Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (2008)

Efficient Haplotype Inference with Answer Set Programming

Esra Erdem and **Ferhan Türe**
Faculty of Engineering and Natural Sciences
Sabancı University, Istanbul 34956, Turkey

Abstract

Identifying maternal and paternal inheritance is essential to be able to find the set of genes responsible for a particular disease. However, due to technological limitations, we have access to genotype data (genetic makeup of an individual), and determining haplotypes (genetic makeup of the parents) experimentally is a costly and time consuming procedure. With these biological motivations, we study a computational problem, called Haplotype Inference by Pure Parsimony (**HIPP**), that asks for the minimal number of haplotypes that form a given set of genotypes. **HIPP** has been studied using integer linear programming, branch and bound algorithms, SAT-based algorithms, or pseudo-boolean optimization methods. We introduce a new approach to solving **HIPP**, using Answer Set Programming (ASP). According to our experiments with a large number of problem instances (some automatically generated and some real), our ASP-based approach solves the most number of problems compared with other approaches. Due to the expressivity of the knowledge representation language of ASP, our approach allows us to solve variations of **HIPP**, e.g., with additional domain specific information, such as patterns/parts of haplotypes observed for some gene family, or with some missing genotype information. In this sense, the ASP-based approach is more general than the existing approaches to haplotype inference.

Introduction

Each genotype (the specific genetic makeup of an individual) has two copies, one from the mother and one from the father. These two copies are called haplotypes, and they combine to form the genotype. The genetic information contained in haplotypes can be used for early diagnosis of diseases, detection of transplant rejection, and creation of evolutionary trees. However, although it is easier to access to the genotype data, due to technological limitations, determining haplotypes experimentally is a costly and time consuming procedure. With these biological motivations, researchers have been studying haplotype inference—determining the haplotypes that form a given set of genotypes—by means of some computational methods.

One haplotype inference problem that has been extensively studied is Haplotype Inference by Pure Parsimony (**HIPP**) (Gusfield 2003). This problem asks for a minimal set of haplotypes that form the given genotypes; the decision version of **HIPP** is NP-hard (Gusfield 2003; Lancia, Pinotti, & Rizzi 2004). **HIPP** has been studied with various approaches, such as, HYBRIDIP (based on integer linear programming) (Brown & Harrower 2006), HAPAR (based on a branch and bound algorithm) (Wang & Xu 2003), SHIPS (based on a SAT-based algorithm) (Lynce & Marques-Silva 2006), and RPOLY (based on pseudo-boolean optimization methods) (Graça *et al.* 2007). In this paper we present a novel approach to solving **HIPP**, using Answer Set Programming (ASP); we call our approach HAPLO-ASP.

ASP is a declarative programming paradigm that provides a highly expressive language for knowledge representation, and efficient solvers for automated reasoning. The idea of ASP is to represent a computational problem as a "program" whose models (called "answer sets") correspond to the solutions of that problem, and to compute the answer sets for the program using an "answer set solver", like CMODELS (Giunchiglia, Lierler, & Maratea 2006), after "grounding" the program, e.g., by the "grounder" LPARSE (Simons, Niemelä, & Soininen 2002). (See (Baral 2003) for more information about ASP.)

Due to the expressive language of ASP, HAPLO-ASP allows us to solve variations of haplotype inference, where genotypes give information about present-absent genes only, where sites of genotypes are biallelic, where some parts of genotype are missing, and/or where domain-specific information, like haplotype patterns for some specific gene family, could be taken into account for more accurate solutions. In this sense HAPLO-ASP is more general than the existing haplotype inference approaches.

As for computations, we have experimented with 334 instances of **HIPP** (40 automatically generated and 294 real) using HAPLO-ASP with CMODELS, and compared our approach to the existing ones mentioned above. In these experiments, HAPLO-ASP has solved the most number of problems; next comes RPOLY. For many problems, RPOLY is faster than HAPLO-ASP. We have also experimented with an instance of Haplotype Inference from Present-Absent Genotype (**HIPAG**), prepared for 17 killer immunoglobulin-like receptor (KIR) genes; and obtained more accurate results compared to the other existing approach HAPLO-IHP (Yoo *et al.* 2007) (based on a greedy algorithm).

