Comparing Integer Linear Programming to SAT-Solving for Hard Problems in Computational and Systems Biology

Hannah Brown¹ , Lei Zuo^{1,2} , and Dan Gusfield¹

¹ Department of Computer Science, University of California,
Davis 1 Shields Avenue, Davis, CA 95616, USA
{hsbrown, leizuo, dm Gusfield}@ucdavis.edu

² Department of Computer Science, The University of Hong Kong,
Pok Fu Lam, Hong Kong, China

Abstract. It is useful to have general-purpose solution methods that can be applied to a wide range of problems, rather than relying on the development of clever, intricate algorithms for each specific problem. Integer Linear Programming is the most widely-used such general-purpose solution method. It is successful in a wide range of problems. However, there are some problems in computational biology where integer linear programming has had only limited success. In this paper, we explore an alternate, general-purpose solution method: SAT-solving, i.e., constructing Boolean formulas in conjunctive normal form (CNF) that encode a problem instance, and using a SAT-solver to determine if the CNF formula is satisfiable or not. In three hard problems examined, we were very surprised to find the SAT-solving approach was dramatically better than the ILP approach in two problems; and a little slower, but more robust, in the third problem. We also re-examined and confirmed an earlier result on a fourth problem, using current ILP and SAT-solvers. These results should encourage further efforts to exploit SAT-solving in computational biology.

Keywords: Integer programming · SAT-solving · Computational biology

1 Introduction

Integer (Linear) Programming, abbreviated “ILP”, is a versatile modeling and optimization technique that has been increasingly used in *computational and systems biology* in *non-traditional* ways. ILP is often (but not always) very effective in solving *instances* of hard computational problems on *realistic* biological

Thanks to NSF for supporting this research under the grant 1528234. Support for H.B. was from NSF Research Experiences for Undergraduates grant 1528234. Thanks also to Chase Maguire for his contributions at the start of this project, and thanks to the very helpful and in-depth conference reviews we received.

© Springer Nature Switzerland AG 2020

C. Martin-Vide et al. (Eds.): AICoB 2020, LNBI 12099, pp. 63–76, 2020.

https://doi.org/10.1007/978-3-030-42266-0_6

data of current importance. See [8] for an in-depth discussion of ILP in computational and systems biology, illustrating many successes (and some failures) of the ILP approach in computational and systems biology.

Despite the many successes of ILP in computational and systems biology, in the testing of ILP formulations during the writing of [8], we observed a few computational biology problems where the ILP approach was only moderately effective, or was almost completely ineffective. For such hard problems, it is of interest to try different well-developed, general, computational techniques. The most well-developed such technique involves formulating a computational problem as a Boolean formula in Conjunctive Normal Form (CNF), and using a *SAT-solver* to determine whether that CNF formula is satisfiable. We wrote programs to create CNF-formulations for four hard computational biology problems, and tested these using the widely-used SAT-solver, *pLingeling* [3]. We wanted to see if this SAT approach could solve problem instances that were difficult for the ILP approach, using the highly-regarded ILP solver created by *Gurobi Optimization*. We then did some additional testing using another highly-regarded SAT-solver, *Glucose-Syrup*, to check that our results were not just artifacts of using pLingeling.

Previous comparisons of ILP and SAT found problems (not from computational biology) where the ILP approach was highly efficient but the SAT approach was ineffective [11]. In contrast, two other comparisons (for problems in computational biology) showed that the SAT approach was much faster than the ILP approach [15–17].

In this Paper. We report on the development of SAT formulations, and in-depth empirical comparisons of ILP and SAT-solving for *three* hard problems, where prior ILP approaches were much less effective than desired: *protein folding* under the *HP model*; *transforming gene orders by reversals*; and computing the *History Bound* on the number of recombinations needed to generate a given set of SNP sequences. We also re-implemented and confirmed an earlier published result on the use of SAT-solving for the problem of *haplotyping by pure parsimony*. All of the software we developed is freely available online, as are the SAT-solvers we used, *pLingeling* (mostly) and *Glucose-Syrup*. The ILP-solver we used, from Gurobi Optimization, is available with a free academic license.

Results. In the protein folding problem, the ILP approach was seen to be generally faster than the SAT approach, with some notable exceptions. But in the other two newly-examined problems (reversals and History Bound), the SAT approach was substantially superior to the ILP approach in terms of solution times, and in the avoidance of extreme behaviors that the ILP approach sometimes exhibits. Also, our reexamination of pure-parsimony haplotyping confirmed the results in [15, 16], and found extreme cases where SAT solved in a practical amount of time, but the ILP approach made little progress towards a solution.

