The Implicit Hitting Set Approach to Solve Combinatorial Optimization Problems with an Application to Multigenome Alignment

Erick Moreno-Centeno
Department of Industrial and Systems Engineering, Texas A&M University, College Station, Texas 77843, e.moreno@tamu.edu

Richard M. Karp
Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, Berkeley, California 94720; and International Computer Science Institute, Berkeley, California 94704, karp@cs.berkeley.edu

We develop a novel framework, the \textit{implicit hitting set approach}, for solving a class of combinatorial optimization problems. The explicit hitting set problem is as follows: given a set $U$ and a family $\mathcal{S}$ of subsets of $U$, find a minimum-cardinality set that intersects (hits) every set in $\mathcal{S}$. In the implicit hitting set problem (IHSP), the family of subsets $\mathcal{S}$ is not explicitly listed (its size is, generally, exponential in terms of the size of $U$); instead, it is given via a polynomial-time oracle that verifies if a given set $H$ is a hitting set or returns a set in $\mathcal{S}$ that is not hit by $H$. Many NP-hard problems can be straightforwardly formulated as implicit hitting set problems. We show that the implicit hitting set approach is valuable in developing exact and heuristic algorithms for solving this class of combinatorial optimization problems. Specifically, we provide a generic algorithmic strategy, which combines efficient heuristics and exact methods, to solve any IHSP. Given an instance of an IHSP, the proposed algorithmic strategy gives a sequence of feasible solutions and lower bounds on the optimal solution value and ultimately yields an optimal solution. We specialize this algorithmic strategy to solve the multigenome alignment problem and present computational results that illustrate the effectiveness of the implicit hitting set approach.

Subject classifications: combinatorial optimization; exact algorithms; heuristic algorithms; multigenome alignment problem; multisequence alignment problem.

Area of review: Optimization.

History: Received October 2011; revisions received July 2012, October 2012; accepted October 2012.

1. Introduction

In an implicit combinatorial optimization problem, the constraints are not listed explicitly but are specified implicitly with auxiliary algorithms or constraints. Two examples are integer programming and convex optimization problems. In an integer program, the constraints that define the convex hull of the (integer) solutions are implied by the constraints of a linear program. Similarly, in many convex optimization problems, the feasible region is defined implicitly through a separation oracle: a polynomial-time algorithm that, given a candidate solution, either certifies that the candidate solution is feasible or identifies a linear constraint that is violated by the candidate solution but is satisfied by all feasible solutions. This paper concerns a large class of implicit combinatorial optimization problems, which we refer to as \textit{implicit hitting set problems}.

The explicit hitting set problem (EHSP) is defined as follows (Karp 1972): given a finite \textit{ground set} $U$ of elements, a weight for each element, and a family $\Gamma$ of subsets of $U$, find a subset $H \subset U$ of minimum total weight that intersects/hits every set in $\Gamma$. The EHSP is identical to the classic weighted set-cover problem, except that the roles of sets and elements are interchanged. In an implicit hitting set problem (IHSP) the family $\Gamma$ of subsets of $U$ (called \textit{circuits}) is not listed explicitly but is specified implicitly by a separation oracle: a polynomial-time algorithm, which given a set $H \subset U$, either certifies that $H$ is a hitting set or returns a circuit that is not hit by $H$.

Because the IHSP is NP-complete, it follows from the theory of NP-completeness that every NP-complete problem can be represented as an IHSP (and any other NP-hard problem). However, the IHSP is important because, as shown in §3, several well-known NP-hard combinatorial optimization problems can be reformulated in relatively simple ways as IHSPs. We refer to these problems as \textit{implicit hitting set problems}. These problems include feedback vertex set in a graph, feedback vertex set in a digraph, max cut, maximum 2-sat, the Steiner tree problem (in graphs), the traveling salesman problem, $k$-matroid intersection, finding a maximum feasible set of linear constraints (defined in §2), and, the problem on which we illustrate the effectiveness of our approach, the multigenome alignment problem (defined in §5).
This paper develops a generic algorithmic framework, referred to as the implicit hitting set approach, to solve the aforementioned class of combinatorial optimization problems. This solution approach integrates exact and heuristic algorithms and exploits the capabilities of commercially available linear and integer programming solvers. More specifically, a central idea of our approach is to use a heuristic algorithm for the EHSP, not only to find feasible solutions for the EHSP (which can be easily extended to get feasible solutions for the IHSP) but also to help in building a small set of “important” circuits that provide a strong relaxation of the IHSP (used to obtain bounds). Thus, the heuristic algorithm is used as a mechanism to guide the search process and generate useful information that is later used by the exact algorithm. Indeed, an important characteristic of this methodology is that it generates, at each iteration, both a feasible solution for the IHSP and a bound (ultimately yielding an optimal solution and a matching bound that certifies its optimality). Finally, a key component in the framework is the use of randomization to diversify the circuits generated by the separation oracle. This diversification greatly improves the strength of the IHSP’s relaxations as well as the quality of the solutions found by the EHSP’s heuristic.

The remainder of this paper is organized as follows. Section 2 briefly reviews some relevant literature. Section 3 presents the implicit hitting set problem, describes its complexity, and shows how several well-known NP-hard combinatorial optimization problems can be reformulated as IHSPs. Section 4 introduces the main ideas and describes how to apply our solution approach to any generic IHSP. Section 5 gives a precise definition of the multigenome alignment problem (MGAP) and reviews some relevant literature for the MGAP. Section 6 shows how to apply the implicit hitting set approach to solve the MGAP, and §6.1 focuses on describing the separation oracles developed for the implicit hitting set formulation of the MGAP. Section 7 reports an extensive computational study, which shows that for most practical-sized instances of the multigenome alignment problem (equivalently any multisequence alignment problem), our algorithm finds an optimal solution quickly. Finally, §8 provides concluding remarks and future research directions.

2. Literature Review
A key feature of the implicit hitting set approach proposed here is that it makes use of both exact and heuristic algorithms to solve the problem. This integration has been explored in other contexts (see, e.g., Franceschi et al. 2006, Archetti et al. 2008, Savelsbergh and Song 2008, Schmid et al. 2009, Hewitt et al. 2010).

The implicit hitting set approach reformulates an NP-hard combinatorial optimization problem as an explicit hitting set problem with an exponential number of constraints and then deals with those constraints implicitly. A closely related general framework to solve some combinatorial optimization problems is to reformulate them as a set-cover problem with an exponential number of columns (closely related because the implicit hitting set problem is identical to the set-cover problem). For example, Monaci and Toth (2006) used this idea as the basis of their set-covering-based heuristic approach for bin-packing problems, and Muter et al. (2010) proposed a combination of metaheuristic and exact algorithms for solving set-covering-type optimization problems with an exponential number of columns (they applied their framework to solve the vehicle routing problem with time windows). In this sense, the implicit hitting set approach can be thought of as a type of branch-and-cut framework (see, e.g., Nemhauser and Wolsey 1988), and the set-covering-type approach can be thought of as a type of branch-and-price framework (Barnhart et al. 1998).

A problem that has received a lot of attention, and that can be solved using the implicit hitting set approach, is the problem of finding a maximum feasible subsystem of linear constraints within a given infeasible system of linear constraints (MAX-FS). Sankaran (1993) and Chakravarti (1994) noted that MAX-FS is NP-hard. Amaldi et al. (1999) showed that MAX-FS is equivalent to finding the minimum number of linear constraints to remove such that the retained constraints constitute a feasible system—a.k.a. the minimum unsatisfied linear relation problem (MIN-ULR). Chinneck and Dravnieks (1991) observed that all infeasible systems have one or more irreducible infeasible subsystems (IISs) of constraints, where an IIS is an infeasible subsystem whose proper subsystems are all feasible. Because deleting at least one of its members eliminates an IIS, MIN-ULR is equivalent to finding the smallest cardinality set of constraints to cover (hit) all IISs (MIN-IIS-COVER) (Chinneck 1996); thus, MIN-IIS-COVER is MAX-FS reformulated as an implicit hitting set problem. Parker and Ryan (1996) solved MIN-IIS-COVER using a similar approach to the implicit hitting set approach. Specifically, their approach is identical to the naive algorithm (Algorithm 1) given in §4. The main differences between Parker and Ryan’s approach and the framework proposed here (Algorithm 3 and the implementation details described in §6.2) are that their approach does not find a feasible solution at every iteration and their approach does not use randomization to diversify the IISs (which correspond to the circuits defined in the second paragraph of the introduction) generated by the separation oracle. For further details on the MAX-FS, MIN-ULR, and MIN-IIS-COVER (equivalent) problems, we refer to Chinneck (2001, 2008).

