A Cost-Aggregating Integer Linear Program for Motif Finding

Carl Kingsford[1], Elena Zaslavsky[2], and Mona Singh[2]

[1] Center for Bioinformatics & Computational Biology, University of Maryland, College Park, MD
carlk@umiacs.umd.edu
[2] Department of Computer Science and Lewis-Sigler Institute for Integrative Genomics, Princeton University, Princeton, NJ
{elenaz,msingh}@cs.princeton.edu

Abstract. In the *motif finding problem* one seeks a set of mutually similar subsequences within a collection of biological sequences. This is an important and widely-studied problem, as such shared motifs in DNA often correspond to regulatory elements. We study a combinatorial framework where the goal is to find subsequences of a given length such that the sum of their pairwise distances is minimized. We describe a novel integer linear program for the problem, which uses the fact that distances between subsequences come from a limited set of possibilities allowing for aggregate consideration of sequence position pairs with the same distances. We show how to tighten its linear programming relaxation by adding an exponential set of constraints and give an efficient separation algorithm that can find violated constraints, thereby showing that the tightened linear program can still be solved in polynomial time. We apply our approach to find optimal solutions for the motif finding problem and show that it is effective in practice in uncovering known transcription factor binding sites.

1 Introduction

A central challenge in post-genomic biology is to reconstruct the regulatory network of an organism. A key step in this process is the discovery of regulatory elements. A common approach finds novel sites by searching for a set of mutually similar subsequences within DNA sequences. These subsequences, when aligned, form *motifs*, and are putative binding sites for a shared transcription factor. The effectiveness of identifying regulatory elements in this manner has been demonstrated when considering sets of sequences identified via shared co-expression, orthology and genome-wide location analysis (e.g., [19, 8, 11]).

Numerous problem formalizations and computational approaches have been developed for motif finding (see [21], and references therein). Probabilistic approaches typically try to maximize the chance of observing the chosen motif instances (e.g., [10, 3, 7]). Combinatorial methods either enumerate all allowed motifs or attempt to optimize some measure based on sequence similarity (e.g., [13, 12]). Here, we take a combinatorial approach and model the motif finding problem as that of finding the gapless local multiple sequence alignment of fixed length that minimizes a sum-of-pairs (SP) distance measure. Such a formulation provides a reasonable scheme for assessing motif conservation [15, 18]. The problem is equivalent to that of finding a minimum weight clique of size p in a p-partite graph (e.g., [16]). For general notions of distance, this problem is NP-hard to approximate within any reasonable factor [4]. The problem and its variants remain NP-hard in the context of biological sequences [1, 22], though in the motif finding setting, where the distances obey the triangle inequality, constant-factor approximation algorithms exist [2]. Nevertheless, the ability to find the optimal solution in practice is preferable.

We introduce and extensively explore a mathematical programming approach to motif finding. We propose a novel integer linear programming (ILP) formulation of the motif finding problem that uses the discrete nature of the distance metric imposed on pairs of subsequences, thereby allowing aggregation of edges of the same weight. Considering its linear programming (LP) relaxation, we show that while it is weaker than an alternative LP formulation for motif finding [23], an exponentially-sized class of constraints can be added to make the two formulations equivalent. We then show that it is not necessary to explicitly add all these constraints by giving a separation algorithm, based on identifying minimum cuts in a graph constructed to model the ILP, that identifies violated constraints. The ellipsoid method [6] can then be used to find the solution to the tightened LP in time polynomial in the number of variables of the mathematical programming problem.

We test the effectiveness of our approach in identifying DNA binding sites of *E. coli* transcription factors. We demonstrate that our new ILP framework is able to find optimal solutions often an order of magnitude faster than the previously known mathematical programming formulation, and that its performance in identifying motifs is competitive with a widely-used probabilistic Gibbs-sampling approach [20]. Finally, we note that in practice the LP relaxations often have integral optimal solutions, making solving the LP sufficient in many cases for solving the original ILP.

2 Formal Problem Specification

We are given p sequences, which are assumed without loss of generality to each have length N', and a motif length ℓ. In our formulation, the goal is to find a subsequence s_i of length N' in each sequence i so as to minimize the sum of the pairwise distances between the subsequences. Here, the distance between two substrings s_i and s_j is computed as the Hamming distance $\mathbf{Hamming}(s_i, s_j)$ between them and thus our goal is to choose the substrings such that $\sum_{i<j} \mathbf{Hamming}(s_i, s_j)$ is minimized.

The problem can be reformulated in graph-theoretic terms. For p input sequences, we define a complete, weighted p-partite graph $G = (V, E)$, with a part for each sequence. In part i, there is a node for every substring of length ℓ in sequence i. Let V_i be the set of nodes in the part corresponding to sequence i. Thus there are $N := N' - \ell + 1$ nodes in each V_i, and the vertex set $V = V_1 \cup \cdots \cup V_p$ has size Np.

For every pair of nodes u and v in different parts there is an edge $(u, v) \in E$. Letting $\mathbf{seq}(u)$ denote the subsequence corresponding to node u, the weight w_{uv} on edge (u, v) equals $\mathbf{Hamming}(\mathbf{seq}(u), \mathbf{seq}(v))$. The goal is to choose a node from each part so as to minimize the weight of the induced subgraph.