We examine each of these hard problems in the next sections. We start with the protein folding problem, presenting the full logic and details of the CNF formulas in order to illustrate the SAT approach. Space limits the full details of

the CNF formulas in discussing the other three problems. These will be detailed in an expanded paper to be written.

2 Protein Folding via the HP Model

Determining the three-dimensional structure of proteins, or learning some parts of the structure, are critical tasks in biochemistry, biophysics, computational biology, and systems biology. For computational effectiveness, an attractive simple model of globular protein folding, the *HP model*, was proposed by Dill [5] in 1995, and has been extensively explored since then. The model simplifies the twenty standard amino acids by dividing them into *two* groups: the *hydrophobic* (H) and the *hydrophilic* (P); that is, water-fearful and water-loving. Hence a protein sequence, based on an alphabet of size twenty, is reduced to a binary sequence (H or P). For the biological motivation for this model, see [8].

The HP Prototein Model and Folding Problem in 2-D. A *prototein*¹ is a binary sequence that we embed on a *two-dimensional grid*, where each 1 encodes an H and each 0 encodes a P. A *legal embedding* of a prototein on the grid must satisfy the following rules:

1. Each character in the sequence gets assigned to some point on the grid.
2. No character in the sequence gets assigned to more than one point on the grid.
3. Each point of the grid gets assigned *at most* one character in the sequence.
4. Two adjacent characters in the sequence must be placed on two points that are *neighbors* on the grid in either the horizontal or vertical direction, but not both.

These four rules mean that the sequence must be embedded into the grid as a *self-avoiding walk*, without deforming the sequence.

In an embedding of the sequence, we say that a *potential contact* exists between neighboring points j and m on the grid G if and only if a 1 is assigned to both points j and m of G . A potential contact (j, m) is a (real) *contact* if the 1s that are assigned to points j and m are *not* adjacent in the sequence. The central assumption in the HP model is that the most stable fold is one that maximizes the number of contacts (H-H bonds). This leads to:

The 2-D Prototein Problem. Given a binary sequence S , find a legal embedding of the sequence in the 2-D grid, G , to maximize the number of contacts.

We define the *offset* as the number of adjacent positions in sequence S that both have character 1. Clearly, the number of contacts in an embedding is the number of potential contacts minus the offset. Since we can easily determine the offset given S , the problem of maximizing the number of contacts can be solved by maximizing the number of potential contacts, and that is the approach we take. The 2-D Prototein Problem has been naturally generalized to a 3-D grid, which better models real protein folding.

¹ Yes, the word is “prototein” not “protein”.

ILP and SAT Approaches to the Prototein Problem. Previously [8, 21], ILP formulations were developed to solve the 2-D and 3-D Prototein Problems, and empirical results were reported in [8, 21]. The ILP formulations directly solve the optimization version of the problem, while the SAT formulation can only implement a *decision* version. The maximum number of contacts is then obtained by solving a series of such SAT decision problems, changing the target each time.

The Logic for the SAT Formulation. Given a target number of contacts, the SAT formulation must ensure that binary sequence S is legally embedded into grid G ; have logic to represent what is and isn't a potential contact; and have logic to count the number of potential contacts in the embedding. The resulting CNF formula will be satisfied if and only if there is a legal embedding where the number of potential contacts meets or exceeds the target number.

The points on the grid we use are numbered from 1 to w^2 for the 2-D version of the problem, and from 1 to w^3 in the 3-D version. So, each point on the grid is indexed by a single number. Clearly, w can be set to n , but choosing a smaller diameter results in much faster computation. We will discuss this later.

Ensuring Legal Embeddings. To represent the conditions for a legal embedding, we use variables $X_{i,j}$, where i is a character-position in S and j is a point in G . If variable $X_{i,j}$ is set true, then the character at position i in S is assigned to point j in G . There are four conditions that ensure a legal embedding (in both 2-D and 3-D) of a binary sequence S :

1. Every character $i \in S$ must be assigned to some point $j \in G$. The CNF formulas for this condition are shown in Eq. 1.

$$\bigvee_{1 \leq j \leq |G|} X_{i,j} \quad (1)$$

2. No character-position $i \in S$ can be assigned to more than one point $j \in G$. The CNF formulas for this condition are shown in Eq. 2.