While solving various haplotype inference problems using ASP, we have introduced new methods to simplify the problems, to improve the computational efficiency, to find a range for the number of unique haplotypes, and to check the accuracy of the inferred haplotypes. These methods can be used by other haplotype inference approaches as well.

ASP formulations discussed below are available at the web page (HAPLO-ASP 2008).

Haplotype Inference by Pure Parsimony

A genotype is the specific genetic makeup of an individual. Each genotype has two copies, one from the mother and one from the father. These two copies are called haplotypes, and they combine to form the genotype. Due to technological limitations, we have access to genotype data rather than haplotype data. However, most of the genetic information is contained in haplotypes, which can be used for early diagnosis of diseases, detection of transplant rejection and creation of evolutionary trees. Motivated by the goal of identifying maternal and paternal inheritance to be able to map disease genes, and find the set of genes responsible for a particular disease, haplotype inference is the problem of determining the haplotypes that form a given set of genotypes.

Different pairs of haplotypes may form the same genotype, and this ambiguity makes it difficult to find the "correct" haplotypes that explain the given genotype. For that, researchers have studied a slight modification of the haplotype inference problem, *Haplotype Inference by Pure Parsimony* (**HIPP**) (Gusfield 2003)—to infer a minimal set of haplotypes that explain the given genotypes. The decision version of **HIPP** (i.e., deciding that a set of k haplotypes that explain the given genotypes exists) is NP-hard (Gusfield 2003; Lancia, Pinotti, & Rizzi 2004).

A standard definition of the concept of two haplotypes "explaining" a genotype appears in (Gusfield 2003). According to this definition, we view a genotype as a vector of sites, each site having a value 0, 1, or 2; and a haplotype as a vector of sites, each site having a value 0 or 1. The values 0 and 1 (called *alleles*) correspond to complementary bases, like `C` and `G`. The sites correspond to single nucleotide polymorphisms (SNPs). A site of a genotype is *ambiguous* if its value is 2; and *resolved* otherwise. Two haplotypes h_1 and h_2 *form (explain)* a genotype g if for every site j the following hold:

- if $g[j] = 2$ then $h_1[j] = 0$ and $h_2[j] = 1$ or $h_1[j] = 1$ and $h_2[j] = 0$;
- if $g[j] = 1$ then $h_1[j] = 1$ and $h_2[j] = 1$; and
- if $g[j] = 0$ then $h_1[j] = 0$ and $h_2[j] = 0$.

For instance, the genotype 20110 can be explained by the haplotypes 10110 and 00110.

We consider the following decision version of **HIPP**:

HIPP-DEC Given a set G of n genotypes each with m sites, and a positive integer k, decide whether there is a set H of at most k unique haplotypes such that each genotype in G is explained by two haplotypes in H.

For a sufficiently small k, a solution to **HIPP-DEC** is a solution to **HIPP** as well.

To solve, **HIPP-DEC**, we assume the following:

A1 H is a set that contains $2 * n$ haplotypes, $h_1, \ldots, h_{2n}$, and

A2 every genotype g_i in G is explained by two haplotypes, h_{2i} and $h_{2i\text{-}1}$, in H.

Then, for this problem, H is a solution if the following hold:

C1 For every genotype g in G, for every ambiguous site j of g, the values of the j'th sites of these haplotypes are different.

C2 For every genotype g in G, for every resolved site j of g, the values of the j'th site of these haplotypes are $g[j]$.

C3 There are at most k unique haplotypes in H.

Representing HIPP-DEC in ASP

Many answer set solvers, like CMODELS, have the same input language as LPARSE does; we present our formulation of **HIPP-DEC** in the input language of LPARSE.

We describe the value of the j'th site of a genotype g by atoms of the form $amb(g, j)$. Consider some answer set X describing a solution. Then

- $amb(g, j) \in X$ iff $g[j] = 2$,
- $\neg amb(g, j) \in X$ iff $g[j] = 1$, and
- $amb(g, j), \neg amb(g, j) \notin X$ iff $g[j] = 0$.