3 Integer Programming Formulations

3.1 Original integer linear programming formulation

We first give the integer linear programming formulation presented in [23] for solving the motif finding problem. In this ILP formulation, there is a variable X_u for each node u in the graph described above. The variable X_u is set to 1 if node u is chosen, and 0 otherwise. Additionally, there is one variable X_{uv} for each edge in the graph (X_{uv} is the same as X_{vu}). These edge variables are set to 1 if both end points of the edge are chosen. In the integer program, all variables are constrained to take values from $\{0, 1\}$. The following ILP is easily seen to model the above graph problem:

$$\begin{array}{ll}
\text{Minimize} & \sum_{\{u,v\}\in E} w_{uv} \cdot X_{uv} \\
\text{subject to} & \\
\quad \sum_{u\in V_i} X_u = 1 & \text{for each } i = 1, \ldots, p \\
\quad \sum_{u\in V_i} X_{uv} = X_v & \text{for each } i = 1, \ldots, p \text{ and for each fixed } i, v \in V \setminus V_i \\
\quad X_u, X_{uv} \in \{0, 1\} &
\end{array} \tag{IP1}$$

The first set of constraints ensures that one node is chosen from each part, and inclusion of the second set of constraints adds the requirement that an edge is chosen if its end points are. This ILP is the same as the ILP formulation for protein side-chain positioning presented in [9].

3.2 More compact integer linear program

We now introduce an alternative ILP that better exploits the structure of the combinatorial problem. In particular, we use the fact that there are typically only a small number of possible pairwise distances. For example, in the case of Hamming distances, edge weights can only take on $\ell + 1$ different values. We can take advantage of the small number of possible weights and the fact that the edge variables of IP1 are only used to ensure that if two nodes u and v are chosen in the optimal solution then w_{uv} is added to the cost of the clique. In our new ILP formulation, we no longer have edge variables X_{uv}. Instead, in addition to the node variables X_u, we have a variable Y_{ujc} for each node u, each part j such that $u \notin V_j$, and each edge weight c. These Y variables model groupings of the edges by cost into *cost bins*, as shown in Fig. 1. The intuition is that Y_{ujc} is 1 if node u and some node $v \in V_j$ are chosen such that $w_{uv} = c$.

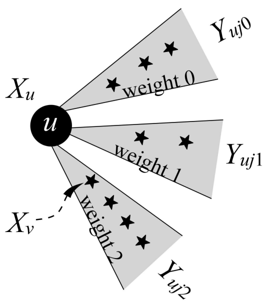

Fig. 1: Schematic of IP2. Adjacent to a node $u \in V_i$ there are at most $|D|$ cost bins for each part $j > i$, each associated with a variable Y_{ujc}. For each cost c there are the nodes $v \in V_j$ for which $w_{uv} = c$ (stars).

Formally, let D be the set of possible edge weights (costs) and let $W = \{(u, j, c) : c \in D, u \in V, j \in 1, \ldots p \text{ and } u \notin V_j\}$ be the set of triples over which the Y_{ujc} variables are indexed, and let $\mathbf{part}(u) = i$ if $u \in V_i$. Then the following ILP models the motif-finding graph problem:

$$
\begin{array}{lll}
\text{Minimize} & \sum_{(u,j,c)\in W:\mathbf{part}(u)<j} c \cdot Y_{ujc} & \\
\text{subject to} & & \\
\quad \sum_{u\in V_i} X_u = 1 & \text{for } i = 1, \ldots, p & \text{(IP2a)} \\
\quad \sum_{c\in D} Y_{ujc} = X_u & \text{for } j \in 1, \ldots, p \text{ and } u \in V \setminus V_j & \text{(IP2b)} \\
\quad \sum_{v\in V_j : w_{uv}=c} Y_{vic} \geq Y_{ujc} & \text{for } (u, j, c) \in W \text{ s.t. } u \in V_i \text{ and } i < j & \text{(IP2c)} \\
\quad X_u, Y_{ujc} \in \{0, 1\} & & \\
& & \text{(IP2)}
\end{array}
$$

As in IP1, the first set of constraints forces a single node to be chosen in each part. The second set of constraints makes certain that if a node u is chosen, then for each j, one of its "adjacent" cost bins must also be chosen (Fig. 1). The third set of constraints ensures that Y_{ujc} can be selected only if some node $v \in V_j$, such that $w_{uv} = c$, is also selected. We discard variables Y_{ujc} if there is no $v \in V_j$ such that $w_{uv} = c$. Fig. 1 gives a schematic drawing of these constraints.

Lemma 1. *IP2 correctly models the sum-of-pairs motif finding problem.*

Proof. For any choice of p-clique $\{u_1, \ldots, u_p\}$ of weight $\gamma = \sum_{i<j} w_{u_i u_j}$, a solution of cost γ to IP2 can be found by taking $X_{u_i} = 1$ for $i = 1 \ldots, p$, and taking $Y_{u_i jc} = 1$ for all $1 \leq j \leq p$ such that $w_{u_i u_j} = c$. This solution is feasible, and between any pair of graph parts i, j it contributes cost $w_{u_i u_j}$; therefore, the total cost is γ. On the other hand, consider any solution (X, Y) to IP2 of objective value γ. Consider the clique formed by the nodes u such that $X_u = 1$. Between every two parts $i < j$, the constraints (IP2a) and (IP2b) imply that exactly one Y_{ujc} and one $Y_{vic'}$ are set to 1 for some $u \in V_i$ and $v \in V_j$ and costs c, c'.