$$\bigwedge_{\substack{1 \leq j, j' \leq |G| \\ j \neq j'}} (\overline{X_{i,j}} \vee \overline{X_{i,j'}}) \quad (2)$$

3. No point $j \in G$ can have more than one character-position $i \in S$ assigned to it. The CNF formulas for this condition are similar to those for condition 2, varying over i and i' rather than j and j' .
4. Every adjacent pair of character-positions $i, i+1 \in S$ must be placed on adjacent points in G (diagonals are not adjacent). So, for each position $i \in S$ and point $j \in G$, we need the CNF clause shown in Eq. 3.

$$\overline{X_{i,j}} \bigvee_{1 \leq j' \leq |G| \text{ } j \text{ and } j' \text{ are neighbors}} X_{i+1,j'} \quad (3)$$

Grid Size. To limit the number of variables, we set the grid diameter w to $1 + \lfloor \frac{n}{4} \rfloor$ for all values of $n \geq 12$, and to n otherwise. Likewise, in the 3-D version, w is set to $2 + \lfloor \frac{n}{8} \rfloor$ for all values of $n \geq 20$ and to $2 + \lfloor \frac{n}{4} \rfloor$ otherwise.

Identifying Potential Contacts. There are two conditions for a potential contact:

1. A potential contact involves point $j \in G$ if and only if a character 1 is assigned to point j . The variable T_j is set true if and only if this condition holds. The CNF formulas for this are omitted for lack of space.
2. A potential contact exists between points j and m on the grid if and only if a 1 is assigned to both points j and to a neighboring point m in G . The variable $C_{j,m}$ will be set to true if both T_j and T_m are set true. The CNF formulas for this are shown in Eq. 4.

$$\begin{aligned} & (\overline{C_{j,m}} \vee T_j) \wedge (\overline{C_{j,m}} \vee T_m) \wedge (C_{j,m} \vee \overline{T_j} \vee \overline{T_m}), \\ & \forall \text{ neighboring positions } j, m, 1 \leq j, m \leq |G| \end{aligned} \quad (4)$$

Counting Potential Contacts. It is not obvious how to use CNF formulas to *count* the number of potential contacts in an embedding, i.e., the number of C variables set true. To implement counting we use a non-trivial method detailed in [11] and originating in [2], which, given a goal number of potential contacts, m , counts the number of C variables that are *not* set true and tests whether it is less than or equal to r , where $r = 2|G| - m$ for the 2-D version and $r = 3|G| - m$ for the 3-D version. The CNF formulas for this are shown in Eq. 5.

$$\begin{aligned} & (\overline{b_i}^{2k} \vee \overline{b_j}^{2k+1} \vee b_{i+j}^k), \\ & 0 \leq i \leq t_{2k}, 0 \leq j \leq t_{2k+1}, 1 \leq i+j \leq t_k + 1, 1 < k < |G| \\ & (\overline{b_i}^2 \vee \overline{b_j}^3), 0 \leq i \leq t_2, 0 \leq j \leq t_3, i+j = r+1 \end{aligned} \quad (5)$$

The b variables exist for all internal nodes in a binary tree with $c = \lceil \log_2(r) \rceil$ levels. Each internal node is assigned $t_k = \min(r, 2^{c-d})$ b variables, where d is the depth of node k . The leaves are set to $\overline{C}$, and b_i^k being set true means there are at least i C variables set false in the leaves that descend from k . Using this method requires adding $O(|G|)$ b variables to the SAT implementation, which are not present in the ILP version, however—consistent with the ILP implementation—the number of variables required for the whole CNF formula remains $O(n|G|)$.

Testing. Both real and random sequences were tested for both 2-D and 3-D. The random sequences were generated with two different distributions: one where the average proportion of 1s in the sequence was $\frac{2}{3}$ (this matches the actual case of globular proteins); and one where the average proportion was $\frac{1}{3}$. The random sequences ranged in length from six to twenty-four characters long for 2-D and from six to sixteen characters long for 3-D. The same random sequences were used for both 2-D and 3-D and were only tested with Gurobi and pLingeling. Ten random sequences of each length from both distributions were generated for testing, so each length was tested with a total of twenty random sequences. The

real sequences used are a subset of those described and used in [12] and ranged in length from six to twenty-nine for 2-D and from six to twenty-five for 3-D; all were tested using Gurobi, pLingeling, and Glucose-Syrup. All the sequences were tested five times in order to compare the variance in run times on the same sequence. The tests were done via a pipeline that generated the files for both the CNF and ILP files; passed the files to the SAT and ILP solvers; and recorded the number of contacts found and the time taken by each solver. All the tests were done using a 2018 Mac Mini with a 3.2 GHz i7 processor. The pipeline programs for both the 2-D and 3-D versions, along with the input sequences used can be found on GitHub.²