3. The Implicit Hitting Set Problem
An independence system is specified by a finite ground set $U$ of elements, a weight $w(x)$ for each element $x$ in $U$, and a family $\Gamma$ of subsets of $U$ called circuits, such that no circuit contains another circuit. The weight of a subset of $U$ is the sum of the weights of its elements. A hitting set is a set that “hits” every circuit, i.e., that has a nonempty intersection with every circuit.
The EHSP is to find a hitting set of minimum weight. The EHSP is identical to the classic weighted set-cover problem, except that the roles of sets and elements have been interchanged. It is NP-hard (Karp 1972) and cannot be approximated in polynomial time within a factor \((1 - \varepsilon)\ln |U|\) (Lund and Yannakakis 1994, Feige 1998). A simple polynomial-time greedy algorithm (described below) achieves the worst-case approximation ratio of \(O(\ln(|U|))\) (Johnson 1974, Lovasz 1975, Chvatal 1979); indeed, Slavik (1997) proved that the worst-case performance ratio of the greedy algorithm is exactly \(\ln |U| - \ln \ln |U| + \Theta(1)\). However, this bound is very pessimistic in practice; in the authors’ limited experience, the greedy algorithm was extremely quickly, removing them from the hitting set is possible. Because these redundant elements can be found can be removed from the hitting set without losing feasibility. A simple, yet efficient and effective, way to improve the quality of the heuristic solutions found by the greedy algorithm has always yielded an approximate solution whose weight is within a few percent of optimal.

The greedy algorithm maintains a candidate hitting set \(H \subseteq U\) and a set \(\Gamma'\) of circuits that are not hit by any element in \(H\). Initially, \(H\) is empty and \(\Gamma'\) is equal to \(\Gamma\). The algorithm constructs a hitting set by repeating the following operations until \(\Gamma'\) is empty (and thus \(H\) is a feasible solution for the EHSP): (a) append \(H\) with the element \(x\) in \(U\setminus H\) that has the minimum ratio of weight \(w(x)\) to the number of circuits in \(\Gamma'\) that are hit by \(x\) (ties are broken arbitrarily); (b) update \(\Gamma'\).

During our experimental study we found out that, even though the greedy algorithm consistently delivered solutions that were within a few percent of optimality, on average 5% of the elements included in the greedy solution were redundant. Given a feasible solution (i.e., a hitting set) for the EHSP, a redundant element is an element that can be removed from the hitting set without losing feasibility. Because these redundant elements can be found extremely quickly, removing them from the hitting set is a simple, yet efficient and effective, way to improve the quality of the heuristic solutions found by the greedy algorithm. We refer to the combination of the greedy algorithm with this improvement strategy as the improved-greedy algorithm.

In the 0-1 integer programming formulation of the EHSP, we denote as \(w_j\) the weight of element \(j\) and have a 0-1 variable \(x_j\) indicating whether element \(j\) is selected. The integer program is as follows: minimize \(\sum_j w_jx_j\) subject to the following constraints: \(\sum_{j \in \gamma} x_j \geq 1\), for each \(\gamma \in \Gamma\). The linear programming relaxation, in which \(x_j\) is allowed to be a real number in \([0,1]\), is very efficiently solvable by a variety of methods (see, e.g., Plotkin et al. 1995, Koufogiannakis and Young 2007).

For an IHSP the set of circuits is not listed explicitly but instead specified implicitly by a separation oracle: a polynomial-time algorithm, which given a set \(H \subseteq U\), either certifies that \(H\) is a hitting set or returns a circuit that is not hit by \(H\). Next, we show how several combinatorial optimization problems can be formulated as implicit hitting set problems. In all of these problems, the circuits can be thought of as being undesirable substructures where each circuit is destroyed by removing one of its elements; the IHSP is then to find the minimum weight set of elements in \(U\) that destroys all of the undesirable substructures. These problems can be classified into two categories: (A) those whose optimal solution is the optimal hitting set \(H^*\), and (B) those whose optimal solution is the complement of the optimal hitting set \(U \setminus H^*\).

Combinatorial optimization problems in category A include the following:

- **Minimum feedback vertex set in a graph or digraph.** Ground set: set of vertices of graph \(G = (V, E)\) or digraph \(G = (V, A)\). Circuits: node sets of simple cycles.
- **Minimum feedback arc set in a digraph.** Ground set: set of edges of digraph \(G = (V, A)\). Circuits: edge sets of simple cycles.
- **Steiner tree (in graphs).** Input: a graph \(G = (V, E)\), a subset \(R \subseteq V\) of required vertices, and an edge-weight function \(w: E \to \mathbb{R}\). The problem is to find a minimum-weight sub-tree in \(G\) that spans \(R\). Ground set: set of edges of graph \(G = (V, E)\). Circuits: edge sets of bipartite subgraphs \(G' = (V_1 \cup V_2, E' \subseteq E)\) of \(G\) such that both \(V_1\) and \(V_2\) contain at least one vertex in \(R\).
- **Synchronization in an acyclic digraph.** Ground set: a collection \(U\) of pairs of vertices drawn from the vertex set of an acyclic digraph \(G\). Circuits: minimal collections \(C\) of pairs from \(U\) with the property that, if each pair in \(C\) is contracted to a single vertex, then the resulting digraph contains a cycle.

In one application of this problem, the acyclic digraph represents precedence relations among a set of tasks. Two tasks are synchronized if they are executed at the same moment in time. The given pairs of vertices represent pairs of tasks that we wish to synchronize, and the hitting set problem is to delete a minimum-weight set of pairs such that there is a schedule respecting the precedence constraints in which the remaining pairs can be synchronized. An entirely different application, involving synchronization in space rather than time, is discussed in §5.

Combinatorial optimization problems in category B include the following:

- **Max cut.** Ground set: Edges set of graph \(G = (V, E)\). Circuits: edge sets of simple odd cycles.
- **k-matroid intersection.** Ground set: common ground set of the \(k\) matroids. Circuits: circuits in the \(k\) matroids.
- **Max 2-sat.** Ground set: for each clause \(x \lor y\), where \(x\) and \(y\) are literals, the pair of (equivalent) implications \(\bar{x} \leftrightarrow y\) and \(\bar{y} \leftrightarrow x\), viewed as edges of a directed graph. Circuits: edge sets of directed cycles containing as vertices both a variable and its complement.
- **Directed traveling salesman problem (TSP).** Ground set: Arc set of digraph \(G = (V, A)\). Circuits: pairs of incoming (outgoing) arcs to a node, and arc sets of subtours.
• Undirected TSP. Ground set: set of edges of graph \( G = (V, E) \). Circuits: triplets of edges incident to a node, and edge sets of simple cycles of length \( < |V| \) (i.e., subtours).

With the exception of the TSPs, all of the problems in category A and B are, as expected, minimization and maximization problems, respectively. Similarly, those formulations are straightforward. However, the IHSP formulations of the TSPs deserve some discussion. For simplicity, the discussion is focused on the undirected TSP. Let \( H^* \) denote an optimal hitting set and \( \bar{H}^* = U \setminus H^* \) denote its complement. We now argue that the \( \bar{H}^* \) forms an optimal TSP tour. First, because any feasible hitting set must contain at least one edge from each triplet of edges incident to a node, then its complement induces a graph whose nodes have degree less than or equal to two; thus, \( \bar{H}^* \) is composed of only paths and cycles. Second, because any feasible hitting set must contain at least one edge from each simple cycle of length strictly less than \( |V| \), then \( \bar{H}^* \) does not contain subtours. Finally, in order to guarantee that \( \bar{H}^* \) has exactly \( |V| \) edges (and thus forms a Hamiltonian tour) and that its total weight is minimized, the edge weights are set equal to a large constant minus the original edge weight. With this adjustment, the objective of minimizing the number of edges in the hitting set is lexicographically more important than the objective of maximizing the weight of the hitting set. Equivalently, the objective of maximizing the number of edges in \( \bar{H}^* \) is lexicographically more important than the objective of minimizing the weight of the edges in \( H^* \).