Constraint (IP2c) corresponding to (u, j, c) with Y_{ujc} on its right-hand side can only be satisfied if the sum on its left-hand side is 1, which implies $c = c' = w_{uv}$. Thus, a clique of weight γ exists in the motif-finding graph problem. $\square$

3.3 Advantages of IP2

In practice, IP2 has many fewer variables than IP1. Letting $d = |D|$, the number of kinds of weights, IP2 has $Np((p-1)d+1)$ variables in the case that a Y_{ujc} variable exists for every allowed choice of (u, j, c), while IP1 has $Np(N(p-1)/2+1)$ variables. If $d < N/2$, the second IP will have fewer variables. In general, d is expected to be much smaller than N: while N could reasonably be expected to grow large as longer and longer sequences are considered, d is constrained by the geometry of transcription factor binding and will remain small. Also, in practice, it is likely that many Y_{ujc} variables are removed because $\mathbf{seq}(u)$ does not have matches of every possible weight in each of the other sequences. On the other hand, IP2, will have $O(d)$ times more constraints than IP1, with the number of constraints being $p + Np(p-1)(d/2+1)$ for IP2, and $p + Np(p-1)$ for IP1.

In practice, smaller integer programs with weaker LP relaxations are often less useful for branch-and-bound approaches to IP solving. Thus, we seek the tightest, smallest IP possible. While the decrease in variables of IP2 tends to be more dramatic than the increase in the number of constraints, experiments must still be performed to gauge the efficacy of various formulations on practical problems. We present experiments below (see **Computational Results** and Fig. 4), which suggest IP2 can be more than an order of magnitude faster than IP1.

4 Linear programming relaxations

The typical approach to solving an ILP is to solve as a subproblem the linear program relaxation derived from the ILP by dropping the requirement that the variables be in $\{0, 1\}$, and instead requiring only that the variables lie in the continuous range $[0, 1]$. While finding a solution to the ILP is computationally difficult, its relaxed LP can be solved in polynomial-time. If the solution to the relaxed LP is integral, then we have found a solution to the original ILP. Alternatively, if the solution to the LP is fractional, then branch and bound or other techniques can be used to obtain optimal solutions to the ILP.

The LP relaxation of IP1, which we refer to as LP1, is stronger than the LP relaxation of IP2. Since we are minimizing the objective function, this means that there are some problem instances for which the optimum value of LP2 is smaller than that of LP1. Because stronger LP relaxations are often more useful subroutines for finding optimal integer solutions, we first present a natural (though exponential) class of constraints that, if added to the LP relaxation of IP2, makes the two formulations equivalent in the sense that they have the same optimum and that an optimal solution to one can be easily derived from the optimal solution of the other. We refer to this fully constrained relaxation of IP2

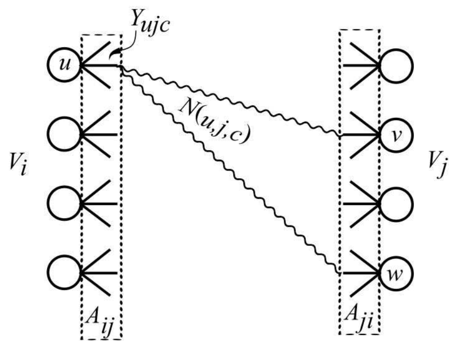

Fig. 2: Mapping for the compatibility graph $\mathcal{C}_{ij}$. The two columns of circles represent nodes in V_i and V_j. Solid lines adjacent to each circle represent the Y_{ujc} or Y_{vic} variables associated with the node. A_{ij} and A_{ji} (dotted boxes) are the sets of these variables associated with the pair of graph parts i and j. The function $\mathcal{N}(u,j,c)$ maps a variable Y_{ujc} to a set of compatible Y_{vic} variables (squiggly lines). $\mathcal{N}(u,j,c)$ is shown assuming that v and w are the only nodes in V_j that have cost c with u.

as LP2. Later we give a separation algorithm for finding violated constraints, and thereby show that LP2 can still be solved in polynomial-time.

4.1 Additional constraints

Focus on a pair of graph parts i and j. In IP1 the edge variables between nodes in V_i and V_j explicitly model the bipartite graph between those two parts. In IP2, however, the bipartite graph is only implicitly modeled by an understanding of which Y variables are compatible to be chosen together. We study this implicit representation by considering the bipartite *compatibility* graph $\mathcal{C}_{ij}$ between two parts i and j. Intuitively, we have a node in this compatibility graph for each Y_{ujc} and Y_{vic}, and there is an edge between the nodes corresponding to Y_{ujc} and Y_{vic} if $w_{uv} = c$. These two Y variables are compatible in that they can both be set to 1 in IP2. More formally, $\mathcal{C}_{ij} = (A_{ij}, A_{ji}, F)$, where $A_{ij} = \{(u,j,c) : u \in V_i, c \in D\}$ is the set of indices of Y variables adjacent to nodes in V_i, going to part j, and A_{ji} is defined analogously, going in the opposite direction. The edge set F is defined in terms of the neighbors of a triple (u,j,c). Let $\mathcal{N}(u,j,c) = \{(v,i,c) : u \in V_i, (v,i,c) \in A_{ji} \text{ and } w_{uv} = c\}$ be the *neighbors* of (u,j,c). They are the indices of the Y_{vic} variables adjacent to part j going to part i so that the edge $\{u,v\}$ has weight c. There is an edge in F going between (u,j,c) and each of its neighbors. Similarly to above, we call c the *cost* of triple (u,j,c). All this notation is summarized in Fig. 2.

