Comparing ASP, CP, ILP on Haplotype Inference by Pure Parsimony

Ferhan Türe and Esra Erdem

Faculty of Engineering and Natural Sciences
Sabancı University, Istanbul 34956, Turkey

Abstract. We study three declarative programming paradigms, Answer Set Programming (ASP), Constraint Programming (CP), and Integer Linear Programming (ILP), on a challenging application: Haplotype Inference by Pure Parsimony (HIPP). We represent HIPP in each formalism in a systematic way, compare the formulations both from the point of view of knowledge representation (e.g., how easy it is to represent some concepts and constraints) and from the point of view of computational efficiency (in terms of computation time and program size). We discuss possible ways of improving the computational efficiency, and other reformulations of the problems based on different mathematical models. Some formulations suggest new approaches to solving HIPP. In particular, the ASP-based approach solves the most number of problems.

1 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. Due to technological limitations, we have access to genotype data rather than haplotype data. 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 by Pure Parsimony (**HIPP**) is the problem of determining a minimal set of haplotypes that explain the given genotypes; the decision version is NP-hard [1, 2].

We consider the decision problem corresponding to **HIPP** and represent it in three declarative programming paradigms, Answer Set Programming (ASP), Constraint Programming (CP), and Integer Linear Programming (ILP), in a systematic way: First we describe the constraints in a metalanguage, as precise and straightforward as possible, and then formulate each constraint in each formalism. We compare these formulations both from the point of view of knowledge representation (e.g., how easy it is to express cardinality constraints) and from the point of view of computational efficiency (i.e., in terms of CPU time and program size). We discuss possible ways of improving the computational efficiency (e.g., by symmetry breaking, introducing auxiliary variables, and adding redundant constraints), and other reformulations of the problem based on different mathematical models. In our experiments, we use the ASP solver CLASP[1] and the commercial CP/ILP solver ILOG OPL.[2]

[1] `http://www.cs.uni-potsdam.de/clasp/`
[2] `http://www.ilog.com/products/solver`

A similar study that compares ASP and CP/ILP is [3]: the authors compare the computation times of some solvers on some problems, relative to various reformulations (e.g., by symmetry breaking, and adding auxiliary variables). Our work can be seen as a continuation and/or a complement of this study. On the other hand, our paper not only reports a comparison of ASP, CP, ILP based on our experiences, but also introduces new approaches and formulations to solving haplotype inference. For instance, the ASP-based approach to **HIPP** solves the most number of problems in our set of benchmarks including both artificially generated problems and real problems [4].

Due to space limitation, we refer the readers to [5–7] for more information about ASP, ILP, and CP. All formulations and experimental results are available at [8].

2 Haplotype Inference by Pure Parsimony

Haplotype Inference with Pure Parsimony (**HIPP**) [1] asks for a minimal set of haplotypes that "explain" a given set of genotypes. A standard definition of the concept of two haplotypes "explaining" a genotype appears in [1]. 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. 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 problem 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 k 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. For this problem, H is a solution if the following hold:

C1 Every genotype g in G is mapped to two haplotypes in H.
C2 For every genotype g in G, for every ambiguous site j of g, the values of the j'th sites of the corresponding two haplotypes are different.
C3 For every genotype g in G, for every resolved site j of g, the values of the j'th sites of the corresponding two haplotypes are $g[j]$.

3 Formulations of HIPP-DEC

In the language of OPL, the given set of genotypes are declared as a matrix of size nm, and the set of k haplotypes to be inferred is declared as a matrix of size km. In the CP representation, we use functions $s[1, i]$ and $s[2, i]$ to describe the first and second haplotype explaining Genotype i. In the ILP representation, array indices cannot be decision variables, so we introduce the variables $s[i_1, i_2, i]$ to describe the explanation of Genotype i by Haplotypes i_1 and i_2. In the ASP representation, since functions are not allowed, we describe the value of the j'th site of a genotype g by atoms of the form $amb(g, j)$; due to the presence of both classical negation and default negation, these atoms are sufficient to represent the three values of sites. Similarly, we describe the value of the j'th site of a haplotype i by atoms of the form $h(i, j)$. Then the formulations of **HIPP-DEC** in the language of OPL and LPARSE follow from the problem description.

We compare these formulations both from the point of view of representation, and from the point of view of computational efficiency.