4. A Generic Algorithm for Implicit Hitting Set Problems

This section presents an algorithm to solve any IHSP. It first presents a naive algorithm and then an improved one, the generic algorithm. The algorithms’ correctness follows from Observation 1.

Observation 1. For any IHSP, given an explicit list \( \Gamma \) of its circuits, the EHSP defined by \( \Gamma \) is a relaxation of the IHSP. Therefore, a lower bound for the EHSP is also a lower bound for the IHSP, and if an optimal solution for the EHSP is feasible for the IHSP, then it is optimal for the IHSP.

The naive algorithm is a dialogue between a separation oracle for the IHSP, and an exact/optimal algorithm to solve the EHSP (e.g., an integer programming solver). The naive algorithm, Algorithm 1, maintains an explicit list \( \Gamma \) of circuits and an optimal hitting set \( H \) for the EHSP defined by \( \Gamma \).

Algorithm 1 (Naive algorithm)

1. \( \Gamma \leftarrow \emptyset \); \( H \leftarrow \emptyset \)
2. while Separation oracle finds a circuit that is not hit by \( H \) do
3. Add the circuit to \( \Gamma \)
4. \( H \leftarrow \text{optimal hitting set for the EHSP defined by } \Gamma \)
5. end
6. return \( H \)  // \( H \) is optimal for the IHSP

Because the EHSP is NP-hard, the running time of the naive algorithm is dominated by the time spent solving the sequence of EHSPs to optimality (line 4). The generic algorithm aims to reduce the number of calls to the exact EHSP algorithm and to find a feasible solution for the IHSP at each iteration. The reduction in the number of calls is achieved by the following two methods:

1. Instead of solving optimally the EHSP between each call to the separation oracle, the generic algorithm solves most of the EHSPs with a heuristic algorithm. Specifically, it solves the EHSP exactly only when the separation oracle does not find a circuit not hit by the EHSP’s heuristic solution; i.e., when the solution found by the heuristic is a feasible solution to the IHSP.

2. Instead of adding a single circuit to \( \Gamma \) and then solving (either heuristically or optimally) the EHSP defined by \( \Gamma \), the generic algorithm adds at once a collection of circuits that are not hit by \( H \). We refer to the procedure used to find such a collection of circuits as a circuit-generation routine. We show this strategy gives as a by-product a feasible solution for the IHSP.

The idea underlying the generic algorithm is to build up a small set of “important” circuits in the hope that before too many circuits have been added to \( \Gamma \), the optimal hitting set \( H \) for this explicit set of circuits will in fact hit all the circuits of the IHSP. In this case, \( H \) will be optimal for the IHSP. The heuristic algorithm is used as a device for finding important circuits rapidly, without the expense of constructing an optimal hitting set for \( \Gamma \). The use of the heuristic algorithm is a win-win proposition: it either provides a feasible solution for the IHSP or it allows the circuit-generation routine to yield a new collection of circuits.

In addition to reducing the number of EHSPs solved throughout the execution of the generic algorithm, using the circuit-generation routine has the advantage of generating a feasible solution to the IHSP. To observe this, we describe how the circuit-generation routine, Algorithm 2, works: it receives as inputs \( \Gamma \) and a hitting set \( H \) for \( \Gamma \), and invokes the separation oracle. If the separation oracle finds a circuit not hit by \( H \), then it augments \( \Gamma \) and \( H \) with the circuit found and an arbitrary element in such circuit, respectively. The circuit-generation routine invokes the separation oracle until it does not find a circuit not hit by \( H \). At termination of the circuit-generation routine, the augmented \( \Gamma \) and \( H \) are returned; note that the returned \( H \) is a feasible solution to the IHSP.

Algorithm 2 (Circuit-generation routine)

Input: \( \Gamma \) and a hitting set \( H \) for \( \Gamma \)
Output: An augmented list of circuits \( \Gamma \), and an augmented hitting set \( H \) for the IHSP

1. while Separation oracle finds a circuit that is not hit by \( H \) do
2 Add the circuit to \( \Gamma \)
3 Add one element of the circuit to \( H \)
4 end
5 return \( \{ \Gamma, H \} \)  // \( H \) is feasible for the IHSP

A feasible solution to the IHSP of particular interest is the best feasible solution found before the first call to the exact EHSP algorithm; we refer to this solution as the heuristic solution. This solution has the property that it is the hitting set found by the heuristic algorithm for the EHSP defined by \( \Gamma \); specifically, this hitting set was not augmented during a circuit-generation routine. As shown in §7, the heuristic solution is found extremely quickly; moreover, during our experimental study, the heuristic solution was always within a few percent of optimal.

So far, we have described two methods that the generic algorithm uses to find feasible solutions for the IHSP. Next, we describe two methods for getting lower bounds on the solution value of the IHSP. The first method is to solve the LP relaxation of the EHSP defined by \( \Gamma \), and the second method is to solve the EHSP to optimality. The validity of these lower bounds follows from Observation 1. Needless to say, the lower bound obtained with the second method is much stronger than the lower bound obtained with the first method. Indeed, unless otherwise stated, we only need to compute a lower bound using the first method whenever the time limit is reached and we have not obtained a lower bound using the second method.

A final simple but effective improvement incorporated into the generic algorithm is the use of the improved-greedy algorithm (described in §3) instead of the usual greedy algorithm.

The generic algorithm, Algorithm 3, maintains an explicit list \( \Gamma \) of circuits, a hitting set \( H \) for \( \Gamma \), the best hitting set for the IHSP found so far \( \bar{H} \), and an optimal hitting set \( \bar{H} \) for the EHSP defined by \( \Gamma \). The generic algorithm combines all of the aforementioned ideas, and clearly its effectiveness depends on (1) the choice of separation oracle, (2) the speed of the heuristic algorithm (here we use the improved-greedy algorithm) and quality of the solutions it produces, and (3) the speed of the exact algorithm (here we use IBM's general-purpose IP solver CPLEX) for the EHSPs.

Less obvious is the fact that the effectiveness of the generic algorithm also depends on the diversity of the circuits found by the separation oracle. This diversity is so important that, during the experimental study (§7), the running times of the algorithm without using a diversification strategy were more than two orders of magnitude slower than those obtained using diversification strategies. Specifically, in the implementation of the generic algorithm to solve the multigenome alignment problem (described in §6), we used the following diversification strategies: (1) developed and simultaneously used two separation oracles that identified structurally different circuits; (2) the circuit-generation routine randomly selected the edge to be deleted (instead of using a fixed rule such as deleting the edge that had been deleted the minimum number of times); and, (3) randomly selected the order in which the separation oracles visited the genomes.

**Algorithm 3** (Generic algorithm)

1 \( \Gamma \leftarrow \emptyset \)
2 \( H \leftarrow \emptyset \)
3 \( \bar{H} \leftarrow U \)
4 repeat
5  repeat
6     \( \{ \Gamma, H \} \leftarrow \text{Circuit-Generation Routine}(\Gamma, H) \)
7     if weight\((H) < \) weight\((\bar{H}) \) then \( \bar{H} \leftarrow H \)
8     \( H \leftarrow \) Use improved greedy to solve the EHSP defined by \( \Gamma \)
9 until Separation oracle asserts that \( H \) is feasible for IHSP
10 if weight\((H) < \) weight\((\bar{H}) \) then \( \bar{H} \leftarrow H \)
11 \( H \leftarrow \) optimal hitting set for the EHSP defined by \( \Gamma \)
12 if weight\((H) = \) weight\((\bar{H}) \) then return \( \bar{H} \)  // \( \bar{H} \) optimal for IHSP
13 if Separation oracle asserts \( H \) is feasible for IHSP then return \( \bar{H} \)  // \( \bar{H} \) optimal for IHSP
14 \( H \leftarrow H \)  // Restart using as starting point the EHSP's optimal solution
15 until some termination condition is met  
  (e.g., in time or optimality gap)

In many cases, one can design the separation oracle to construct a minimum-cardinality circuit that is not hit by the current set \( H \). For example, in the feedback vertex set problem for a digraph \( G = (V, A) \), this amounts to finding a shortest cycle in the digraph induced by \( V \setminus H \), which is easily done using an all-pairs shortest path algorithm. During our computational experiments, we frequently observed that a hitting set for the collection of small-cardinality circuits will also be a hitting set for the collection of all circuits. In this case, the generic algorithm will succeed after generating a polynomial-bounded number of circuits. A companion paper (Chandrasekaran et al. 2011) shows that the implicit hitting set approach is also valuable in developing approximation algorithms. Specifically, that paper develops an online approximation algorithm for the minimum feedback vertex set problem on random graphs.