In any feasible integral solution, if $Y_{ujc} = 1$, then some Y_{vic} for which $(v,i,c) \in \mathcal{N}(u,j,c)$ must also be 1. Extending this insight to subsets of the Y_{ujc} variables yields a class of constraints that will ensure that the resulting LP formulation is as tight as LP1. That is, choose any set of Y_{ujc} variables adjacent to part i. Their sum must be less than or equal to the sum of the Y variables for their neighbors. Formally, if $Q_{ij} \subseteq A_{ij}$, then let $\mathcal{N}(Q_{ij}) = \bigcup_{(u,j,c)\in Q_{ij}} \mathcal{N}(u,j,c)$ be the set of indices that are neighbors to any vertex in Q_{ij}. If $Q_{ij} \subseteq A_{ij}$ then $\mathcal{N}(Q_{ij}) \subseteq A_{ji}$. The following constraint is true in IP2 for any such Q_{ij}:

$$\sum_{(u,j,c)\in Q_{ij}} Y_{ujc} \le \sum_{(v,i,c)\in \mathcal{N}(Q_{ij})} Y_{vic}\,. \tag{1}$$

Notice that the set of constraints (IP2c) is of the form (1), taking Q_{ij} to be the singleton set $\{(u,j,c)\}$.

Theorem 1. *If for every pair of graph parts $i < j$, constraints of the form (1) are added to IP2 for each $Q \subseteq A_{ij}$ s.t. all triples in Q are of the same cost, the resulting LP relaxation LP2 has the same optimal solution as that of the relaxation LP1 of IP1.*

Proof. It is clear that the LP relaxation LP2 described in Theorem 1 is no stronger than LP1 as any solution to LP1 can be converted to a feasible solution of LP2 by making the node variable weights the same and putting the weight of edge variables X_{uv} onto Y_{ujc} and Y_{vic}, where $w_{uv} = c$. This solution to LP2 will satisfy all the constraints in the theorem, and be of the same objective value.

The rest of the proof will involve showing that for any feasible solution for LP2, there is a feasible solution for LP1 with the same objective value, thereby demonstrating that the optimal solution to LP2 is not weaker than the optimal solution to LP1. In particular, fix a solution (X, Y) to LP2 with objective value γ. We need to show that for any feasible distribution of weights on the Y variables a solution to LP1 can be found with objective value γ.

In order to reconstruct a solution $\hat{X}$ for LP1 of objective value γ, we will set $\hat{X}_u = X_u$, using the values of the node variables X_u in the optimal solution to LP2. We must assign values to $\hat{X}_{uv}$ to complete the solution. Recall the compatibility graph $\mathcal{C}_{ij}$. Because all edges in $\mathcal{C}_{ij}$ are between nodes of the same cost, $\mathcal{C}_{ij}$ is really $|D|$ disjoint bipartite graphs $\mathcal{C}^c_{ij}$, one for each cost c. Let $A^c_{ij} \cup A^c_{ji}$ be the node set for the subgraph $\mathcal{C}^c_{ij}$ for cost c. Each edge in a subgraph $\mathcal{C}^c_{ij}$ corresponds to one edge in the graph G underlying LP1. Conversely, each edge in G corresponds to exactly one edge in one of the $\mathcal{C}^c_{ij}$ graphs (if edge $\{u, v\}$ has cost c_1, it corresponds to an edge in $\mathcal{C}^{c_1}_{ij}$). We will thus proceed by assigning values to the edges in the various $\mathcal{C}^c_{ij}$, and this will yield values for the $\hat{X}_{uv}$.

If $y(A)$ is defined as $\sum_{(u,j,c)\in A} Y_{ujc}$, by the sets of constraints (IP2a) and (IP2b), $y(A_{ij}) = y(A_{ji}) = 1$. Since the constraints (1) are included with $Q = A^c_{ij}$ for each cost c, $y(A^c_{ij}) = y(A^c_{ji})$ for every cost c. Thus, for each subgraph $\mathcal{C}^c_{ij}$, the weight placed on the left half equals the weight placed on the right half. We will consider each induced subgraph $\mathcal{C}^c_{ij}$ separately.

We modify $\mathcal{C}^c_{ij}$ as follows to make it a capacitated flow network. Direct the edges of $\mathcal{C}^c_{ij}$ so that they go from A^c_{ij} to A^c_{ji}, and set the capacities of these edges to be infinite. Add source and sink nodes $\{r, s\}$ and edges directed from r to each

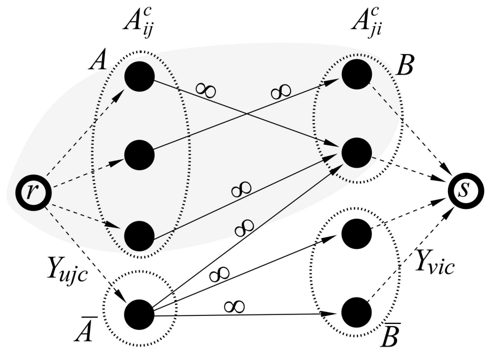

Fig. 3: Flow network $\mathcal{C}^c_{ij}$ between part i and j. Nodes r and s are a source and sink. Each solid node corresponds to a Y variable. The edges between A_{ji} and A_{ij} have infinite capacity, while those entering s or leaving r have capacity equal to the value of the Y variable to which they are adjacent. The shading gives an $r-s$ cut.

node in A^c_{ij} and edges directed from each node in A^c_{ji} to s. Every edge adjacent to r and s is also adjacent to some node representing a Y variable; put capacities on these edges equal to the value of the adjacent Y variable (see Fig. 3).