Unlike Gurobi, the SAT-solvers can only report whether a target number of potential contacts is possible or not. Because of this, it was necessary to include a wrapper binary-search function which performs a modified binary search over the targets. The search starts with target 1 and doubles the target until a result of *unsatisfiable* is returned, then executes a normal binary search between that value and the last satisfiable value. The times for each run of the SAT-solver are recorded and aggregated in the end report, but the *generation* of the CNF formulas was not timed, as it is minimal, and not the subject of this study.

Results. Overall, Gurobi performed better than both pLingeling and Glucose-Syrup, both achieving faster times to reach an optimal solution and having smaller files. Between the two SAT-solvers, pLingeling performed better than Glucose-Syrup. In the tests run on real sequences, Glucose-Syrup generally performed better on short sequences than pLingeling, but was outperformed by pLingeling for longer sequences in both the 2-D and 3-D versions. One counterpoint of importance is that there were some sequences where Gurobi took much longer than did either SAT-solver,³ and there were fewer cases where pLingeling took a similar amount more time than Gurobi.⁴ Thus, pLingeling exhibited somewhat greater robustness than did Gurobi. Although both the SAT and ILP formulations have $O(n|G|)$ variables, it takes many more clauses and occurrences of un-negated and negated variables to describe the same problem using CNF than using ILP inequalities.

Comparison of SAT and ILP Run Times. The run times follow a trend that is similar to the trend for file sizes, with pLingeling and Glucose-Syrup generally taking longer than Gurobi for both 2-D and 3-D variants (see Fig. 1) and having a much higher variance per run than Gurobi. There was a small subset of problem instances that both SAT-solvers were able to solve much faster than Gurobi, but this was not the norm. For the 2-D version (with sequences between length 6 and length 29), pLingeling was able to reach a solution to most in less than 30 min, with none taking longer than three hours. Glucose-Syrup took at least twice as

² `prototein-problem` codebase: <https://github.com/hannah-aught/prototein-problem>.

³ Notably, the `2drpD2`, `1tf3A1`, and `1a1iA1` sequences in the 2-D version.

⁴ The `1be3k0` and `1byya0` sequences in the 3-D version, which, respectively, took pLingeling 3 h and half an hour to solve and Gurobi 5 min and 28 s.

long as pLingeling on most of the real sequences longer than 24, but performed slightly better on shorter sequences. Gurobi had times under ten minutes for most sequences between length 6 and 27 in 2-D; however, at lengths 28 and 29 the times became much less consistent, ranging from under 10 min to more than 48 h. Due to space limitations, we only show three graphs, two for real data in 2-D and 3-D, and one for random data in 2-D.

For the 3-D version (with lengths ranging from 6 to 15), all solvers were generally able to reach a solution within an hour, with Gurobi usually being the fastest. However, unlike in the 2-D version, Gurobi's times stayed more consistent at each length than both SAT-solvers'. At lengths 16 and above, all solvers became less consistent, with times between 20 s and 16 h for pLingeling, between 2 h and more than 3 days for Glucose-Syrup,⁵ and between 3 s and 1.6 h for Gurobi.

Conclusion from our Exploration of the HP Problem. Gurobi generally performs better in terms of time, and the file sizes are always smaller. Although both SAT-solvers usually take longer than Gurobi on both 2-D and 3-D versions of the problem, there are (rarer) cases where Gurobi takes much longer than either on a particular sequence. Hence, if Gurobi seems to be taking much longer than the typical time for that length, it is worth trying pLingeling or Glucose-Syrup.⁶

3 Transforming Gene Order by Reversals

There are important biological phenomena that occur at a scale larger than individual nucleotides, e.g. at *chromosomal* or *genomic* scales. One such informative biological phenomenon is *long chromosomal reversals (inversions)*, where the DNA in a long interval on a chromosome *reverses* direction. For example, if we represent each gene by a distinct integer, the interval with ten genes: 1 10 4 5 2 6 3 9 8 7 becomes: 1 10 6 2 5 4 3 9 8 7 when the interval containing 4 5 2 6 is reversed. Since genes can be separated by long distances on a chromosome, what seems like a small reversal of just four integers actually represents a very long chromosomal reversal. It is believed that long chromosomal reversals are much rarer, with longer periods of time between reversals, than are mutations of single nucleotides, so long chromosomal reversals allow us to look *farther back* into evolutionary history. So, what we are interested in is the *order* of the genes on the chromosome, and how that order changes over time. Then, following the principle of *parsimony*, the computational problem of interest is:

The Sorting-by-Reversals Problem. Given two permutations, P_1 and P_2 , of the integers 1 to n , find the *minimum* number of interval reversals that transforms P_1 to P_2 .

