Algorithms for Comparing Pedigree Graphs

Bonnie Kirkpatrick[1], Yakir Reshef[2], Hilary Finucane[3], Haitao Jiang[4], Binhai Zhu[5], Richard M. Karp[6]

[1] Electrical Engineering and Computer Sciences, University of California, Berkeley, and International Computer Science Institute, Berkeley, bbkirk@eecs.berkeley.edu.
[2] Harvard College, yreshef@post.harvard.edu.
[3] Weizmann Institute of Science, Rehovot, Israel, hilary.finucane@weizmann.ac.il.
[4] School of Computer Science and Technology, Shandong University, China. htjiang@cs.montana.edu.
[5] Department of Computer Science, Montana State University, Bozeman, MT 59717, USA bhz@cs.montana.edu.
[6] Electrical Engineering and Computer Sciences, University of California, Berkeley, and International Computer Science Institute, Berkeley, karp@cs.berkeley.edu.

Abstract. Pedigree graphs, which represent family relationships, are often constructed by collecting data from genealogical records to determine which pairs of people are parent and child. This process is expensive, and small mistakes in data collection–for example, one missing parent-child relationship–can cause large differences in the pedigree graphs created. In this paper, we introduce a simple pedigree definition based on a different type of data which is potentially easier to collect. This alternative characterization of a pedigree that describes a pedigree as a list of the descendants of each individual, rather than a list of parent-child relationships. We then introduce an algorithm that generates the pedigree graph from this list of descendants.
We also consider the problem of comparing two pedigree graphs, which could be useful to evaluate the differences between pedigrees constructed via different methods. Specifically, this could be useful to evaluate pedigree reconstruction methods. We define the edit distance between two pedigrees and prove that calculating this edit distance is APX-hard. Our new characterization of a pedigree allows us to introduce a fast heuristic for the edit distance between pedigrees. In addition we introduce several exact algorithms for calculating distances in restricted and general cases.

1 Introduction

Most pedigree construction algorithms are manual in nature and use genealogical information that can include birth and death dates. Unfortunately, there can be multiple sources of information with contradictory information. Pedigree errors are a common source of error for pedigree calculations [19,15,1]. It has been estimated that between 2-10% of people do not know who their biological father is [9,18]. In addition, in order to evaluate the accuracy of pedigrees, it is necessary to be able to compare putative pedigrees. For these reasons, it is important to have algorithmic methods for generating correct pedigrees and for detecting the differences between pedigrees.

Pedigrees are objects of interest in computer science due to their close connection with machine learning methods [12,7]. Most calculations on pedigree graphs are hard [16,13,10]. Notable attempts have been made to improve the speed of pedigree calculations [2,8,14].

For diploid individuals, a pedigree is a directed graph where individuals I are nodes, every node has a gender (male or female), and there is a directed edge $i \rightarrow j$ if and only if i is the parent of j (see Figure 1). For each chromosome, this edge represents the transmission from parent i to offspring j of a single (possibly recombinant) copy of that chromosome. The accuracy of each edge is important, since the presence or absence of a single edge will determine whether many pairs of individuals are related to each other.

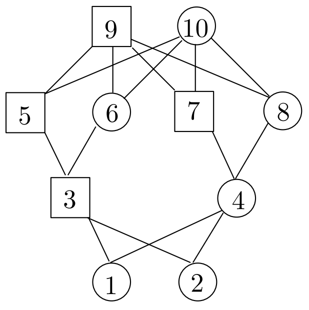


Fig. 1. An Example Pedigree. Each node is an individual, with boxes representing males and circles representing females. The two individuals adjacent to and above each node are the parents of the child below. Time proceeds in the downward direction, implicitly directing this graph. It is standard to discard the arrows on the edges and to use the top-to-bottom ordering of nodes in the graph to convey the directionality. Individuals without parents are called *founders*, and they are usually assumed to be unrelated.