5. The Multigenome Alignment Problem

The MGAP is an important problem in computational biology. Loosely speaking, given a set of genomic sequences (sequences of nucleotides), the MGAP is to align the sequences so as to reveal their similarities. (Hereafter, we use the terms genome and sequence interchangeably.) Applications of the MGAP include predicting structural properties of unknown genomes based on known genomes and aiding in phylogenetic analysis (the study of organism relationships based on evolutionary similarities and differences).
There are several approaches to set up an MGAP; for a complete discussion, we refer to Heringa (2004) and Gotth et al. (2006) and the references therein. In this paper, we approach the MGAP with a so-called consistency-based method, which comprises the following two main steps.

**Step 1.** Find the most similar nucleotide subsequences (some researchers refer to the nucleotide subsequences as segments) between every pair of genomes. One method to achieve this is Smith and Waterman’s two-sequence local-alignment method (Smith and Waterman 1981); as explained below, here we use an alternative method.

The output of Step 1 is a set of diagonals. Each diagonal is a pair of nucleotide subsequences, called anchors, each on a different genome. In addition, each diagonal can have an associated weight. This weight aims to capture the similarity strength between the two anchors.

In the experimental study reported in §7, we performed Step 1 using the fast statistical alignment (FSA) software (Bradley et al. 2009). FSA is a multigenome-alignment program that is based on a pair of hidden Markov models (HMMs). These HMMs find the most similar nucleotide subsequences between every pair of genomes by approximating an insertion/deletion process. In FSA, each of the weights is the posterior probability that the two nucleotide subsequences are biologically related. For further details, we refer to Bradley et al. (2009).

Given a set of diagonals and associated anchors, each genome is a linearly ordered sequence of anchors. Hereafter, unless otherwise stated, we assume that none of these anchors overlap.

**Step 2.** Find the multigenome alignment (MGA) that is most consistent with the diagonals found in Step 1.

An MGA is a matrix $A$ that satisfies the following properties: (1) the entries $a_{ij}$ are either empty or contain an anchor; and (2) ignoring the empty entries, concatenating all the entries in any row $i$, we obtain the $i$th genome. Pictorially, an MGA can be described as follows: each of the genomes is assigned to a row, and each of its anchors is assigned to a column, so that the column assignments respect the linear order defined by the genomes.

The realized diagonals are those whose two anchors are assigned to the same column. Thus, in Step 2, we look for the MGA that maximizes the sum of weights of the realized diagonals. The rest of this paper shows how to reformulate and solve this problem (a precise definition of this problem is given in the next section) as an IHSP.

### 5.1. Graph-Theoretic Formulation

Several studies (e.g., Kececioglu 1993, Reinert et al. 1997) refer to Step 2 (defined in the previous section) as the maximum-weight trace problem (MTP). Informally, MTP is defined as follows: given a set of genomes and a set of weighted diagonals, find a trace with maximum total weight, where, informally, a trace is a subset of diagonals such that there exists an MGA where all of the diagonals in the trace are realized. To formally define the MTP, we need the following concepts.

**Definition 1.** An alignment graph is a mixed graph $G = (V, E, A)$ with nonnegative weights $w(e)$ assigned to the edges $e \in E$, where the nodes correspond to the anchors, the edges $E$ correspond to the diagonals, and there is an arc (a directed edge) between each anchor and the anchor that follows it in the genome. (Figure 1(a) illustrates an alignment graph.)

Even though, given the above definition, the terms node and anchor are equivalent; hereafter, we use the term anchor when referring to an actual subsequence in a genome and the term node when referring to the representation of an anchor in an alignment graph. Similarly, we use the term diagonal when referring to a pair of anchors in a genome and the term edge when referring to the representation of a diagonal in an alignment graph.

**Definition 2.** A trace $T \subseteq E$ is a subset of edges that can be realized in an alignment. That is, there exists a matrix $A$ that is assigned to a row, and each of its anchors is assigned to a column, so that the column assignments respect the linear order defined by the genomes.

The realized diagonals are those whose two anchors are assigned to the same column. Thus, in Step 2, we look for the MGA that maximizes the sum of weights of the realized diagonals. The rest of this paper shows how to reformulate and solve this problem (a precise definition of this problem is given in the next section) as an IHSP.

### 5.1. Graph-Theoretic Formulation

Several studies (e.g., Kececioglu 1993, Reinert et al. 1997) refer to Step 2 (defined in the previous section) as the maximum-weight trace problem (MTP). Informally, MTP is defined as follows: given a set of genomes and a set of weighted diagonals, find a trace with maximum total weight, where, informally, a trace is a subset of diagonals such that there exists an MGA where all of the diagonals in the trace are realized. To formally define the MTP, we need the following concepts.

**Definition 1.** An alignment graph is a mixed graph $G = (V, E, A)$ with nonnegative weights $w(e)$ assigned to the edges $e \in E$, where the nodes correspond to the anchors, the edges $E$ correspond to the diagonals, and there is an arc (a directed edge) between each anchor and the anchor that follows it in the genome. (Figure 1(a) illustrates an alignment graph.)

Even though, given the above definition, the terms node and anchor are equivalent; hereafter, we use the term anchor when referring to an actual subsequence in a genome and the term node when referring to the representation of an anchor in an alignment graph. Similarly, we use the term diagonal when referring to a pair of anchors in a genome and the term edge when referring to the representation of a diagonal in an alignment graph.

**Definition 2.** A trace $T \subseteq E$ is a subset of edges that can be realized in an alignment. That is, there exists a matrix $A$ that is assigned to a row, and each of its anchors is assigned to a column, so that the column assignments respect the linear order defined by the genomes.

The realized diagonals are those whose two anchors are assigned to the same column. Thus, in Step 2, we look for the MGA that maximizes the sum of weights of the realized diagonals. The rest of this paper shows how to reformulate and solve this problem (a precise definition of this problem is given in the next section) as an IHSP.
where each genome is assigned to a row, each pair of nodes connected by an edge in \( T \) appears in the same column, and every node is placed in a column with a higher index than that of the preceding node. The weight of a trace is calculated as \( \sum_{e \in T} w(e) \). (Figure 1(b) illustrates a trace and its corresponding alignment.)

Now, we can give a formal definition of MTP.

**Definition 3.** Given an alignment graph, the maximum-weight trace problem is to find a trace with maximum weight.

The unweighted versions of the multigenome alignment problem and the maximum-weight trace problem are those problems where the weights of all diagonals/edges are equal to one.

5.2. Complexity of Multigenome Alignment

The unweighted two-genome MTP can be solved by dynamic programming in \( O(n \log n) \) time, where \( n \) is the number of anchors/nodes (reduction to the longest common subsequence problem).

Kececioglu (1993) proved that the general MTP is NP-hard. However, for a fixed number of genomes, MTP can be solved in polynomial time by a dynamic programming (DP) algorithm that takes \( O(n^3) \) time and uses \( O(n^3) \) space. Gotoh et al. (2006) pointed out that the DP algorithm, even if cleverly implemented by exploiting heuristic upper bounds, is computationally infeasible for as few as 10 genomes/sequences.

6. Formulating and Solving the MTP as an IHSP

To formulate and solve the MTP as an IHSP, we need the following concepts.

**Definition 4.** Given an alignment graph \( G = (V, E, A) \), a mixed cycle is a cycle that contains at least one arc, and all of its arcs are traversed in the correct direction. (Figure 2(a) illustrates mixed cycles and cycles that are not mixed cycles because its arcs are not in the correct direction.)

**Theorem 2 (The Mixed Cycle Theorem, Kececioglu 1993).** Let \( G = (V, E, A) \) be an alignment graph, let \( T \) be a subset of \( E \), and let \( G' = (V, T, A) \) be the alignment graph induced by \( T \). Then, \( T \) is a trace if and only if there is no mixed cycle in \( G' \).

