances under the combination of six branching rules and three pre-processing.

Table 3: Performance Measurement of Distributed SAT Solver

Problem	Result	Level	Preprocessing Combinaton	$\|r^{opt} - r^{used}\|$	Number of Recursive Calls					
					BR 01	BR 02	BR 03	BR 04	BR 05	BR 06
p1r6n12	SAT	130	COMB 01	| 6 - 6 | = 0	371871	258751	DNF	97998	1110049	3971955
p1r6n12	SAT	130	COMB 02	| 6 - 6 | = 0	4788603	371507	DNF	125261	1160264	6587852
p1r7n12	SAT	130	COMB 01	| 6 - 7 | = 1	661365	349559	212981601	443183	402317	35625254
p1r8n12	SAT	130	COMB 01	| 6 - 8 | = 2	1800481	1053514	180699667	1822693	3135	798389848
p2r4n12	UNSAT	130	COMB 01	| 5 - 4 | = 1	205735	191801	1620977734	16327	21071	99841
p2r5n12	SAT	130	COMB 01	| 5 - 5 | = 0	1135223	1074644	DNF	19386	109489	721379
p2r5n12	SAT	130	COMB 02	| 5 - 5 | = 0	2886391	1108612	DNF	19247	119626	770141
p2r6n12	SAT	130	COMB 01	| 5 - 6 | = 1	197007	71615	4778	200094	1977	DNF

Elaboration of terms used in Table 3:

1) pArBnC = Problem instance named ‘pA’ in which ‘B’ number of haplotype is used to explain the genotypes and ‘C’ number of genotypes is used.
2) r^{opt} = Optimal ‘r’ for a specific problem
3) r^{ud} = Used ‘r’ in that specific problem
4) DNF = Did Not Finished
5) COMB 01 = Unit Clause Propagation + Monotone Literal Fixing
6) COMB 02= Unit Clause Propagation + Monotone Literal Fixing + Subsumption Rule
7) BR 01 = 1 sided Jeroslow-Wang Rule
8) BR 02 = 2 sided Jeroslow-Wang Rule
9) BR 03 = Minlen Rule
10) BR 04 = DSJ Rule
11) BR 05 = C-SAT Rule
12) BR 06 = Maximum Occurrence Rule

7.3 Analysis of experiment

From the Table 3, we have come to some decisions. Those are as follows:

1) Unit clause propagation must has to be used for solving HIPP problem. From our some experimental test, we saw that if we do not use unit clause propagation, the result becomes DNF.
2) Monotone literal minimize the number of SETVAR a great deal but it need Unit clause propagation as a helping hand.
3) Subsumption rule gives a better result where used r is optimal. If you have a look on problem p2r5n12, you can see that the combination that uses subsumption rule gives better result than the other one. But this is hypothetical.
4) We know that near the optimal r the complexity of the problem is much. And in this condition the branching rule that can create minimum SETVAR to solve the problem will be the best branching rule for our problem. Considering this condition, we find that "DSJ Rule" is the best branching rule for solving HIPP problem. You will find that in our experiment table among the 8 problems, DSJ rule produce least number of SETVAR for 5 problems to solve those problems.

8. Conclusion and Future Work

After we have taken some promising preprocessing steps such as unit, monotone and subsumption and some well known branching rules, operating on input which is taken from real haplotype data, the experimental result shows the importance of preprocessing steps and also of efficiency of the brunching rule DSJ when HIPP by SAT solver is entailed.

With this conclusive finding we can propound some future work direction in this area. Our SAT solver is chronological backtracking. A non-chronological backtracking based SAT solver can be customized with our proposed pre-processing steps. At any stage of that solver, the sub problems can be assigned priority according to value returned by our proposed branching rule DSJ. Our experiments shows that C-SAT and 2-sided jeroslow wang are promising. A more rigorous performance measurement can be done among DSJ, C-SAT and 2-sided jeroslow wang rule with some more experiments.

References

[1] Lynce, I. and Marques-Silva, "Efficient Haplotype Inference with Boolean Satisfiability". *National Conference on Artificial Intelligence (AAAI)*, Boston, USA, 2006.

[2] HapMap Project overview. http://www.genome.gov/10001688/ , 2007

[3] D. Gusfield and S.H.Orzach. "Handbook on Computational Molecular Biology", Chapter Haplotype Inference, volume 9 of Chapman and Hall/CRC Computer and Information Science Series,. CRC Press, 2005.

[4] M. Stephens, N. Smith, and P. Donelly. "A new statistical method for haplotype reconstruction from Population Data". *American Journal of Human Genetics*., vol- 68, no-4, pp. 978–989, 2001.

[5] Paola Bertolazzi, Alessandra Godi , Martine Labb´e2 and Leonardo Tininini , "Solving haplotyping inference parsimony problem using a new basic polynomial formulation". *Technical Report 561*, ULB, 2006.

[6] Center for discrete mathematics and theoretical computer science, http://www.dimacs.rutgers.edu/ , 2007

[7] Dr. Rajkumar Buyya , Krishna Nadiminti. "The Gridbus Middleware Manual". *A Manual for Girdbus Middleware*, pp. 10-16 , 2004.

[8] D. Gusfield, "Haplotype Inference by Pure Parsimony". *Proc. 14th Ann. Symp. Combinatorial Pattern Matching*, vol 2676, pp:144-155, 2003.

[9] Rui-Sheng Wang, Ling-Yun Wu, Zhen-Ping Li and Xiang-Sun Zhang, "Haplotype reconstruction from SNP fragments by minimum error correction" *Bioinformatics*. vol. 21, no. 10, pp. 2456–2462, 2005.

[10] International HapMap Project, http://www.hapmap.org/, 2007.

[11] Ming, Ouyang, "Implementations of the DPLL Algorithm", *PHD thesis.* Rutgers University, 1999.

Md. Solimul Bor Chowdhury has completed his undergrad program at the department of Computer Science and Information Technology (CIT) of Islamic University of Technology (IUT,OIC), Gazipur, Bangladesh, in 2007. Now working as a full time lecturer in Sylhet International University (SIU) , Sylhet, Bangladesh. His research interest is Haplotype Inference, algorithms of ANN.

Md. Sakibul Hasan has completed his undergrad program at the department of Computer Science and Information Technology (CIT) of Islamic University of Technology (IUT,OIC), Gazipur, Bangladesh, in 2007. Now working as a Software Engineer in Therap (BD) Ltd. His research interest is Haplotype Inference, Mathematical algorithms and Application framework on JAVA platform..

Sarder Anisul Haque has completed his undergrad program at the department of Computer Science and Information Technology (CIT) of Islamic University of Technology (IUT,OIC), Gazipur, Bangladesh, in 2002. Currently, he is a graduate student at the department of Mathematics and Computer Science of the University of Lethbridge, Alberta Canada. He is also a assistant professor (on leave) of Islamic University of Technology(IUT,OIC), Gazipur, Bangladesh. His research interest is Optimization Algorithm, Haplotype Inference and Compiler Design.