Then a set of genotypes is described by a set of literals of the forms $amb(g, j)$ and $\neg amb(g, j)$. For instance, Genotype 1 with the sites 20110 is represented in the language of LPARSE as follows:

```
amb(1,1). -amb(1,3). -amb(1,4).
```

Similarly, we describe the value of the j'th site of a haplotype i by atoms of the form $h(i, j)$. Consider some answer set X describing a solution. Then

- $h(i, j) \in X$ iff $i[j] = 1$, and
- $h(i, j) \notin X$ iff $i[j] = 0$.

Then a set of haplotypes is described by a set of atoms of the form $h(i, j)$. For instance, Haplotype 1 with the sites 10110 is described by the atoms:

```
h(1,1). h(1,3). h(1,4).
```

Suppose that we are given n genotypes, each with m sites. We represent that genotypes are labeled `1..n`, sites are labeled `1..m`, and, due to Assumption A1, haplotypes are labeled `1..2*n`, by the following domain predicates.

```
geno(1..n).  site(1..m). haplo(1..2*n).
```

First, for every haplotype `H` and `J`, a value is generated by the rules

```
{h(H,J)} :- haplo(H), site(J).
```

Then these generated haplotypes are tested with respect to Conditions C1–C3 described in the previous section, keeping in mind Assumption A2.

In the following, suppose that variables `G` and `J` denote a genotype and a site respectively:

```
#domain geno(G), site(J).
```

Condition C1 is tested by the constraints

```
% G[J]=2
:- amb(G,J), h(2*G,J), h(2*G-1,J).
:- amb(G,J), not h(2*G-1,J), not h(2*G,J).
```

which express that, for every ambiguous site `J` of every genotype `G`, the values of the haplotypes `2*G` and `2*G-1` at site `J` cannot be both `1` or `0`.

Similarly, Condition C2 is tested by the constraints

```
% G[J]=1
:- not h(2*G-1,J), -amb(G,J).
:- not h(2*G,J), -amb(G,J).
% G[J]=0
:- h(2*G-1,J), not -amb(G,J), not amb(G,J).
:- h(2*G,J), not -amb(G,J), not amb(G,J).
```

To express Condition C3, first we define unique haplotypes. We say that a haplotype labeled `H` is unique if it differs from every haplotype with a label less than `H`:

```
unique(1).
unique(H) :- H-1{diffhapp(H1,H):haplo(H1)},
   haplo(H), H>1.
```

Here `diffhapp(H1,H2)` describes that haplotypes `H1` and `H2` are different if they have different values at some site `J`:

```
diffhapp(H1,H2) :- 1{h(H1,J), h(H2,J)}1,
   haplo(H1;H2), H1<H2.
```

Then Condition C3 is tested by the constraints

```
:- k+1 {unique(H):haplo(H)}.
```

which express that the number of unique haplotypes included in an answer set can not be more than `k`.

With the program described above we can solve instances of **HIPP-DEC** using an answer set solver whose input language is the same as the input language of LPARSE. CMODELS, SMODELS (Simons, Niemelä, & Soininen 2002) and CLASP (Gebser *et al.* 2007) are three such solvers.

Solving HIPP with an Answer Set Solver

An instance of **HIPP** can be solved with the ASP program above, by trying various values for k (the number of unique haplotypes explaining the given genotypes). One possibility is, as in SHIPs, to compute a lower bound l for k, and try to solve the corresponding **HIPP-DEC** problem with $k = l, l+1, ..., 2n$ until a solution is found. In our approach, we compute an approximate lower bound l for k and an upper bound u for k, using ASP, and try to find the optimal value for k by a binary search between l and u as described below.

Computing a Lower Bound

We say that two genotypes g_1 and g_2 are *incompatible* if, for some site j, $g_1[j] + g_2[j] = 1$. Otherwise, they are *compatible*. The *incompatibility graph of a set of genotypes* is a graph where the nodes correspond to the given genotypes and edges describe the incompatibility of genotypes.