Representing **HIPP** All three formalisms have special constructs to solve optimization problems. For instance, in the CP formulation, after we change the size of the haplotype matrix to $2nm$, and the mapping of genotypes to haplotypes accordingly, we can add the following minimization statement:

```
minimize max(i in 1..2,j in 1..n) s[i][j];
```

In the ILP formulation, after we ensure an increasing order of indices for two haplotypes explaining a genotype, we can add the minimization statement:

```
minimize max(i1 in 1..2*n,i2 in 1..2*n,i in 1..n) i2*s[i1,i2,i];
```

As for the ASP encoding, after modifying the range of haplotypes to $1..2n$, we can add the minimization statement

```
minimize [mapped(H):haplo(H)].
```

where `mapped` describes the haplotypes that explain some genotype. However, solving **HIPP** instances directly is difficult for the solvers (cf. the results at [8]); therefore we compute a solution to **HIPP** by iteratively solving **HIPP-DEC** to find the minimum k.

Symmetry breaking constraints and auxiliary atoms/variables Two haplotypes are different if they have a site with different values; otherwise they are identical. To improve computational efficiency, we add to the formulations of **HIPP-DEC** the following constraint to ensure that the haplotypes in the inferred set H of haplotypes are unique: (S1) Every haplotype h in H is different from all other haplotypes in H. Alternatively, instead of the constraint S1, we can consider a different formulation which eliminates some of the symmetries only: the idea is (S2) not to generate a haplotype i unless haplotype $i - 1$ is already generated. However, adding to the formulations of **HIPP-DEC** either of these symmetry breaking constraints does not generally improve the computational efficiency; on the other hand, the symmetry breaking improves the computational efficiency when added to the formulations of **HIPP**.

Other formulations Another definition of **HIPP** (and thus **HIPP-DEC**), tuned for ILP, is due to [9]: a genotype is *ambiguous* if its value is 1 and *resolved* otherwise; and two haplotypes h_1 and h_2 *form (or explain)* a genotype g if for every site j the following hold: $g[j] = h_1[j] + h_2[j]$. Then, a set H of k haplotypes is a solution to **HIPP-DEC** if, for every genotype g in G, there exist two haplotypes h_1 and h_2 in H such that, for

Table 1. Experimental results for **HIPP-DEC** wrt Gusfield's definition: computation times

Problem	n	m	k	Straightforward formulation			Alternative formulation		
				CP – OPL	ASP – CLASP	ILP – OPL	CP – OPL	ASP– CLASP	ILP – OPL
I0	10	41	16	1.65	3.17	-	0.71	1.01	1.03
			15	-	-	-	20.25	1.01	133.37
I1	10	41	15	0.82	2.63	-	1.25	0.96	1.52
			14	-	-	-	11.35	0.99	1.28
I2	15	48	22	102.29	10.99	-	3.64	5.58	4.05
			21	-	-	-	-	8.74	-
I3	20	54	26	-	608.99	-	58.03	11.07	-
			25	-	-	-	-	109.84	-

Table 2. Experimental results **HIPP-DEC** wrt Gusfield's definition (straightforward formulation): program size

Problem	n	m	k	CP – OPL		ASP – CLASP		ILP – OPL	
				# of var.s	# of constraints	# of var.s	# of constraints	# of var.s	# of rules
I0	10	41	16	4416	1333	88138	87796	-	-
I1	10	41	15	4465	1369	70457	70135	-	-
I2	15	48	22	7551	2299	302462	301770	-	-
I3	20	54	26	11194	3469	612646	611564	-	-

every site j, $g[j] = h_1[j] + h_2[j]$. In another approach, we can get rid of large matrices describing mappings between haplotypes and genotypes; and enforce that each genotype i be explained by haplotypes $2i$ and $2i - 1$. With these alternatives, we have represented **HIPP** and **HIPP-DEC** in 16 different ways, in each declarative programming paradigm. All formulations are available at [8]. Out of the 16 formulations, the one that eliminates large matrices leads to faster computations.

Computational efficiency With the formulations above, we tried to solve four problem instances of **HIPP**, randomly chosen amongst the ones tested in [9], using CLASP 1.0.4[3] and ILOG OPL 5.5 with CP-OPTIMIZER 1.1, on a machine with Intel Centrino 1.8GHz CPU and 1 GB of RAM running on Windows XP. For each instance of **HIPP**, we considered two instances of **HIPP-DEC**: one with the optimal value of k, and the other with 1 less than the optimal.