Using Theorem 2, illustrated in Figure 2(b), the MTP can be formulated as follows: given an alignment graph \( G = (V, E, A) \) with weights \( w(e) \) \( \geq 0 \) assigned to the edges \( e \in E \), find a subset \( T \subseteq E \) of maximum weight such that \( G' = (V, T, A) \) has no mixed cycle. Because finding a trace \( T \) of maximum weight is equivalent to finding the minimum-weight subset \( S \subseteq E \) such that \( G' = (V, E \setminus S, A) \) has no mixed cycle, then the MTP is equivalent to the following IHSP: the ground set is the edge set of the alignment graph \( G = (V, E, A) \). The circuits are the edge sets of simple mixed cycles.

Note that the optimal solution to this IHSP, a minimum-weight set \( S \subseteq E \) that hits every set in \( \Gamma \), readily gives a maximum-weight trace, \( T = E \setminus S \).

Algorithm 3 in §4 is an exact method to solve any IHSP formulation (and thus the MTP). As noted in §4, it can be used as a heuristic, and can be stopped at any time and give both a feasible solution for the MTP and a corresponding lower bound on the optimal value of the MTP.

A branch-and-bound algorithm (Kececioglu 1993), a branch-and-cut approach (Reinert et al. 1997), and an evolutionary algorithm (Koller and Raidl 2004) have been proposed to solve the MTP. Gotoh et al. (2006) pointed out that the first two approaches are able to solve only moderate-size instances exactly. The third approach is able to solve larger instances; however, as with any other metaheuristic, this approach does not provide any indication on how the objective value of the solution found compares to the optimal objective value.

The cutting plane algorithm designed by Reinert et al. (1997) essentially solves the IHSP formulation of the MTP. However, the main differences between their cutting plane approach and our implicit hitting set approach are the following:

1. Our method to solve IHSP works with integer feasible solutions, whereas their cutting plane approach solves...
a sequence of LP relaxations of the intermediate EHSPs. Moreover, our method finds an integer feasible solution every iteration, whereas the cutting plane approach rarely finds intermediate integer feasible solutions to the problem. Indeed, in order to overcome this limitation, Lenhof et al. used DIALIGN (Morgenstern et al. 1996) to solve the MTP to obtain an initial feasible solution for their branch-and-cut algorithm.

2. We explicitly use randomization to diversify the circuits found by the separation oracles (and use two separation oracles that find structurally different circuits).

3. Their separation oracle takes \( O(|A|(|E| + |A|)) \) time to find a cycle or assert that none exists, whereas a full FS-oracle circuit-generation routine (§6.1) takes only \( O(K(|E| + |A|)) \) time.

4. Our solution approach integrates exact and heuristic algorithms, using a heuristic to guide the search process and generate useful information that is later used by the exact algorithm.

6.1. Separation Oracles and Associated Circuit-Generation Routines

Once a problem is formulated as an IHSP, we only need to provide a separation oracle in order to solve it with the generic algorithm (§4). Here we briefly describe two separation oracles designed to solve the IHSP formulation of the MTP; the full details, including correctness proofs and running-time calculations, are found in the electronic companion (§EC.1) (available as supplemental material at http://dx.doi.org/10.1287/opre.1120.1139). These separation oracles are designed to find structurally different circuits—which is one of the diversification strategies used. Recall that (1) a separation oracle for this IHSP is an algorithm that solves the following separation problem: given an alignment graph \( G = (V, E, A) \), find a mixed cycle in \( G \) or report that none exists; and (2) a circuit-generation routine identifies a collection of sets in \( \Gamma \) that were not hit by \( H \).

6.1.1. Depth-First Search Oracle and Circuit-Generation Routine

The depth-first search separation oracle (DFS-oracle) is a simple modification of the well-known depth-first search (DFS) algorithm. In our implementation, it only takes one DFS call to check for the existence of a mixed cycle involving any arc associated with one given sequence. Therefore, DFS-oracle solves the separation problem in \( O(K(|E| + |A|)) \) time. And, the running time of a DFS-oracle circuit-generation routine is \( O(K(|E|(|A| + |E|))) \).

During a DFS-oracle circuit-generation routine, the starting sequence is randomly selected. This strategy promotes finding a diverse set of cycles/circuits. Otherwise, if DFS always started in the same sequence, then the cycles that include nodes from sequence \( k \) would be over-represented in the set of cycles found by DFS (which are added to the explicit list of circuits, \( \Gamma \)).

6.1.2. Fast-Search Oracle and Circuit-Generation Routine

Informally, the second separation oracle, which we call fast-search oracle (FS-oracle), attempts to find mixed cycles with at most \( K \) edges; as argued at the end of §4, it is important to detect circuits with small cardinality because, intuitively, if these circuits are not included in \( \Gamma \), then it is likely that the solution for the EHSP will not hit such circuits.

Although FS-oracle does not detect all mixed cycles, it is a very fast heuristic and finds different cycles than those found by DFS-oracle. Moreover, the FS-oracle is guaranteed to find mixed cycles at most \( K \) edges; as argued at the end of §4, it is important to detect circuits with small cardinality because, intuitively, if these circuits are not included in \( \Gamma \), then it is likely that the solution for the EHSP will not hit such circuits.

Finally, using both the DFS and FS oracles helps identify a diverse set ofmixed circuits; this considerably speeds up the generic algorithm. The combination of these oracles works as follows: First, perform a DFS-oracle circuit-generation routine; then restore all the edges deleted (i.e., added to \( H \)) during the DFS routine; next run a FS-oracle circuit-generation routine; finally, call a DFS-oracle circuit-generation routine to find the cycles missed by the FS-oracle.

6.2. Other Implementation Details

In the electronic companion (§EC.2) we briefly describe some relevant implementation details. The strategies therein described are simple but of critical importance for the IHS approach’s performance. For example, the diversification strategies therein described are so important that the running time difference between the using or not diversification strategies is of at least two orders of magnitude.
7. Performance Evaluation

This section presents our experimental setup, testing methodology, and computational experiments for testing the effectiveness of our approach to solve the MTP.

7.1. Computing Environment and Implementation

The experiments were performed on a personal computer (PC) with a 2.4 GHz Pentium 4 processor and 2 GB of RAM memory. The operating system of the machine was Red Hat 3.4.6 Linux.

The code was written in C++ using Concert Technology C++ version 2.5 interface for the mixed integer program (MIP) solver CPLEX (version 11.0) and the free version of the library of efficient datatypes and algorithms, LEDA 6.0 (Mehlhorn and Naher 1995). Concert and CPLEX are registered trademarks of ILOG Inc., and LEDA is a registered trademark of Algorithmic Solutions Software GmbH. Unless otherwise specified, we used the default options of LEDA and CPLEX.

Throughout this paper, the optimality gap (in %) is defined as $\frac{UB - LB}{LB}$, where $UB$ is the objective value of the best feasible solution found for the IHSP and $LB$ is the largest found lower bound on the optimal objective value of the IHSP. The vast majority of the LBs are obtained by optimally solving the EHSP defined by the explicit circuit list, $\Gamma$. The only two exceptions are (1) when the time limit is reached and we have not already optimally solved any EHSP, and (2) to calculate the optimality gap of the heuristic solution (recall that the heuristic solution is defined as the best feasible solution found before the first call to CPLEX); it is only in these two cases when we obtain the value $LB$ by solving the LP relaxation of the EHSP defined by $\Gamma$.

7.2. Experimental Results

We used three problem families in our experimental evaluation. The first family consisted of one real-data problem instance, the second family was produced by a random problem generator, and the third family was a set of related real-data instances.

For the experiments on real data (genomes of flies and worms), as explained in §5, we used fast statistical alignment (FSA) (Bradley et al. 2009) to compute the diagonals, their weights, and the anchors. We set the parameters in FSA such that each diagonal consisted of two ungapped, unique nucleotide subsequences of 10 or more nucleotides; each subsequence/anchor on a different genome and consisting of exactly the same nucleotides. Then, we represented each diagonal as an edge between the middle nucleotide of each anchor in the diagonal. Then, without loss of generality, we removed all edge-less nucleotides. Finally, we created the alignment graph $G = (V, E, A)$, as described in §6.