We say that a genotype g is *isolated* from a set G of genotypes, if it is not included in G, it has some ambiguous site j, and, for all genotypes that are in G and that are compatible with g, the site j is resolved in the same way.

Let G be a set of genotypes, let M be a maximal clique in the incompatibility graph of G. Let MA denote the set of all genotypes in M that have some ambiguous site, MR denote the set of all genotypes in M that have no ambiguous site, and let IG denote the set of all genotypes in G that are isolated from the clique M. We define the *contribution of a genotype $g \in G$ to the lower bound* with respect to M, denoted $lb_M(g)$, as follows: $lb_M(g) = 2$ if $g \in MA$; $lb_M(g) = 1$ if $g \in IG \cup MR$; $lb_M(g) = 0$ otherwise. Then an approximate lower bound for the number of haplotypes, k, can be computed as the maximum value of $\sum_{g \in G} lb_M(g)$. This is an approximate value for a lower bound because, when some isolated genotypes are compatible with each other and share some haplotype with a genotype in the clique, some isolated genotypes do not contribute to the lower bound.

With such an approximate value av, we check whether there is an answer set for the ASP program (with $k = av$) above. If there is not an answer set then av is a lower bound indeed, and HAPLO-ASP tries to find the optimal value for k by a binary search between av and an upper bound (computed using ASP) until an answer set is found. Otherwise, av is not a lower bound, so HAPLO-ASP tries $k = av - 1, av - 2, ...$ until no answer set is computed.

A lower bound for the number of unique haplotypes explaining the given genotypes is computed in SHIPs by means of the incompatibility graph also; according to (Lynce *et sl.* 2008), SHIPs computes an approximate size x of a maximal clique in the incompatibility graph of G (by a greedy algorithm), and some number y of genotypes isolated from that clique (relative to some heuristics), and then define the lower bound as $2 \times x - z + y$, where z is the number of genotypes in the clique whose sites are all resolved.

The computation of an approximate lower bound described above can be done using ASP. Suppose that `G,G1` denote a genotype, and `J` denotes a site. First we define incompatible genotypes:

```
incompatible(G,G1) :- -amb(G,J),
   not -amb(G1,J), not amb(G1,J).
incompatible(G1,G) :- incompatible(G,G1).
```

After that we generate a clique of genotypes that are incompatible with each other

```
{in(G)}.
:- in(G), in(G1), G<G1, not incompatible(G,G1).
```

and make sure that it is a maximal clique, by the optimization statement

```
maximize [in(G):geno(G)].
```

We define the contribution of a genotype `G` to the lower bound as the number of atoms of the form `lb(G,I)`, where `I=1,2`. For instance, a genotype in MA contributes to the lower bound by introducing `2` atoms:

```
lb(G,1) :- in(G), ambig(G).
lb(G,2) :- in(G), ambig(G).
```

and a genotype in IG contributes to the lower bound by introducing `1` atom:

```
lb(G,1) :- isolated(G).
```

Then an approximate lower bound can be computed as the maximum number of atoms of the form `lb(G,I)` included in the answer set.

We define isolated genotypes that are compatible with each other and that share some haplotype with a genotype in the maximal clique, by some rules, and decide for the existence of such genotypes by checking the presence of an atom of the form `isolatedCompatible(G)` in the computed answer set.

CMODELS does not support optimization statements, so we use CLASP instead to compute lower bounds.

Computing an Upper Bound

In a similar way, we can compute an upper bound for k, in ASP, but instead by means of the compatibility graph for the given set of genotypes. Basically, we compute the size x of a maximal clique in the compatibility graph, and the number y of genotypes that are not in the clique and whose sites are all resolved. After that we define the upper bound as $2n - x + 1 - y$. We compute upper bounds with CLASP.

Symmetry Breaking

As in other approaches to solving **HIPP**, to improve the computational efficiency we can add to our program some symmetry breaking constraints. Currently we add to our program the following constraints only

```
:- {amb(G,I):site(I):I<J}0, amb(G,J),
   h(2*G,J), not h(2*G-1,J).
```

which enforce a lexicographic order on the haplotypes.