Table 1 compares the computation time (CPU time in sec.s), and Table 2 compares the program size (number of variables and constraints in the case of OPL, and number of variables and rules in the case of CLASP). Two sorts of formulations are considered in Table 1: a straightforward representation of Conditions C1–C3 and the alternative formulation mentioned above. In Table 1, a dash - indicates that the problem could

[3] The answer set solver CMODELS solves **HIPP** instances faster than CLASP [4]. Due to the incompatibility of CMODELS with Windows XP, and since we do not have license for a version of OPL for Linux, we used CLASP in our experiments.

not be solved in 900 sec.s. For instance, consider Problem I0. A set of 16 haplotypes that explain the given genotypes is computed in 1.65 sec.s using OPL with the CP formulation (straightforward formulation); with the same formulation and the solver, it can not be verified in 900 sec.s that there is no set of 15 haplotypes that explain the given genotypes. In the computation of the former instance, the theory contains 4416 variables and 1333 constraints. The ILP formulations of problems have larger program sizes, mainly due to the larger size of matrices s (since indices cannot be decision variables as in CP). These problems are too large for the straightforward formulation of ILP, none of them could be compiled into a theory in 900 sec.s.

With the alternative definition of **HIPP-DEC** due to [1], with the CP and ASP formulations, the computation time decreases (at least by a factor of 2). For instance, it can be verified that I1 does not have a solution with $k = 14$, in 11.35 (resp. 0.99) sec.s with the alternative CP (resp. ASP) formulation of **HIPP-DEC**; neither OPL nor CLASP can verify this in 900 sec.s with the straightforward formulations. With the alternative ILP formulation, OPL can solve most of the problems (5 out of 8 in Table 1), even though it did not terminate for any of them with the straightforward ILP formulation [8].

4 Conclusion

We studied **HIPP** in the context of three declarative programming paradigms, ASP, CP, ILP, usually used by different research communities in AI/CP/OR to solve such hard problems. We observed some differences in the representations due to specifics of the input languages of the solvers. For instance, in CP formulations, OPL allows us to declare indices of a matrix as decision variables, and, as observed in our experiments with **HIPP-DEC** this leads to less number of atoms compared to ILP formulations. In all formalisms one can express constraints on the cardinality of a set: in CP and ILP one can use summation, and in ASP one can use a cardinality expression.

Some differences are due to the expressivity of the formalisms. For instance, to express the uniqueness of haplotypes generated so far, in CP/ILP, $2k^2$ auxiliary variables are introduced. In ASP, a recursive definition of k atoms suffices. In the formalization of **HIPP-DEC**, due to the presence of negation as failure in ASP, it is sufficient to introduce nm atoms to describe the given genotypes; in the CP/ILP formulations, genotypes are described by $3nm$ atoms. On the other hand, being able to represent functions and matrices in CP and ILP helps us formulate some problems more concisely, as observed in the formalization of **HIPP-DEC** relative to [9].

Besides comparing ASP, CP, ILP on **HIPP**, we introduced new approaches and formulations to solve it, sometimes inspired by the representations in other formalisms. For instance, the ASP-based approach to HIPP-DEC (alternative formulation, with Gusfield's definition) solves the most number of problems compared to the existing haplotype inference systems based on SAT/ILP/PBO methods, as reported at [4].

Acknowledgments

We heard about **HIPP** from Pierre Flener and Uğur Sezerman. Pierre, Uğur, and Serdar Kadıoğlu shared with us their experiences with the existing haplotype inference systems. Inês Lynce and Joao Marques-Silva provided us with the executables for SHIPs and patiently answered our questions about it. Daniel Brown, Ian Harrower, and Inês Lynce kindly provided us with some benchmarks. Martin Gebser and Benjamin Kaufmann helped us with the use of CLASP to solve optimization problems, and made useful suggestions to us about some of the ASP formulations. Elvin Çoban helped us with the use of OPL. Yuliya Lierler provided us with useful comments on an earlier draft of this note.

References

1. Gusfield, D.: Haplotype inference by pure parsimony. In: Proc. of CPM. (2003) 144–155
2. Lancia, G., Pinotti, M.C., Rizzi, R.: Haplotyping populations by pure parsimony: Complexity of exact and approximation algorithms. INFORMS Jour. on Computing **16**(4) (2004) 348–359
3. Cadoli, M., Mancini, T., Micaletto, D., Patrizi, F.: Evaluating asp and commercial solvers on the csplib. In: Proc. of ECAI. (2006) 68–72
4. Türe, F., Erdem, E.: Efficient Haplotype Inference with Answer Set Programming. `http://people.sabanciuniv.edu/~esraerdem/haplo-asp.html`
5. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
6. Hoffman, K.L., Padberg, M.: Encyclopedia of Operations Research and Management Science. Kluwer, Massachusetts (2001)
7. Rossi, F., van Beek, P., Walsh, T.: Handbook of Constraint Programming. Elsevier (2006)
8. `http://people.sabanciuniv.edu/~esraerdem/benchmarks/hipp.html`
9. Brown, D., Harrower, I.: Integer programming approaches to haplotype inference by pure parsimony. IEEE/ACM TBCB **3** (2006) 348–359