One early attempt to construct pedigrees from genetic data attempts to chose pedigree graphs that maximized the likelihood of the observed data [22]. That approach to pedigree construction is a typical example of structured machine learning, where the graphical model of interest is the pedigree model, and it works only on very small families. More recent theoretical work takes a combinatorial and probabilistic perspective [21,20]. This paper focuses on combinatorial methods of comparing pedigree graphs. In Section 2, we introduce a new formulation of pedigrees that is equivalent to the parent-child formulation, but more robust. In Sections 3 and 4, we define pedigree distance and prove that it is APX-hard to calculate. In Section 5, we introduce some heuristics for calculating pedigree distance, based on our new formulation from Section 2, and in Section 6, we evaluate these new heuristics.

2 Descendant-Splits Formulation

An alternative formulation of a pedigree would allow the hypothesis that a set of individuals is descended from a common ancestor (called a **descendant split**), without specifying the number of generations between each of the individuals and their common ancestor(s). The presence or absence of a single descendant hypothesis may only change the closeness of the relationship between a pair of individuals (perhaps from cousins to 2nd-cousins), rather than removing the relationship entirely. This is in contrast to the traditional formulation of a pedigree as a collection of parent-offspring edges, where a missing edge entirely changes the nature of many relationships. This section is a revised version of a similar section in [11].

Definition 1. *Let I be the set of individuals in a pedigree, and let X be the set of labeled individuals. The* **descendant split** *(or* **d-split***) of an individual $i \in I$ is defined as a subset of X:*

$$D_i(X) = \{j \in X | \ \ j \textit{ is descended from } i\}$$

where an individual with no child is considered a descendant of itself. For a particular set of interest, X, refer to the set of d-splits as $\mathcal{D}_X = \{D_i(X) \mid i \in I\}$.

Each d-split specifies some relationship between all the individuals in D_i. For the example given in Figure 1, when all individuals are labeled the list of d-splits, $\mathcal{D}_I$, are: $D_1 = \{1\}$, $D_2 = \{2\}$, $D_3 = \{1,2\}$, $D_4 = \{1,2\}$, $D_5 = \{1,2,3\}$, $D_6 = \{1,2,3\}$, $D_7 = \{1,2,4\}$, $D_8 = \{1,2,4\}$, $D_9 = \{1,2,3,4,5,6,7,8\}$, and $D_{10} = \{1,2,3,4,5,6,7,8\}$. Similarly, if we restricted our attention to $X = \{1,2\}$, then $\mathcal{D}_X$ would contain $\{1\}$, $\{2\}$, and $\{1,2\}$.

The term "descendant split" is deliberately chosen to evoke the image of a split in a perfect phylogeny and to pay homage to phylogenetic reconstruction methods [17] which are a source of inspiration for this work. Just as a set of splits determines a class of perfect phylogeny trees that are compatible with the splits, a set of descendant splits specifies a class of pedigree graphs that are compatible with the splits. We will formalize this idea with several lemmas.

Lemma 1. *Let $\mathcal{D}_I = \{D_i(I) \mid i \in I\}$ be the d-splits defined by a pedigree P. This set can be used to construct a unique pedigree which is identical to pedigree P.*

Lemma 2. *Let $\mathcal{D}_X = \{D_i(X) \mid i \in I\}$ be the d-splits defined by a pedigree P and a set X. This set of d-splits specifies a class of pedigrees compatible with the splits. Pedigree P is one of the pedigrees compatible with the d-splits.*

First consider the d-splits in $\mathcal{D}_I$. Any singleton d-split, $D_i \in \mathcal{D}_I$ with $|D_i| = 1$, represents an individual that is childless. Therefore these d-splits represent individuals in the most recent generation of the pedigree. Now, find some ancestor i_1 and examine any directed path descending from that person, for example, $i_1 \to i_2 \to ... \to i_{k-1} \to i_k$, where the arrow indicates a directed parent-offspring relationship. We see that the d-splits along that descendant path are ordered $D_{i_1} \supset D_{i_2} \supset ... \supset D_{i_k}$. Indeed, the cardinality of the d-split sets D_{i_j} strictly decreases as we consider individuals lower in the path. These two ideas result in a simple algorithm for constructing the pedigree.