⁵ At which point it had still not reached a result and the process was terminated.

⁶ Although pLingeling was seen to perform better than Glucose-Syrup for most sequences.

(a) Randomly generated sequences with $\frac{2}{3}$ ones in the 2-D version

(b) Real sequences in the 2-D version

(c) Real sequences in the 3-D version

Fig. 1. Average SAT and ILP runtimes vs. sequence length for 2-D and 3-D embeddings. Note that the drop in times at lengths 12 and 20 for the 2-D and 3-D versions (respectively) is due to the choice of a smaller grid diameter

In the *signed* version of the problem, each integer in P_1 has a positive sign, and each integer in P_2 has either a positive or negative sign. Whenever an interval is reversed, the sign of each integer in the interval changes to the opposite sign. Then, the goal, as before, is to transform P_1 to the correctly signed P_2 using the minimum number of reversals.

A Hard Problem. There are no known algorithms to solve the sorting-by-reversals problem that are efficient in the worst-case sense,⁷ and it is not obvious how to solve sorting-by-reversal problems in practice. It is even less obvious how one could efficiently *prove* that an optimal solution had been found, even if you had one. So, ILP formulations for this problem were previously created and examined in [10, 14, 23], and an exposition of the ILP approach in [14] appears in [8].

ILP v. SAT. Although the ILP approach successfully solves many datasets, we previously found that the ILP approach becomes impractical (on a 4-core MacBook Pro) on permutations longer than around twelve. Therefore, we investigated how well the SAT approach handles the problem of sorting-by-reversals, both the unsigned and signed versions.

Two classic datasets were first examined. In the **Turnips to Cabbage** dataset [22], with ten genes, the turnip genes were labeled in the natural order 1 through 10, so that the order of the analogous genes in Cabbage is 1 10 4 5 2 6 3 9 8 7.

Gurobi 8.1 solved the problem in fifteen seconds, and the SAT formulation was generated and solved by our program, **do-unsigned**, in five seconds. **do-unsigned.py** implements the SAT approach for a given instance of the reversal problem by creating a series of CNF formulas, each specifying a maximum allowed number of reversals. Each CNF formula is found to be satisfiable or unsatisfiable by the SAT-solver *pLingeling*.

Note that the ILP time only includes the solution time by the ILP solver, excluding the time (usually quite small) to create the ILP formulation from the description of a problem instance. In contrast, the time for the SAT formulation also includes the time to generate (by **do-unsigned.py**) the CNF formulas used during the search for the minimum number of reversals.

In the **Field Mustard to Black Mustard** dataset, of length twelve, the ILP formulation was solved by Gurobi 8.1 in about 102 min, but earlier in only 33 min by Gurobi 8.0 (go figure!). The SAT approach solved the reversal problem in about 3 min, 25 s.

These tests indicate overall that the SAT approach is superior to the ILP approach. Results from the examination of random permutations of length up to eight (shown in Table 1) is consistent with these tests.

We also compared running times for the ILP and SAT approaches to the *signed* variant. There, the overall running times were longer for both approaches, but more interestingly, there were substantial differences in the times for the two approaches, shown in Table 2, which show dramatic superiority of the SAT approach.

⁷ Although there is an efficient algorithm for the *signed* version [9].

Table 1. Times (rounded to the nearest tenth of a second) taken by Gurobi and pLingeling with ten randomly generated unsigned sequences of length eight

Sequence number	1	2	3	4	5	6	7	8	9	10
pLingeling	1.3	3.1	2.5	1.3	1.5	1.7	2.8	1.6	12.9	1.3
Gurobi	5.0	13.5	11.2	9.0	6.2	10.5	9.1	3.0	74.0	8.7

Table 2. Times (rounded to the nearest second) taken by Gurobi and pLingeling with ten randomly generated signed sequences of length eight

Sequence number	1	2	3	4	5	6	7	8	9	10
pLingeling	57	463	56	11	50	430	54	58	19	78
Gurobi	30381	36387	7095	7855	3294	62050	6282	22293	362	13618

Other Observations. Although we saw that the SAT approach was faster than the ILP approach—and sometimes dramatically so—on permutations of length up to twelve, we found that its range of practicality is not much greater than that of the ILP approach. Another interesting finding is that for a fixed permutation length, the speed of the ILP approach falls as the number of cycles in the permutation increases, while the speed of the SAT approach is essentially constant over those permutations. For example, for length seven permutations, the SAT approach takes about one-half second no matter what the permutation is, but the time for the ILP approach falls from three seconds for permutations with one cycle, to about one-half second for permutations with four cycles.