The desired solution to LP1 can be found if the weight of the nodes (Y variables) in each compatibility subgraph can be spread over the edges. That is, a solution to LP1 of weight γ can be found if, for each pair (i,j) and each c, there is a flow of weight $y(A^c_{ij})$ from r to s in the flow network. The assignment to $\hat{X}_{uv}$ will be the flow crossing the corresponding edge in the $\mathcal{C}^c_{ij}$ of appropriate cost. In Lemma 2 below, we show that the set of constraints described in the theorem ensure that the minimum cut in the flow network is equal to $y(A^c_{ij})$, and thus there is a flow of the required weight. The proof of this fact is similar to those of other flow feasibility problems found in [5]. Together with the lemma we have shown LP1 and LP2 to be equivalent. □

Lemma 2. *The minimum cut of the flow network described in the proof of Theorem 1 (and shown in Fig. 3) is $y(A^c_{ij})$.*

Proof. Recall that the capacities of the edges leaving r are Y_{ujc} and those entering s are Y_{vic}, and that the total capacity leaving r equals the total capacity entering s, and it is $y(A^c_{ij})$. We want to show that the minimum $r-s$ cut in this graph is $\geq y(A^c_{ij})$.

Consider an $r-s$ cut $\{r\} \cup A \cup B$ where $A \subseteq A^c_{ij}$ and $B \subseteq A^c_{ji}$ (shaded in Fig. 3). Define $\bar{A} = A^c_{ij} \setminus A$ and $\bar{B} = A^c_{ji} \setminus B$. If any edges go between A and $\bar{B}$ then the capacity is infinite, and we are done. Otherwise the value of the cut is the sum of the capacities of the edges leaving r and entering $\bar{A}$ plus the sum of the capacities of the edges entering s from B. We will now show that $y(\bar{A}) \geq y(\bar{B})$, which implies that the value of the cut is $\geq y(A^c_{ji}) = y(A^c_{ij})$.

Assume for the moment that all nodes in $\bar{A}$ have a neighbor in $\bar{B}$. Then $\mathcal{N}(\bar{B}) = \bar{A}$ because there are no edges between A and $\bar{B}$. By (1), $y(\bar{B}) \leq y(\mathcal{N}(\bar{B})) = y(\bar{A})$. On the other hand, if there is a node in $\bar{A}$ that does not have a neighbor in $\bar{B}$ then we can add that node to A to make A' (without increasing the cost of the cut), and the above argument shows that $y(A^c_{ij} \setminus A') \geq y(\bar{B})$, which implies $y(\bar{A}) \geq y(\bar{B})$ since $A^c_{ij} \setminus A' \subseteq \bar{A}$. $\square$

4.2 Separation algorithm

Despite the exponential number of constraints, it is possible to solve LP2 in time polynomial in the number of variables of LP2 (which is polynomial in the size of the input graph) using the ellipsoid algorithm [6], provided that there exists a separation algorithm. Such an algorithm, given any values for the X and Y variables of LP2, finds constraint of LP2 that is violated, if one exists, in polynomial time or reports that no constraints are violated. The next theorem gives such an algorithm, formalizing the intuition in the proof of Theorem 1, by which all constraints are satisfied in a compatibility graph only if a large enough maximum flow exists. Otherwise, the minimum cut identifies a violated constraint.

Theorem 2. *There is a polynomial-time algorithm that, given values for the X and Y variables of LP2, can find a violated constraint of LP2 if one exists or otherwise report that none exists.*

Proof. The first step in such an algorithm is to explictly check X and Y against each of the polynomial number of constraints IP2a, IP2b, IP2c as well as the non-negativity constraints. If X and Y violate one of these constraints, we return that constraint. Otherwise, either all the constraints of LP2 are satisfied or the violated constraint is of the form (1). We describe below a polynomial-time algorithm that will find a violated constraint of the form (1) if one exists.

Because each constraint in (1) involves variables of a single cost, if (1) is violated for some set Q, then Q is a subset of an A^c_{ij} for some i, j, c, and so we can consider each subgraph $\mathcal{C}^c_{ij}$ independently. The proof of Theorem 1 shows that there is a violated constraint of the form (1) between i, j involving variables of cost c if and only if the maximum flow in C^c_{ij} is less than $y(A^c_{ij})$. Thus, the minimum cut can be found for each triple i, j, c, and, if a triple i, j, c is found where the minimum cut is less than $y(A^c_{ij})$, one knows that a violated constraint exists between parts i and j with $Q \subset A^c_{ij}$.