Other Haplotype Inference Problems

Due to the expressive formalism of ASP, we can represent other variations of **HIPP** that take into account some domain-specific information, and/or that take into account biallelic sites and presence/absence of genes.

HIPP with Observed Haplotype Patterns

In haplotype studies of some gene families, sometimes some patterns may be observed. For instance, (Yoo *et al.* 2007) derived three patterns of haplotypes for the family of KIR genes of Caucasian population, from the observations in KIR haplotype studies like (Hsu *et al.* 2002). These patterns may help generating more accurate haplotypes, if included in the computation of haplotypes.

For instance a pattern may describe that the value of site `2` at all haplotypes is one; this information can be described in ASP by the constraints

```
:- not h(H,2), haplo(H).
```

Another pattern may describe that the values of sites `2` and `5` are always complementary of each other:

```
:- h(H,2), h(H,5), haplo(H).
:- not h(H,2), not h(H,5), haplo(H).
```

Such domain-specific information can be described by an ASP program, like in the examples above, and, during computation of haplotypes, it can be given to the answer set solver as an input in a separate program file.

HIPP with Observed Haplotypes

Some haplotypes can be identified (completely or partially) from the observations, as described in (Yoo *et al.* 2007). This information can be represented by an ASP program as well. Suppose that observed haplotypes are described by a set of atoms of the form `observed(O,J)` ("observed haplotype `O` is resolved at site `J`"). We can express that generated haplotypes should match these observed haplotypes, by the constraints

```
:- not matchObserved(O), observedHaplo(O).
```

Such domain-specific information can be presented to the answer set solver as an input in a separate program file.

Haplotype Inference from Present-Absent Genotype

Haplotype Inference from Present-Absent Genotype (**HIPAG**) is another haplotype inference problem, which asks for the minimal set of haplotypes "compatible" with the given genotypes. Both haplotypes and genotypes can be viewed as vectors, as in **HIPP**. In this problem, each site of a haplotype takes one of the two values 0 or 1 specifying the presence/absence of a particular gene. Sites of genotypes are biallelic; the value of each site is a pair of numbers from $\{0, 1, ?\}$. For instance, $(1, ?)(1, ?)(1, 1)$ is a genotype.

For a genotype g of the form $(g_{11}, g_{12})...(g_{m1}, g_{m2})$, let us denote by g^1 the vector $g_{11}g_{21}...g_{m1}$ and by g^2 the vector $g_{12}g_{22}...g_{m2}$. We say that two alleles i and j are *compatible* if they are identical or if one of them is ?. Two haplotypes h_1 and h_2 are *compatible* with a genotype g, all with m sites, if, for every site j, $h_1[j]$ is compatible with one of the two alleles, $g^1[j]$ or $g^2[j]$, and $h_2[j]$ is compatible with the other. For instance, the haplotypes 011 and 111 are compatible with $(1, 0)(1, 1)(1, 1)$ and $(1, ?)(1, ?)(1, 1)$ but not with $(1, ?)(1, 0)(1, 1)$. Note that, we can discard the sites with missing information while computing a solution for **HIPAG**.

We consider the following decision version of **HIPAG**:

HIPAG-DEC Given a set G of n genotypes each with m biallelic sites, and a positive integer k, decide that there is a set H of at most k unique haplotypes such that each genotype in G is compatible with two haplotypes in H.

For a sufficiently small k, a solution to **HIPAG-DEC** is a solution to **HIPAG** as well.

To solve **HIPAG-DEC**, we assume A1 and A2 as well. Then, for this problem, a set H of haplotypes is a solution if the following hold:

H1 For every genotype g in G, for every site j of g, the values of the j'th sites of the corresponding haplotypes h_1 and h_2 are compatible with the values of $g^1[j]$ and $g^2[j]$.

H2 There are at most k unique haplotypes in H.