Software. All the software for the SAT approach to the reversal problem is available on GitHub.⁸

4 The History Bound in Phylogenetic Networks

Genealogical and phylogenetic **networks** are graph-theoretic models of evolution that go beyond phylogenetic **trees**, incorporating *non-treelike* biological events such as *meiotic recombination* that occur in *populations* of individuals in a *single* species. The central algorithmic problems are to reconstruct *plausible* histories, with mutations, treelike events, and non-treelike events that generate a given set of observed genomic *sequences*, and to determine the *minimum* number of such biological events needed to derive the sequences.

The problem of finding a genealogical network that reconstructs the observed sequences, M , using the fewest recombinations (denoted $R_{\min}(M)$), and at most one mutation per site, has been heavily studied in both population genetics and

⁸ **transform-order-by-reversals** codebase: <https://github.com/zi674421351/transform-order-by-reversals>.

computational biology [7]. However, the best current algorithm takes super-exponential time in worst case. Hence, there has been great interest in finding effective algorithms to compute good (high) *lower bounds* on $R_{\min}(M)$. Empirically, the *History Bound* is usually the highest lower bound of all the lower bounds studied (there are more than ten bounds that have been intensely studied). However, its computation, by dynamic programming [1], examines all 2^n subsets of n input sequences, and hence becomes impractical as n increases.⁹ In its most intense use, in a program called *RecMin* [20], n is very large, but the number of sites, m , is generally under ten.

The Defining Algorithm for the History Bound. The History Bound was first defined *procedurally*,¹⁰ based on an algorithm that computes a *Candidate History Bound (CHB)*, given an n -by- m binary matrix M . In short, the CHB algorithm reduces M to the empty matrix by iterating three rules for removing rows or columns. In one rule (called the D_t rule), the choice of row removal is *arbitrary*, so different executions of the algorithm, with the same input, can remove a different set of rows. The History Bound is defined as the minimum number of times rule D_t is used, over *all* possible executions of Algorithm CHB. For a proof that the History Bound is a true lower bound on $R_{\min}(M)$, see [7].

The History Bound, ILP and SAT. Previously, we developed three different ILP formulations to compute the History Bound [18]. None of these three ILP formulations was effective in empirical testing on data of current interest. Therefore, we developed a SAT approach for the History Bound, as follows.

For any given target, t , our program creates a CNF formula that is satisfiable if and only if there is an execution of Algorithm CHB on the input M , which applies rule D_t at most t times. The CNF formula encodes the three row and column removal rules of Algorithm CHB, with additional CNF clauses that “count” the number of times rule D_t is used; and has another clause to forbid using it more than t times. Then pLingeling determines if that CNF formula is satisfiable or not. The History Bound is found by searching over the range of possible target values to find the smallest target where the associated CNF is satisfiable.

Results. When the target t is larger than the History Bound,¹¹ so that the associated CNF formula is satisfiable, pLingeling very efficiently finds a satisfying assignment. However, pLingeling has difficulty terminating when the CNF formula is *not* satisfiable. Space limits us to a few illustrative examples. For a matrix of dimension 60-by-8, the SAT approach determined that the History Bound is less than 11 in 45 s; less than 10 in 64 s; less than 9 in 31 s; less than 8 in 52 s; and less than 7 in 87 s. Then the test of whether the History Bound is less than

⁹ The original algorithm, in [20] required $\theta(mn!)$ time, where each of the n input sequences has m sites.

¹⁰ A non-procedural definition of the History Bound was later established in [19].

¹¹ In tests where we know the true History Bound.

6 ran for 7,200 s without termination, at which point we killed the computation. In contrast, the ILP approach determined that the History Bound is less than 11 after 900 s, and less than 7 after 3,351 s; and that computation continued without termination until 18,000 s, when it was killed. Significantly, the ILP could only establish that the History Bound for this case is greater or equal to zero! So, the SAT approach found an execution of Algorithm CHB using the D_t rule only six times, much faster than did the ILP approach. And it provided much stronger evidence than did the ILP computation, for concluding that the History Bound is six for that matrix.