The minimum cut can then be examined to determine the violated constraint explicitly. Let $\{r\} \cup A \cup B$ be the minimum $r - s$ cut in C^c_{ij}, with $A \subseteq A^c_{ij}$ and $B \subseteq A^c_{ji}$. Such a cut is shaded in Fig. 3. Let m be the capacity of this cut, and assume, because we are considering a triple i, j, c that was identified as having a violated constraint, that $y(A^c_{ij}) > m$. For ease of notation let $\bar{A} = A^c_{ij} \setminus A$ and $\bar{B} = A^c_{ji} \setminus B$. Because $m < \infty$ there are no edges going from A to $\bar{B}$, and hence two things hold: (1) $m = y(B) + y(\bar{A})$ and (2) $\mathcal{N}(A) \subseteq B$, and therefore

$y(\mathcal{N}(A)) \leq y(B)$. Chaining these facts together, we have

$$y(A) = y(A^c_{ij}) - y(\bar{A}) > m - y(\bar{A}) = y(B) \geq y(\mathcal{N}(A))\,,$$

Thus, the set A is a set for which the constraint of the form (1) is violated. $\square$

In practice the ellipsoid algorithm is often slower than the well optimized simplex algorithm. We can use the faster simplex algorithm because not all of these constraints are necessary for real problems. Some particular choices of Q_{ij} yield constraints that are intuitively very useful and are usually sufficient in practice. The constraints with the largest Q_{ij}, that is $Q_{ij} = A^c_{ij}$ for every c, were used in the proof of Theorem 1, and we have found them to be useful in practice. The relaxation of IP2 already includes all the constraints with Q_{ij} corresponding to single Y_{ujc} variables. Rather than including constraints with $1 < |Q_{ij}| < |A^c_{ij}|$, we include the constraints with i and j reversed, which can be seen to be weaker versions of constraints (1) with larger Q_{ij} sets. Examples can be constructed for which the constraints we use in practice are insufficient to make LP2 as tight as LP1. However, we have not encountered such pathological cases in biological data. More detail about our approach to and experiences with real problems can be found in Section 5.

5 Computational Results

We apply our LP formulation to find binding sites for *E. coli* transcription factors, and we show that in practice our LP formulation results in significantly faster running times than the previous simpler linear program. Moreover, in order to demonstrate that our formulation of the motif finding problem results in biologically relevant solutions, we show that our approach identifies binding sites as well as a widely-used probabilistic technique [20].

5.1 Test Sets

We present results on identifying the binding sites of 39 *E. coli* transcription factors (see Table 1). We construct our data set from the data of [17, 14] in a fashion similar to [15]. In short, we remove all sites for sigma-factors, duplicate sites, as well as those that could not be unambiguously located in the genome. Data sets for all factors with only two sites remaining were discarded as uninteresting for motif finding; datasets for *ihf* and *crp* are omitted due to size considerations. For each transcription factor considered, we gather at least 300 base pairs of genomic sequence upstream of the transcription start sites of the regulated genes. In the cases where the binding site is located further upstream, we extend the sequence to include the binding site. This results in graphs with up to 20 parts and 5,960 nodes. The motif length for each dataset was chosen based on the length of the consensus binding site, determined from other biological studies and ranging between 11 and 48. The transcription factors, the length of their binding site, and the number of DNA sequences considered are shown in Table 1.

TF	ℓ	p	n	TF	ℓ	p	n	TF	ℓ	p	n	TF	ℓ	p	n
ada	31	3	810	farR	10	3	873	hns	11	5	1485	ompR	20	9	3057
araC	48	6	1715	fis	35	18	5371	lexA	20	19	5554	oxyR	39	4	1048
arcA	15	13	4790	flhCD	31	3	810	lrp	25	14	4090	phoB	22	14	4618
argR	18	17	5960	fnr	22	12	3705	malT	10	10	3410	purR	26	20	5856
cpxR	15	9	2614	fruR	16	11	4082	metJ	16	15	5754	soxS	35	13	4004
cspA	20	4	1410	fur	18	9	3182	metR	15	8	3312	torR	10	4	2198
cysB	40	3	783	galR	16	7	2188	modE	24	3	934	trpR	24	4	1108
cytR	18	5	1695	gcvA	20	4	1234	nagC	23	6	1870	tus	23	5	1390
dnaA	15	8	2381	glpR	20	11	3829	narL	16	10	3301	tyrR	22	17	5258
fadR	17	7	2122	hipB	30	4	1084	ntrC	17	5	1516				

Table 1: Sizes for the 39 problems considered: number of sequences (p), motif length (ℓ), and total number of nodes in the underlying graph (n).

5.2 Methodology

We first solve the LP relaxation of IP2. If the solution is not integral, we find and add violated constraints and re-solve. We have observed that certain classes of constraints of the form (1) are powerful in practice, and so we consider these first:

1. $Q_{ij} = A_{ij}^c$ for every $i < j, c$.
2. $Q_{ij} = \{(u, j, c) : c \in D\}$ for every $i < j, u \in V_i$.

In addition, we consider the above constraints with $i > j$. We iterate, adding all violated constraints of the above types and re-solving, until all such constraints are satisfied. While in theory this heuristic approach may lead to a solution that is not as tight as that of LP1, in all cases considered, we find that adding this particular set of constraints is sufficient for making LP2 as tight as LP1. Moreover, in practice, this heuristic approach will be faster than using the ellipsoid method [6] with our separation algorithm and, we show below, is usually faster than solving LP1.

LP1 was solved using two different simplex variants. In the first (`primal dualopt`), the primal problem was solved using the dual simplex algorithm. In the second (`dual primalopt`), the dual problem was solved using the primal simplex algorithm. LP2 was always solved using the dual simplex method applied to the primal problem so that we could use the optimal basis of the previous iteration as a starting point for the next, setting the dual variables for the added constraints to be basic. This strategy eliminates the need to re-solve using an arbitrary starting solution and provides a significant speedup.