The ASP representation of **HIPAG-DEC** is similar to the ASP representation of **HIPP-DEC**: We generate values for the sites of haplotypes and compute the maximum number of unique haplotypes with the same set of rules. The ASP representation of **HIPAG-DEC** is different from that of **HIPP** in two ways. First genotypes are represented in terms of atoms of the form `present(G,J,I)` ("Chromosome `I` of Genotype G has value `1` at Site `J`") describing present genes; `-present(G,J,I)` describes absence of genes. For instance, Genotype 3 of the form $(1, ?)(1, 0)(1, 1)$ is represented as follows:

```
present(3,1,1). present(3,2,1). present(3,3,1).
-present(3,2,2). present(3,3,2).
```

Second difference, due to the nature of genotype data and the definition of compatibility, is the constraints expressing the compatibility of alleles (Condition H1). For instance, we can express that if both genes at some site are present then both haplotypes should have value `1` at that site:

```
:- {h(2*G-1,J), h(2*G,J)}1, bothPresent(G,J).
```

Similarly, we can express that if at least one of the two sites one gene is present then at least one of the haplotypes has the value `1` at that site:

```
:- {h(2*G-1,J), h(2*G,J)}0, onePresent(G,J).
```

With the program whose some parts are described above, HAPLO-ASP can solve problem instances of **HIPAG-DEC**. The number of haplotypes (k) compatible with n given genotypes is between 1 and $2n$; the optimal value for k is found by binary search between these two bounds.

As with **HIPP-DEC**, we can add domain-specific information to the program above as well.

Experimental Results

We have implemented a haplotype inference system, also called HAPLO-ASP, based on the ASP-based approach above; it is a PERL script including system calls to answer set solvers. We have performed two groups of experiments using this system. In the first group of experiments, we have considered various problem instances of **HIPP** and compared HAPLO-ASP with the other state-of-the-art haplotype inference systems, RPOLY (Graça *et al.* 2007) (based on pseudo-boolean optimization methods), SHIPS (Lynce & Marques-Silva 2006) (based on a SAT-based algorithm) and HAPAR (Wang & Xu 2003) (based on a branch and bound algorithm); these systems can solve **HIPP** problems only. We have excluded from our experiments the systems based on integer linear programming, such as HYBRIDIP (Brown & Harrower 2006), since the systems above perform much better (Lynce & Marques-Silva 2006; Graça *et al.* 2007).

The second group of experiments is about **HIPAG**. We have compared HAPLO-ASP with the haplotype inference system HAPLO-IHP (Yoo *et al.* 2007) with respect to the computation time and the accuracy of generated haplotypes. HAPLO-IHP is based on statistical methods and it can compute approximate solutions to instances of **HIPAG** only. The other haplotype inference systems can not solve **HIPAG**.

Setting the Stage for Experiments

In these experiments, the executable for SHIPS is obtained from their authors. RPOLY (executables), HAPAR (source files) and HAPLO-IHP (source files) are available at their web pages. These systems do not have any version number; we use the most recent versions as of January 28, 2008. In the experiments, as their search engines, RPOLY uses MINISAT+ (Version 1.0), SHIPS uses MINISAT (Version 2.0), and HAPLO-ASP uses CMODELS (Version 3.74) with LPARSE (Version 1.0.17) and MINISAT (Version 2.0). In our experiments, we have used a workstation with 1.5GHz Xeon processor and 4x512MB RAM, running Red Hat Linux(Version 4.3).

Table 1: Number of problems solved, with a timeout of 1000 sec.s for each problem

Group of problems	# of problems	# of problems solved		
		SHIPS	HAPLO-ASP	RPOLY
abcd	90	90	90	90
ace	90	90	90	90
hapmap	24	24	23	23
ibd	90	78	89	88
unif	20	20	20	20
nonunif	20	20	20	20

Experimenting with HIPP Problems

We have experimented with 334 instances of **HIPP**, used also in the experiments of (Brown & Harrower 2006; Lynce & Marques-Silva 2006; Graça *et al.* 2007): 40 instances generated using MS of (Hudson 2002) (20 *uniform*, 20 *nonuniform*), 294 real instances (24 *hapmap*, 90 *abcd*, 90 *ace*, 90 *ibd*). (These datasets are explained in detail in the cited articles above.) All problem instances are simplified by eliminating duplicates (of genotypes and haplotypes) as described in (Lynce & Marques-Silva 2006; Graça *et al.* 2007) before our experiments.