In a test with a matrix of dimension 30-by-60, the SAT approach determined that the History Bound is less than 30, 29, 28, and 27, in 21, 38, 26, and 63 s respectively. Then in trying to determine if the History Bound is less than 26, we terminated pLingeling after 480 s. In contrast, we allowed the ILP computation to run for 13,000 s at which point it has determined only that the History Bound was less than 30 and greater than zero! So, the SAT approach gave strong evidence that the History Bound is 26 for this matrix, while the ILP approach gave much less information, using much more time.

All of the software to create the ILP and CNF formulations for the History Bound is available on GitHub.¹²

5 Haplotyping by Pure Parsimony

Very briefly, input consists of n sequences of length m , over a *ternary* alphabet $\{0, 1, 2\}$. Output consists of a set of binary sequences, H , of length m , where each input sequence, s , is associated with some pair of binary sequences, (h_i, h_j) in H , which must satisfy the following conditions: At any position k in s , if the value of s is 0 or 1, then for both h_i and h_j , the value at position k must be identical to the value in s . However, if the value of s at position k is 2, then exactly one of h_i or h_j must have value 0, and the other must have value 1, at position k . When s has q positions with value 2, there are 2^{q-1} possible binary sequences that satisfy those conditions.

The **Pure-Parsimony Haplotyping Problem** is to find a *smallest* such set H , given the input. This minimization problem was first examined in [6], where an exponential-size ILP formulation was developed.¹³ That ILP formulation solves very quickly for instances where the formulation fits into main memory. However, rapid growth of the formulation size limits the size of the instances that can be handled. Later (in 2004 and 2006), polynomial-sized ILP formulations were developed [4, 13], but ILP solvers at that time were not able to solve problem instances of interest in a practical amount of time. Subsequently (in 2006), a SAT-formulation, based on the general ideas in [4, 13] and additional improvements, was developed in [15, 16], and was shown to be highly

¹² History Bound/Haplotyping codebase: <https://github.com/gusfield/ILP-SAT-comparisons>.

¹³ See also [8] for a full discussion of the problem and its biological import.

effective. Those results used ILP and SAT-solvers that are much slower than current solvers. So, we re-implemented the ILP and SAT approaches¹⁴ to see how the more modern SAT-solver, pLingeling, and the fastest current ILP-solver, Gurobi 9.0, handle this problem. The code for our implementation is available on GitHub.¹⁵

Our new results confirm and extend the results from the older studies. Space limits the discussion to just one problem instance, of dimension 200-by-200, which is much larger than the instances examined in the earlier literature. In the SAT approach, using a first target of 87, pLingeling found a satisfying solution to the CNF formula in 1,525 s. With targets of 85 and 84, it found satisfying solutions in 5,700 and 8,521 s, respectively. Then, using a target of 83, pLingeling determined that the associated CNF formula was unsatisfiable, in 20,627 s. This established that the optimal value is 84. In contrast, using the ILP formulation for this problem instance, Gurobi initially found a feasible solution of size 134, and a lower bound of 31 (in 1,300 s), but then made no further progress in two more days of computation (when it was terminated).

6 Conclusions

While the ILP approach is becoming more widely used in computational biology, only a few studies have explored the SAT-solving approach. Our work shows that SAT-solving can be (but is not always) much more effective than ILP-solving for instances of hard problems in computational biology.¹⁶ This gives us another powerful tool, which should be more widely used and incorporated into biological computation.

References

1. Bafna, V., Bansal, V.: Inference about recombination from haplotype data: lower bounds and recombination hotspots. *J. Comput. Biol.* **13**, 501–521 (2006)
2. Bailleux, O., Boufkhad, Y.: Efficient CNF encoding of Boolean cardinality constraints. In: Rossi, F. (ed.) *CP 2003*. LNCS, vol. 2833, pp. 108–122. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45193-8_8
3. Biere, A.: <http://fmv.jku.at/lingeling>
4. Brown, D., Harrower, I.: Integer programming approaches to haplotype inference by pure parsimony. *IEEE/ACM Trans. Comput. Biol. Bioinform.* **3**(2), 141–154 (2006)

¹⁴ Although our SAT implementation is more basic and naive than the one developed in [15, 16].

¹⁵ **History Bound/Haplotyping** codebase: <https://github.com/gusfield/ILP-SAT-comparisons>.