The linear and integer programs were specified in the mathematical programming modeling language Ampl and solved using CPLEX 7.1. All experiments were run on a public 1.2 GHz SPARC workstation using a single processor. All the timings reported are in CPU seconds. Any problem taking longer than five hours was aborted. Interestingly, only 3 of the 34 problems solvable in less than five hours using either LP1 or LP2 were not integral. Since the problem is NP-complete, this is somewhat surprising. This suggests that handling non-integral cases may not be as pressing an issue as one would think.

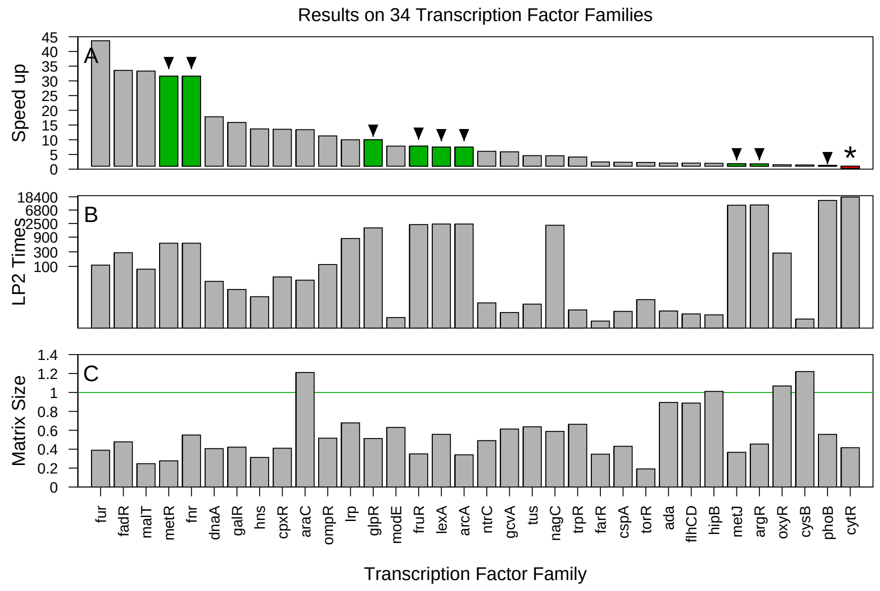

Fig. 4: (a) Speed-up factor of LP2 over LP1. A triangle indicates problems for which LP1 did not finish in less than five hours. An asterisk (far right) marks the problem for which LP2 did not finish in less than five hours, but LP1 did. (b) Running times in seconds for LP2 (log scale). (c) Ratio of matrix sizes for LP2 to LP1.

5.3 Performance of the LP relaxations

We solved LP1 and LP2 relaxations for the transcription factors listed in Table 1. Fig. 4 plots the running times, matrix sizes, defined as the number of constraints times the number of variables, and speed-up factors of LP2 over LP1. For five problems, each LP failed to find a solution in the allotted five hours; these are omitted from the figure. In most cases, the initial set of constraints was sufficient to get a solution at least as good as that obtained by LP1. Six problems required additional constraints to LP2 to make their solutions as tight. The problems *flhCD*, *torR*, and *hu* required two iterations of adding violated constraints, *ompR* required three, *oxyR* four, and *nagC* five. Running times reported in Fig. 4(b) are the sum of the initial solve times and of all the iterations. Fig. 4(c) plots (size of LP2)/(size of LP1). As expected, the size of the constraint matrix is typically smaller for LP2. While in four cases the matrix for LP2 is larger, often it is less than 50% the size of the matrix for LP1.

When comparing the running times of LP2 with those of LP1, the speed-up factor is computed as min{`primal dualopt LP1`, `dual primalopt LP1`}/`LP2`, that is, using the better running time for LP1. For all but one of the datasets, a significant speed-up when using LP2 is observed, and an order of magnitude speed-up is common, as shown in Fig. 4(a). For nine problems, while LP2 was solved, neither simplex variant completed in less than five hours when solving LP1. For these problems, the timing for LP1 was set at five hours, giving

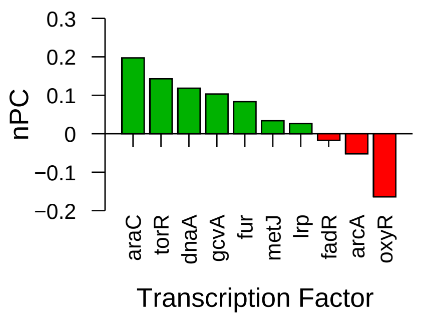

Fig. 5: Difference between nPC values obtained using the ILP approach and Gibbs Motif Sampler [20]; data sets with identical motifs are omitted. Bars above zero indicate that ILP performs better.

a lower bound on the speed up. For one problem, *cytR*, the reverse was true and LP2 did not finish within five hours, while LP1 successfully solved the problem. For this dataset, the timing for LP2 was taken to be five hours, giving an upper bound.

We also compared the performance of our approach, measured by the nucleotide performance coefficient (nPC) [21], in identifying existing transcription factor binding sites to that of Gibbs Motif Sampler [20]. The nPC measures the degree of overlap between known and predicted motifs, and is defined as $nTP/(nTP + nFN + nFP)$, where nTP, nFP, nTN, nFN refer to nucleotide level true positives, false positives, true negatives and false negatives respectively. We compare the nPC values for the two methods in Fig. 5. Each bar in the chart measures the difference in nPC between the ILP approach and Gibbs Motif Sampler, omitting those transcription factor datasets for which the found motifs are identical. Of the 30 problems for which the integral optimal was found using LP2, the sum-of-pairwise hamming distances measure more accurately identifies the biologically known motif in seven cases, with nPC 0.11 better on average. In 20 cases, the two methods find identical solutions. In the remaining 3 cases, Gibbs sampling does better, with nPC 0.08 better on average. Since the Gibbs sampling approaches have comparable performance to other stochastic motif finding methods [21] and most combinatorial methods are restricted by the lengths of the motifs considered, our ILP framework provides an effective alternative approach for identifying DNA sequence motifs.