For each haplotype inference system we have assigned 1000 sec.s of CPU time to solve each problem. Table 1 shows the number of problem instances solved by each system. According to this table, HAPLO-ASP solves the most number of problems (332 out of 334 problems). RPOLY aborts for only one problem that HAPLO-ASP solves, but in most of the problems for which it computes a solution in 1000 sec.s it is faster than HAPLO-ASP by a magnitude of up to 300. SHIPS aborts on 12 problems out of 334, but can solve a problem that both HAPLO-ASP and RPOLY can not solve in 1000 sec.s. We have also tried HAPAR (Wang & Xu 2003), and obtained results similar to those in (Lynce & Marques-Silva 2006): SHIPS, RPOLY, HAPLO-ASP perform better. For instance, HAPAR can solve 11 out of 90 for *ibd* problems, and 78 out of 90 *ace* problems.

Although the computation of an upper bound by HAPLO-ASP does not take much time (usually less than a second), the computation of a lower bound sometimes takes quite some time. For instance, for *ibd_50.04*, HAPLO-ASP spends 96.81 sec.s for computing a lower bound, 0.05 sec.s for computing an upper bound, and 648.45 sec.s to compute a solution afterwards. This specific *ibd* problem is one of the most difficult problems in our data set; a solution cannot be computed by SHIPS or RPOLY in 1000 sec.s.

Both SHIPS and HAPLO-ASP use MINISAT as their search engine. Usually the propositional theory prepared for MINISAT by SHIPS is smaller than the one prepared by HAPLO-ASP. For instance, for *ibd_50.04*, the theory prepared by SHIPS for $k = 51$ contains 16161 variables and 159227 clauses; the theory prepared by HAPLO-ASP for $k = 55$ contains 441477 variables and 1458105 clauses.

Experimenting with HIPAG Problems

We have compared HAPLO-ASP with the haplotype inference system HAPLO-IHP (Yoo *et al.* 2007), on one of the data sets generated by Yoo et. al for 17 KIR genes of Caucasian population. This data set contains 200 genotypes with 14 biallelic sites. Yoo et al. derived three patterns of haplotypes for this family of genes from the observations in KIR haplotype studies like (Hsu *et al.* 2002). As in our experiments with **HIPP** problems, we have first simplified this data set by eliminating the duplicates, and modified the haplotype patterns accordingly. After eliminating the two genotypes that do not match any patterns, the simplified data set contains 28 genotypes with 11 sites.

We also have measured the accuracy of the inferred haplotypes H' by checking how much they match the original haplotypes H. For every inferred haplotype $h' \in H'$ and for every original haplotype $h \in H$, both with m sites, let $f(h', h)$ be the number of sites at which h and h' have the same value. Then the accuracy rate can be defined as

$$\frac{\sum_{h' \in H'} \max_{h \in H} f(h', h)}{|H'| \times m}.$$

Without the given haplotype patterns, no solution can be found in 30 minutes by HAPLO-IHP; whereas HAPLO-ASP finds 11 haplotypes in 57.08 CPU sec.s, with the accuracy rate 0.702 (by performing a binary search between 1 and 56.) With the given haplotype patterns, HAPLO-IHP finds 19 haplotypes compatible with the given genotypes in 9.1 CPU sec.s, with the accuracy rate 0.732; whereas HAPLO-ASP finds 11 haplotypes in 640 CPU sec.s, with the accuracy rate 0.768. Here 640 sec.s include 1.4 sec.s to infer the set of haplotypes (for $k = 11$), and 631 sec.s to verify its minimality (for $k = 10$).

HAPLO-ASP computes an exact solution to **HIPAG**, whereas HAPLO-IHP computes an approximation with a greedy algorithm; this explains the difference between the computation times as well the higher accuracy rate of HAPLO-ASP's solution. As expected, adding patterns improves the accuracy rate of HAPLO-ASP.