¹⁶ We reiterate, however, that the specific problems examined in this paper were chosen because they were known to be hard for ILP-solvers to handle. Therefore, these results do not contradict the more general conclusion (such as detailed in [8]), that the ILP approach has broad, successful (even transformative) application in computational biology.

5. Dill, K.A., et al.: Principles of protein folding - a perspective from simple exact models. *Protein Sci.* **4**, 561–602 (1995)
6. Gusfield, D.: Haplotype inference by pure parsimony. In: Baeza-Yates, R., Chávez, E., Crochemore, M. (eds.) *CPM 2003. LNCS*, vol. 2676, pp. 144–155. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-44888-8_11
7. Gusfield, D.: *ReCombinatorics: The Algorithmics of Ancestral Recombination Graphs and Explicit Phylogenetic Networks*. MIT Press, Cambridge (2014)
8. Gusfield, D.: *Integer Linear Programming in Computational and Systems Biology: An Entry-Level Text*. Cambridge University Press, Cambridge (2019)
9. Hannenhalli, S., Pevzner, P.: Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. In: *Proceedings of the 27th ACM Symposium on the Theory of Computing*, pp. 178–189 (1995)
10. Hartmann, T., Wieseke, N., Sharan, R., Middendorf, M., Bernt, M.: Genome rearrangement with ILP. *IEEE/ACM Trans. Comput. Biol. Bioinform.* **15**(5), 1585–1593 (2018). <https://doi.org/10.1109/TCBB.2017.2708121>
11. Knuth, D.E.: *The Art of Computer Programming. Fascicle 6: Satisfiability*, vol. 4. Addison-Wesley, Boston (2015)
12. Kolodny, R., Koehl, P., Levitt, M.: Comprehensive evaluation of protein structure alignment methods: scoring by geometric measures. *J. Mol. Biol.* **346**(4), 1173–1188 (2005). <https://doi.org/10.1016/j.jmb.2004.12.032>
13. Lancia, G., Pinotti, C., Rizzi, R.: Haplotyping populations by pure parsimony: complexity, exact and approximation algorithms. *INFORMS J. Comput. Spec. Issue Comput. Biol.* **16**, 348–359 (2004)
14. Lancia, G., Rinaldi, F., Serafini, P.: A unified integer programming model for genome rearrangement problems. In: Ortuño, F., Rojas, I. (eds.) *IWBBIO 2015. LNCS*, vol. 9043, pp. 491–502. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16483-0_48
15. Lynce, I., Marques-Silva, J.: Efficient haplotype inference with Boolean satisfiability. In: *Proceedings of the Twenty-First AAAI Conference on Artificial Intelligence*, pp. 104–109 (2006)
16. Lynce, I., Marques-Silva, J.: SAT in bioinformatics: making the case with haplotype inference. In: Biere, A., Gomes, C.P. (eds.) *SAT 2006. LNCS*, vol. 4121, pp. 136–141. Springer, Heidelberg (2006). https://doi.org/10.1007/11814948_16
17. Malikic, S., et al.: PhISCS: a combinatorial approach for subperfect tumor phylogeny reconstruction via integrative use of single-cell and bulk sequencing data. *Genome Res.* **29**, 1860–1877 (2019)
18. Matsieva, J.: *Optimization techniques for phylogenetics*. Ph.D. thesis, Department of Computer Science, University of California, Davis (2019)
19. Matsieva, J., Kelk, S., Scornavacca, C., Whidden, C., Gusfield, D.: A resolution of the static formulation question for the problem of computing the history bound. *IEEE/ACM Trans. Comput. Biol. Bioinf.* **14**(2), 404–417 (2017). <https://doi.org/10.1109/TCBB.2016.2527645>
20. Myers, S., Griffiths, R.C.: Bounds on the minimum number of recombination events in a sample history. *Genetics* **163**, 375–394 (2003)
21. Nunes, L., Galvao, L., Lopes, H., Moscato, P., Berretta, R.: An integer programming model for protein structure prediction using the 3D-HP side chain model. *Discret. Appl. Math.* **198**, 206–214 (2016)
22. Palmer, J., Herbon, L.: Plant mitochondrial DNA evolves rapidly in structure, but slowly in sequence. *J. Mol. Evol.* **27**, 87–97 (1988)
23. Shao, M., Moret, B.M.E.: Comparing genomes with rearrangements and segmental duplications. *Bioinformatics* **31**(12), i329–i338 (2015)