6 Conclusions

We introduced a novel ILP for the motif finding problem and showed that it works well in practice. There are many interesting avenues for future work. While the underlying graph problem is similar to that of [4, 9], one central difference is that the edge weights satisfy the triangle inequality. In addition, edge weights in the graph are not independent, as each node represents a subsequence from a sliding window. Incorporating these features into the ILP may lead to further advances in computational methods for motif finding. It would also be useful to extend the basic formulation presented here to find multiple co-occurring or repeated motifs (as supported by many widely-used packages). Finally, we note that graph pruning and decomposition techniques (e.g., [16, 23]) may allow mathematical programming formulations to tackle problems of considerably larger size.

Acknowledgments. M.S. thanks the NSF for PECASE award MCB-0093399 and DARPA for award MDA972-00-1-0031. The authors thank Dan Gusfield for helpful comments on an earlier version of this manuscript.

References

1. Akutsu, T., Arimura, H., Shimozono, S. On approximation algorithms for local multiple alignment. RECOMB, 2000, pp. 1–7.
2. Bafna, V., Lawler, E., Pevzner P. A. Approximation algorithms for multiple alignment. Theoretical Computer Science 182: 233–244, 1997.
3. Bailey, T., Elkan, C. Unsupervised learning of multiple motifs in biopolymers using expectation maximization. Machine Learning 21: 51–80, 1995.
4. Chazelle, B., Kingsford, C., Singh M. A semidefinite programming approach to side-chain positioning with new rounding strategies, INFORMS J. on Computing 16: 380–392, 2004.
5. Cook, W., Cunningham, W., Pulleyblank, W., Schrijver, A. Combinatorial Optimization. Wiley-Interscience, New York, 1997.
6. Grötschel, M., Lovász, L., Schrijver, A. 1993. Geometric Algorithms and Combinatorial Optimization. Springer-Verlag, Berlin, Germany, 2nd edition.
7. Hertz, G., Stormo, G. Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. Bioinf. 15: 563–577, 1999.
8. Kellis, M. Patterson, N., Endrizzi, M., Birren, B., Lander E. Sequencing and comparison of yeast species to identify genes and regulatory elements. Nature 423: 241–254, 2003.
9. Kingsford, C., Chazelle, B., Singh, M. Solving and analyzing side-chain positioning problems using linear and integer programming. Bioinf. 21: 1028–1039, 2005.
10. Lawrence, C., Altschul, S., Boguski, M., Liu, J., Neuwald, A., Wootton, J. Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. Science 262: 208–214, 1993.
11. Lee, T., Rinaldi, N., Robert, F., Odom, D., Bar-Joseph, Z., Gerber, G. *et al.* Transcriptional regulatory networks in *S. cerevisiae.* Science 298: 799–804, 2002.
12. Li, M., Ma, B., Wang, L. Finding similar regions in many strings. J. Computer and Systems Sciences 65(1): 73–96, 2002.
13. Marsan L., Sagot M.F. Algorithms for extracting structured motifs using a suffix tree with an application to promoter and regulatory site consensus identification. J. Comp. Bio. 7:345-362, 2000.

14. McGuire, A., Hughes, J., Church, G. Conservation of DNA regulatory motifs and discovery of new motifs in microbial genomes. Genome Res. 10: 744–757, 2000.
15. Osada, R., Zaslavsky, E., Singh, M. Comparative analysis of methods for representing and searching for transcription factor binding sites. Bioinf. 20: 3516–3525, 2004.
16. Pevzner, P., Sze, S. Combinatorial approaches to finding subtle signals in DNA sequences. ISMB, 2000, pp. 269–278.
17. Robison, K., McGuire, A., Church, G. A comprehensive library of DNA-binding site matrices for 55 proteins applied to the complete *Escherichia coli* K-12 Genome. J. Mol. Biol. 284: 241–254, 1998.
18. Schuler, G., Altschul, S., Lipman, D. A workbench for multiple alignment construction and analysis. Proteins 9(3): 180–190, 1991.
19. Tavazoie, S., Hughes, J., Campbell, M., Cho, R., Church, G. Systematic determination of genetic network architecture. Nat. Genetics 22(3): 281–285, 1999.
20. Thompson, W., Rouchka, E., Lawrence, C. Gibbs Recursive Sampler: finding transcription factor binding sites. Nucleic Acids Res. 31: 3580-3585, 2003.
21. Tompa, M., Li, N., Bailey, T., Church, G., De Moor, B., Eskin, E., *et al.* Assessing computational tools for the discovery of transcription factor binding sites. Nat. Biotech. 23: 137–144, 2005.
22. Wang, L., Jiang, T. On the complexity of multiple sequence alignment. J. Comp. Bio. 1: 337–348, 1994.
23. Zaslavsky, E., Singh, M. Combinatorial Optimization Approaches to Motif Finding. Submitted. Also available as Princeton University Computer Science Dept. Technical Report TR-728-05.