Conclusion

We presented HAPLO-ASP, an ASP-based approach to solving various haplotype inference problems, such as **HIPP** and **HIPAG**, possibly by taking into account the given domain-specific information. This is the first haplotype inference approach/system that can solve all these variations of haplotype inference.

While solving these problems using ASP, we have introduced new formulations of these problems, as well as new methods to computing a lower bound and an upper bound for the number of unique haplotypes to be inferred in a **HIPP** problem. We have also introduced a new method to simplify **HIPAG** problems, and to check the accuracy of the inferred haplotypes. These methods can be implemented in other haplotype inference systems as well.

According to our experiments with various instances of **HIPP**, HAPLO-ASP solved more number of problems compared to the state-of-the-art solvers, such as RPOLY, SHIPS, HAPAR. HAPLO-ASP solved the **HIPAG** instance generated for KIR genes, with a higher accuracy compared to HAPLO-IHP, the only other system that can compute solutions for HIPAG problems.

Some improvements to HAPLO-ASP could be to add more symmetry-breaking constraints and to decrease program size by some preprocessing step, as implemented in the other existing systems; this is a part of our ongoing work.

Acknowledgements

Pierre Flener, Serdar Kadıoğlu, Uğur Sezerman told us their experiences with the existing haplotype inference systems. Inés Lynce and Joao Marques-Silva sent us the executables for SHIPS and answered our questions. Dan Brown, Ian Harrower, Inés Lynce sent us the benchmarks. Yun Joo Yoo and Kui Zhang answered our questions about HAPLO-IHP.

References

Baral, C. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

Brown, D., and Harrower, I. 2006. Integer programming approaches to haplotype inference by pure parsimony. *IEEE/ACM TCBB* 3:348–359.

Gebser, M.; Kaufmann, B.; Neumann, A.; and Schaub, T. 2007. Conflict-driven answer set solving. In *Proc. of IJCAI*, 386–392.

Giunchiglia, E.; Lierler, Y.; and Maratea, M. 2006. Answer set programming based on propositional satisfiability. *Journal of Automated Reasoning* 36(4):345–377.

Graça, A.; Marques-Silva, J. P.; Lynce, I.; and Oliveira, A. 2007. Efficient haplotype inference with pseudo-boolean optimization. In *Proc. of Algebraic Biology*.

Gusfield, D. 2003. Haplotype inference by pure parsimony. In *Proc. of CPM*, 144–155.

HAPLO-ASP. 2008. `http://people.sabanciuniv.edu/~esraerdem/haplo-asp.html`

Hsu, K. C.; Chida, S.; Geraghty, D. E.; and Dupont, B. 2002. The killer cell immunoglobulin-like receptor (KIR) genomic region: gene-order, haplotypes and allelic polymorphism. *Immunological Reviews* 190(1):40-52.

Hudson, R. 2002. Generating samples under a wrightfisher neutral model of genetic variation. *Bioinfo.* 18:337-338.

Lancia, G.; Pinotti, M. C.; and Rizzi, R. 2004. Haplotyping populations by pure parsimony: Complexity of exact and approximation algorithms. *INFORMS Journal on Computing* 16(4):348–359.

Lynce, I., and Marques-Silva, J. 2006. Efficient haplotype inference with boolean satisfiability. In *Proc. of AAAI*.

Lynce, I.; Marques-Silva, J.; and Prestwich, S. 2008. Boosting haplotype inference with local search. *Constraints* 13(1), 2008.

Simons, P.; Niemelä, I.; and Soininen, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence* 138:181–234.

Wang, L., and Xu, Y. 2003. Haplotype inference by maximum parsimony. *Bioinformatics* 19(14):1773-1780.

Yoo, Y. J.; Tang, J.; Kaslow, R. A.; and Zhang, K. 2007. Haplotype inference for presentabsent genotype data using previously identified haplotypes and haplotype patterns. *Bioinformatics* 23(18):2399–2406.