7.2.1. First Family: One Real Data Instance. The first family comprised one two-genome alignment problem. The sequences corresponded to an orthologous region of the genomes of fruit flies, in particular, Drosophila melanogaster (a widely used model organism in biology) and Drosophila yakuba. These regions contain 989,005 and 999,683 nucleotides, respectively.

The alignment graph for this instance consisted of two sequences with 9,337 and 9,863 nodes, respectively, and 9,982 edges. The first sequence had 8,732 nodes with one edge, 566 nodes with two edges, 38 nodes with three edges, and one node with four edges. The second sequence had 9,748 nodes with one edge, 111 nodes with two edges, and four nodes with three edges. Thus, the average (edge) degree of each node was close to one (we used this information in the design of the random instance generator used to create the second family of test problems).

We created two instances of the real-data problem: a weighted and an unweighted version (in the unweighted problem, the objective was to find the alignment consistent with the most diagonals/edges). We were able to optimally solve the weighted problem in 1.25 minutes (without the diversification strategies, it would take six hours); the alignment had a total weight of 0.298517. To optimally solve the unweighted problem took 2.6 minutes (without the diversification strategies, it would take over 10 hours) and the multisequence alignment was consistent with 5,755 out of the 9,982 diagonals/edges received as input; that is, about 60% of the edges in the input alignment graph were part of the alignment (we used this information in the design of the random instance generator used to create the second family of test problems).

7.2.2. Second Family: Randomly Generated Instances. Here, we describe the random instance generator used to test the performance of our algorithm. The generator was designed to create multisequence alignment graphs with similar properties to alignment graph in family 1.

The generator created an MTP problem instance from; (1) a seed to initialize the random number generator; (2) $K$: number of sequences; (3) $n$: length of the sequences, and (4) $p$: noise %.

The generator created a random instance in two stages. The first stage is to create an alignment graph without mixed cycles (corresponding to a valid multiple alignment). The second stage is to add random edges to the alignment graph (as a form of noise that hid the planted valid alignment).

The first stage of the generator worked as follows:

1. For each sequence, select $n$ columns uniformly at random out of $0.6n(K - 1)$ possible columns. In each selected column, place one character of the corresponding sequence.
2. Add an edge between the characters that were assigned to the same column.

Note that each character had, on expected value, 0.6 incident edges that were part of a valid multiple alignment. This number was chosen in accordance to the real-data instance previously described. The weight of each edge was an integer uniformly distributed in the interval $[1, 25]$.

Let $m_i$ be the number of edges generated during the first stage of the random instance generator. In the second
Figure 3. The running time appears to be polynomial with respect to the number of sequences and their length.

![Log-log plot of time vs. number of sequences](image)

![Log-log plot of time vs. sequence length](image)

stage, the generator added \([m_i p/100]\) edges to the alignment graph. These edges were chosen uniformly at random from the character pairs that did not have an edge between them. The weight of each edge was an integer uniformly distributed in the interval \([1, 25]\).

In Figures 3 and 4, we report the running time to optimality of our algorithm when solving problems created with the random generator. We solved problems for a varying number of sequences \((K \in [5, 10, 20])\), sequence lengths \((n \in [5,000, 10,000, 20,000])\), and noise levels \((p \in [5, 10, 20, 40, 80])\). For each combination, five random instances were generated. The average running time is reported in the figures. We set a one-hour time limit (3,600 seconds). The time limit was not checked during the circuit-generation routine; therefore, some runs went a little over the time limit. For those instances whose optimal solution was not found within one hour, we solved the EHSP’s LP relaxation to get the optimality gap. The time to solve this LP is not shown in Figures 3 and 4.

On one hand, Figure 3 shows that our algorithm’s running time appears to depend polynomially on the number of sequences and on the sequences’ lengths. On the other hand, Figure 4 illustrates that the running time is more sensitive to the noise levels. This suggests that the algorithm’s running time is more sensitive to the density of the alignment graph than to the number of nodes/edges. A possible explanation for this behavior is that the number of mixed cycles increases rapidly as the density of the graph increases.

For the same set of instances described above, Table 1 compares the performance of our algorithm when solving the problem to optimality and the performance when stopping the algorithm just before calling CPLEX (the heuristic solution). In particular, Table 1 reports the results of the experiments for a varying number of sequences \((K)\), sequence lengths \((n)\), and noise levels \((p)\). For each combination, five random instances were generated using the aforementioned random instance generator, and the results shown are the average performance in those five instances.
Table 1. Average and worst performance on five simulated instances for each set of parameters.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>To optimality</th>
<th>Heuristic solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (sec)</td>
<td>Gap (%)</td>
</tr>
<tr>
<td></td>
<td>Avg/max</td>
<td>Avg/max</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>K</th>
<th>n</th>
<th>p</th>
<th>M</th>
<th>Deg</th>
<th>Time</th>
<th>Gap</th>
<th>Cycle</th>
<th>Time</th>
<th>Gap</th>
<th>TGap</th>
</tr>
</thead>
<tbody>
<tr>
<td>5,000</td>
<td>5</td>
<td>8</td>
<td>1.28</td>
<td>1/1</td>
<td>2,411</td>
<td>1/1</td>
<td>0.07/0.27</td>
<td>0.06/0.27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10,000</td>
<td>5</td>
<td>16</td>
<td>1.28</td>
<td>3/3</td>
<td>4,733</td>
<td>2/2</td>
<td>0.06/0.17</td>
<td>0.02/0.11</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>17</td>
<td>1.30</td>
<td>6/6</td>
<td>9,772</td>
<td>4/5</td>
<td>0.07/0.15</td>
<td>0.01/0.06</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>18</td>
<td>1.34</td>
<td>14/19</td>
<td>20,702</td>
<td>9/9</td>
<td>0.13/0.22</td>
<td>0.01/0.03</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>21</td>
<td>1.43</td>
<td>69/90</td>
<td>47,832</td>
<td>33/34</td>
<td>0.16/0.20</td>
<td>0.03/0.06</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>27</td>
<td>1.60</td>
<td>3,603/3,607</td>
<td>4.2/7.6</td>
<td>912,032</td>
<td>dfn/dfn</td>
<td>dfn/dfn</td>
<td>dfn/dfn</td>
<td></td>
</tr>
<tr>
<td>5,000</td>
<td>5</td>
<td>16</td>
<td>1.31</td>
<td>4/5</td>
<td>6,405</td>
<td>3/4</td>
<td>0.12/0.24</td>
<td>0.04/0.19</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>16</td>
<td>1.33</td>
<td>9/10</td>
<td>13,519</td>
<td>6/7</td>
<td>0.19/0.25</td>
<td>0.06/0.16</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>18</td>
<td>1.37</td>
<td>29/40</td>
<td>30,340</td>
<td>14/15</td>
<td>0.21/0.27</td>
<td>0.06/0.16</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>21</td>
<td>1.45</td>
<td>125/147</td>
<td>82,739</td>
<td>72/87</td>
<td>0.12/0.20</td>
<td>0.07/0.10</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>27</td>
<td>1.62</td>
<td>3,611/3,615</td>
<td>22.2/26.9</td>
<td>1,809,634</td>
<td>dfn/dfn</td>
<td>dfn/dfn</td>
<td>dfn/dfn</td>
<td></td>
</tr>
<tr>
<td>5,000</td>
<td>5</td>
<td>13</td>
<td>1.32</td>
<td>14/16</td>
<td>12,782</td>
<td>9/10</td>
<td>0.09/0.14</td>
<td>0.00/0.02</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>13</td>
<td>1.34</td>
<td>33/48</td>
<td>27,024</td>
<td>20/21</td>
<td>0.12/0.21</td>
<td>0.02/0.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>36</td>
<td>1.38</td>
<td>108/137</td>
<td>60,265</td>
<td>50/55</td>
<td>0.16/0.22</td>
<td>0.03/0.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>42</td>
<td>1.46</td>
<td>572/623</td>
<td>163,981</td>
<td>261/304</td>
<td>0.09/0.15</td>
<td>0.05/0.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>54</td>
<td>1.62</td>
<td>3,623/3,633</td>
<td>46.4/47.4</td>
<td>1,708,714</td>
<td>dfn/dfn</td>
<td>dfn/dfn</td>
<td>dfn/dfn</td>
<td></td>
</tr>
<tr>
<td>5,000</td>
<td>5</td>
<td>13</td>
<td>1.33</td>
<td>21/22</td>
<td>17,356</td>
<td>15/17</td>
<td>0.11/0.16</td>
<td>0.02/0.05</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>33</td>
<td>1.35</td>
<td>61/77</td>
<td>38,317</td>
<td>31/35</td>
<td>0.16/0.20</td>
<td>0.02/0.05</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>36</td>
<td>1.39</td>
<td>217/249</td>
<td>96,368</td>
<td>104/135</td>
<td>0.13/0.17</td>
<td>0.04/0.06</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>42</td>
<td>1.47</td>
<td>3,605/3,612</td>
<td>0.7/1.1</td>
<td>448,386</td>
<td>3,110/dfn</td>
<td>0.11/dfn</td>
<td>0.11/dfn</td>
<td></td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>54</td>
<td>1.62</td>
<td>3,641/3,656</td>
<td>64.4/65.6</td>
<td>1,386,114</td>
<td>dfn/dfn</td>
<td>dfn/dfn</td>
<td>dfn/dfn</td>
<td></td>
</tr>
<tr>
<td>5,000</td>
<td>5</td>
<td>64</td>
<td>1.34</td>
<td>82/145</td>
<td>42,832</td>
<td>54/95</td>
<td>0.13/0.25</td>
<td>0.04/0.14</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>66</td>
<td>1.35</td>
<td>182/324</td>
<td>76,520</td>
<td>98/106</td>
<td>0.13/0.18</td>
<td>0.01/0.02</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>72</td>
<td>1.39</td>
<td>957/1171</td>
<td>191,340</td>
<td>396/472</td>
<td>0.11/0.17</td>
<td>0.03/0.05</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>84</td>
<td>1.47</td>
<td>3,626/3,635</td>
<td>4.5/5.4</td>
<td>743,254</td>
<td>dfn/dfn</td>
<td>dfn/dfn</td>
<td>dfn/dfn</td>
<td></td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>108</td>
<td>1.63</td>
<td>3,747/3,809</td>
<td>94.7/106.1</td>
<td>934,085</td>
<td>dfn/dfn</td>
<td>dfn/dfn</td>
<td>dfn/dfn</td>
<td></td>
</tr>
<tr>
<td>5,000</td>
<td>5</td>
<td>126</td>
<td>1.33</td>
<td>250/310</td>
<td>68,857</td>
<td>147/163</td>
<td>0.10/0.11</td>
<td>0.01/0.03</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>132</td>
<td>1.35</td>
<td>943/1,273</td>
<td>152,468</td>
<td>432/555</td>
<td>0.16/0.18</td>
<td>0.02/0.04</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>144</td>
<td>1.39</td>
<td>3,260/3,872</td>
<td>378,911</td>
<td>1,676/1,817</td>
<td>0.10/0.11</td>
<td>0.01/0.02</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>168</td>
<td>1.47</td>
<td>3,706/3,745</td>
<td>25.7/26.9</td>
<td>1,139,580</td>
<td>dfn/dfn</td>
<td>dfn/dfn</td>
<td>dfn/dfn</td>
<td></td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>216</td>
<td>1.63</td>
<td>4,010/4,035</td>
<td>156.1/162.4</td>
<td>1,310,769</td>
<td>dfn/dfn</td>
<td>dfn/dfn</td>
<td>dfn/dfn</td>
<td></td>
</tr>
</tbody>
</table>

For the solution times and the optimality gaps Table 1 also reports the maximum value, which (a) gives an idea of the (small) variability on these performance measures and (b) reports the worst performance of the algorithms in each set of five instances. The columns of Table 1 give number of sequences (K); sequence lengths (n); noise level (p); average number of edges (rounded to the nearest thousand) in the alignment graphs (M); average edge-degree of nodes in the alignment graphs (deg); time to optimality (rounded to the nearest second) (Time); upper bound on the optimality gap at termination (Gap); average number of mixed cycles found throughout the execution of the algorithm (Cycles); time (rounded to the nearest second) before the first call to CPLEX, that is, time to obtain the heuristic solution (Time); an upper bound on the heuristic solution’s optimality gap (Gap) (obtained by solving the LP relaxation of the EHSP that contained all the mixed cycles found before the first call to CPLEX. The time to solve this LP relaxation is not included in Table 1); and true optimality gap of the heuristic solution (TGap) (obtained by
over 95% of the problems were optimally solved in less than 10 seconds. The rows with entries “dnf” indicate that CPLEX was not called within the one-hour time limit. Note that the time limit was, in rare cases, not respected by CPLEX; so, in some instances, the running time exceeded the time limit.

We make the following observations regarding the results shown in Table 1:

1. Whenever the heuristic solution was obtained within the one-hour time limit, the upper bound on its optimality gap (Gap) was at most 0.36% and was generally below 0.2%. Similarly, its true optimality gap (TGap) was at most 0.36% and was generally below 0.05%.

2. The quality of the upper bound on the heuristic solution’s optimality gap (Gap) is very good; i.e., it is very close to the true optimality gap (Gap). This suggests a good effectiveness of the strategy of using the greedy algorithm to help in finding a set of “important circuits” (see §4), and thus making the EHSP a strong relaxation of the IHSPP.

3. The size of the problems that our algorithm was not able to optimally solve within one hour was huge. To appreciate the size of these problems, consider the following: the linear program solved to obtain the optimality gap at termination had at least 27,000 variables and 631,000 constraints (the biggest of such LPs had more than 216,000 variables and 1.5 million constraints!). Indeed CPLEX took more than five hours to solve 23 of those LPs and took over three days to solve the five biggest such LPs. (These LPs have a special structure and can be solved very efficiently, but we had no access to such algorithms and used CPLEX’s general purpose LP solver. This is why throughout this section all running times reported do not include the time to solve the LP relaxation when this LP relaxation was solved only for the purpose of obtaining a LB.) Given this, we believe that it is quite remarkable that within one hour, our algorithm was able to obtain a feasible integer solution whose optimality gap is at most 163% and, in most cases, below 50%.

7.2.3. Third Family—Multiple Alignments of Worm Genomes. The third family of problems consisted of 4,096 multigenome alignment problems. Each problem corresponded to a multiple alignment of an orthologous region across five (or fewer, if no orthologous region was present in a genome) worm genomes. To compute the input alignment graph from these genome alignment problems, we used FSA using the same parameters as those used for the first problem family. The alignment graphs obtained had up to five sequences; the sequences had lengths (counting only nodes with one or more adjacent edges) of up to 6,756 nodes, and the alignment graphs had up to 17,663 edges. Note that these real-life problems had sizes comparable with the smallest of the (randomly generated) problems in the previous section. To report the results obtained in these instances, we classified the problems into two groups. Group A with 4,066 problems that were solved to optimality within one hour. Group B with the remaining 30 problems.

Analysis of Group A

Table 2 summarizes the features of the problems in group A and the performance of the IHSPP approach when solving them. Each row in this table describes the problems that were optimally solved in a time greater than or equal to Min and less than Max. Each row gives the number of problems solved (Num Prob); the number of sequences (K); the length of the longest sequence (MaxL); the number of edges (M); and the average edge-degree of the nodes (Deg). The last four columns are given in the following format: (Min; Median; Max).

Next, we focus our attention on the solution obtained before the first call to CPLEX—which we refer to as the heuristic solution. Table 3 gives a summary of the performance of the heuristic solution. In particular, Table 3(a) gives a summary of the times (in seconds) to obtain the heuristic solution; Table 3(b) gives a summary of the heuristic solution’s true optimality gap, TGap, which was obtained by comparing the objective value of the heuristic solution with the optimal objective value; and, Table 3(c) gives a summary of the upper bound on the heuristic solution’s optimality gap, Gap, obtained by solving the LP relaxation of the EHSP that contained all the mixed cycles found before the first call to CPLEX (i.e., obtained quickly and relatively effortlessly). In this respect, we make the following observations:

1. The heuristic solution was obtained extremely quickly. In particular, the heuristic solution was obtained in less than one second for 98% of the problems. Other relevant statistics for the time to obtain the heuristic solution are
Table 3. Performance of the heuristic solution on group A problems.

(a) Time (seconds) to get the heuristic solution.

<table>
<thead>
<tr>
<th>Time</th>
<th>No. of problems</th>
<th>Cumulative percentage (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>Max</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0.01</td>
<td>2,544</td>
</tr>
<tr>
<td>0.01</td>
<td>0.1</td>
<td>1,011</td>
</tr>
<tr>
<td>0.1</td>
<td>1</td>
<td>435</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>68</td>
</tr>
<tr>
<td>10</td>
<td>60</td>
<td>5</td>
</tr>
<tr>
<td>60</td>
<td>240</td>
<td>3</td>
</tr>
</tbody>
</table>

(b) Optimality gap of the heuristic solution, obtained by comparing the objective value of the heuristic solution with the optimal value.

<table>
<thead>
<tr>
<th>TGap (%)</th>
<th>No. of problems</th>
<th>Cumulative percentage (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>Max</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>3,121</td>
</tr>
<tr>
<td>0.1</td>
<td>1</td>
<td>65</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>360</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>54</td>
</tr>
<tr>
<td>10</td>
<td>25</td>
<td>13</td>
</tr>
</tbody>
</table>

(c) Upper bound on the heuristic solution’s optimality gap, obtained by solving the LP relaxation of the EHSP that contained all the mixed cycles found before the first call to CPLEX.

<table>
<thead>
<tr>
<th>Gap (%)</th>
<th>No. of problems</th>
<th>Cumulative percentage (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>Max</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2,578</td>
</tr>
<tr>
<td>0.1</td>
<td>1</td>
<td>37</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>703</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>204</td>
</tr>
<tr>
<td>10</td>
<td>25</td>
<td>48</td>
</tr>
</tbody>
</table>

average = 0.2 seconds, standard deviation = 3.96 seconds, and maximum = 3.62 minutes.

2. The heuristic solution attained an objective value that was very close to the optimal solution value (see Table 3(b)). In particular, the heuristic solution was optimal for 3,121 problems (that is, 76.8% of all problems). Other relevant statistics for the TGap of the heuristic solution are average = 0.4%, standard deviation = 1.3%, and maximum = 23%.

3. The quality of the upper bound on the heuristic solution’s optimality gap, obtained by solving the LP relaxation of the EHSP that contained all the mixed cycles found before the first call to CPLEX, is very good (compare Table 3(c) with Table 3(b)). In particular, the relevant statistics for gap, which are average = 1.0%, standard deviation = 2.2%, and maximum = 25%, are very close to those of the true optimality gap, TGap. These results suggest that the greedy algorithm does help to find a set of “important circuits” (see §4), and make the EHSP a strong relaxation of the IHSP.

Therefore, we believe that the heuristic solution is good enough for most practical applications.

Analysis of Group B

In Table 4, we report on the experiments on the problems of group B, the only 30 problems (out of the 4,096 problems) that were not optimally solved within one hour. We observe that, in general, solving the EHSP with CPLEX dominates the solution time (also, in rare instances CPLEX did not respect the time limit). Although these problems were not optimally solved within the time limit, all but one of the feasible solutions found had optimality gaps smaller than 7%. The heuristic solution, obtained in less than four minutes in all but four of the problems, has optimality gaps comparable to those of the best solution found during the one-hour execution of the algorithm.

In only one of the problems, the heuristic solution could not be found within the time limit (however, other previous feasible solutions were found—the best one with only a 17% optimality gap). Note that, when solving this problem, at the time limit, our algorithm had found more than one million mixed cycles. This number of cycles is comparable to the number of cycles found in the most difficult instances of the second family of problems (those that we randomly generated).

8. Concluding Remarks and Future Work

This paper showed that several combinatorial optimization problems can be formulated in a relatively simple way as implicit hitting set problems and proposed a generic algorithmic framework to solve these problems. Our solution approach integrates exact and heuristic algorithms and exploits the capabilities of commercially available linear and integer programming solvers. More specifically, a central idea of our approach is to use an heuristic algorithm for the explicit hitting set problem, not only to find feasible solutions for the EHSP (which can be easily augmented to get feasible solutions for the implicit hitting set problem) but also to help in building a small set of important circuits that provide a strong relaxation of the IHSP (which is used to obtain bounds). Thus, in this sense, the heuristic algorithm is used as a mechanism to guide the search process and generate useful information that is later used by the exact algorithm. Indeed, an important characteristic of this methodology is that it generates, at each iteration, both a feasible solution for the IHSP and a bound (ultimately yielding an optimal solution and a bound certifying its optimality). Finally, a key component in the framework is the use of randomization to diversify the circuits generated by the separation oracle. This diversification greatly improves the strength of the IHSP’s relaxations as well as the quality of the solutions found by the EHSP’s heuristic.

We also showed that by reformulating and solving the multigenome alignment problem, and equivalently any multisequence alignment problem, as an IHSP, we are able...
to solve practical-size instances. Our extensive experimental results support the usefulness of the implicit hitting set approach to solve the MGAP, and we plan to investigate whether this approach is also practical when applied to other hard combinatorial optimization problems. Here are a few final remarks:

1. The algorithm performed extremely well on the real-data problems (optimally solving most of the instances in less than one minute). Indeed, for practical multisequence alignment problems, our algorithm was able to optimally solve the vast majority of the problems within one hour. Moreover, because for practical applications obtaining the optimal alignment might not be required, then the heuristic solution, which was found in all but four of the real problems in less than five minutes, might be good enough.

2. The time to find the heuristic solution is insignificant compared to the time to optimally solve the problems. Moreover, for most instances, the optimality gap of the heuristic solution was less than 5%, and for all of the instances, the optimality gap was less than 23%.

3. The running time to optimally solve the problems is dominated by the time required to optimally solve the EHSPs. Thus, using a specialized algorithm to solve the EHSP, instead of using a general purpose IP solver, might help to significantly reduce the running times.

4. The algorithm’s running time is more sensitive to the density of the alignment graph than to the number of nodes/edges. A possible explanation for this behavior is that the number of mixed cycles increases rapidly as the density of the graph increases. In contrast, the DP algorithm’s running time is exponential in the number of nodes, which makes DP impractical for solving practical-size instances.

A promising research direction, which was not explored in this paper, is the possible (duality-based) connection between the concept of “important circuits” (i.e., the important constraints that the greedy algorithm aims to find) with the concept of “backdoors.” Backdoors are small sets of variables such that, when their values are determined, the problem is significantly simplified and easily solved. Williams et al. (2003) introduced the concept of backdoors to explain why it is possible to obtain solutions to practical-sized instances of NP-hard optimization problems even though worst-case complexity analysis suggests that these problems are intractable. It would be interesting to analyze circuits/constraints using methods analogous to those used in Dilkina et al. (2007, 2009) to analyze backdoors/variables.
Supplemental Material

Supplemental material to this paper is available at http://dx.doi.org/10.1287/opre.1120.1139.

Acknowledgments

The authors are grateful to Lior Pachter and his postdoctoral fellow Robert K. Bradley for sharing the real-data instances and their FSA software package. The authors thank two anonymous referees for their valuable comments and suggestions that helped improve this paper.

References


Erick Moreno-Centeno is an assistant professor at Texas A&M University in the Department of Industrial and Systems Engineering. In 2012 he was named a Montague–Center for Teaching Excellence Scholar (Texas A&M). His research interests are in the general area of mathematical programming, particularly the design and analysis of combinatorial optimization models and algorithms to solve complex problems. He is working on
developing optimization methods that are free of round-off errors and algorithms for transmission switching in the energy network.

Richard M. Karp is University Professor at the University of California, Berkeley. His current activities center on algorithmic methods in genomics and computer networking. Honors and awards include the U.S. National Medal of Science, Turing Award, Fulkerson Prize, Kyoto Prize (Japan), Harvey Prize (Technion), Centennial Medal (Harvard), Lanchester Prize, von Neumann Theory Prize, von Neumann Lectureship, Distinguished Teaching Award (Berkeley), Faculty Research Lecturer (Berkeley), Miller Research Professor (Berkeley), Babbage Prize, and eight honorary degrees. He is a member of the U.S. National Academies of Sciences and Engineering, the American Philosophical Society, and the French Academy of Sciences. He is a Fellow of the American Academy of Arts and Sciences, the American Association for the Advancement of Science, the Association for Computing Machinery, and the Institute for Operations Research and the Management Sciences.