Example. If we take the d-splits $\mathcal{D}_I$ from the example in Figure 1, we can apply the algorithm to construct the pedigree. Figure 2 shows the d-splits using a Venn diagram. Each step in the algorithm constructs a set of parent-child edges. This example also illustrates the ambiguity of the d-splits when individuals in the pedigree are unlabeled. As noted before, if only individuals $\{1,2\}$ are labeled, then many of the ancestral d-splits are identical.

Proof. of Lemma 1
Since we have a d-split for every individual, the algorithm will either assign founder status or parents to every individual. Now, if we look at a single step in the algorithm, each individual will be assigned the parents. Due to the strictly increasing cardinality of d-splits as one moves up the pedigree (notice that this is due to an individual being contained in their own d-split), these parents are represented by the unique two smallest descendant splits. □

Algorithm 1 GraphConstruction()

```
1: $Heap := (D_{i_0}, ..., D_{i_k})$ where $|D_{i_0}| \leq |D_{i_1}| \leq ... \leq |D_{i_k}|$
2: Create pedigree $P$ with nodes $\{i_0, i_1, ..., i_k\}$.
3: while $Heap \neq \emptyset$ do
4:     $D_{i_j} := pop(Heap)$
5:     Look for the smallest $D_{i_f}$ and $D_{i_m}$, respectively the female and male splits, such that $D_{i_j} \subseteq D_{i_f}$ and $D_{i_j} \subseteq D_{i_m}$
6:     if $D_{i_f}$ and $D_{i_m}$ are found then
7:         add to $P$ the edges $i_m \rightarrow i_j$ and $i_f \rightarrow i_j$
8:     else
9:         $i_j$ is a founder and has no parents.
10:    end if
11: end while
12: return $P$
```

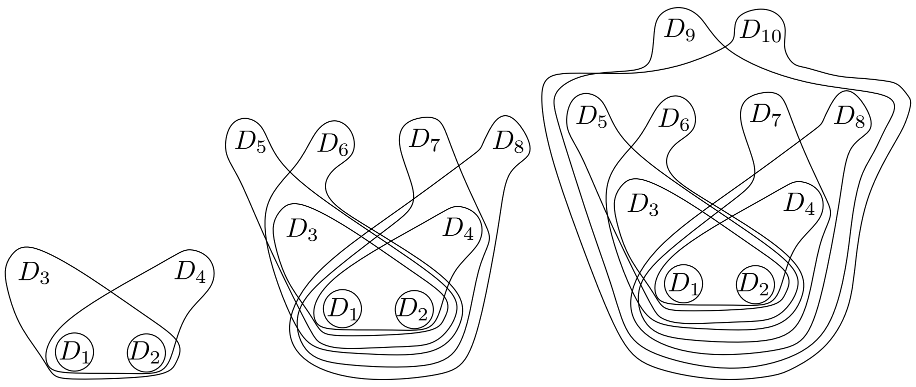


Fig. 2. Constructing a Pedigree from the Full D-Splits. Given the d-splits in $\mathcal{D}_I$ for the set I of all the individuals in the pedigree in Fig. 1, we can use the construction algorithm to recover the pedigree. These are the Venn diagrams of the construction at three different steps in the algorithm. Each panel shows a full generation of algorithm iterations, and right-most panel shows the complete construction. Each d-split is drawn as a set containing the related individuals. Each set in the diagram is labeled with the name of its d-split, and the names of the d-splits are arbitrary as long as they are distinct.

Interestingly, we can use the same algorithm when we consider d-splits on a subset of the individuals. As long as we have a separate d-split for each person in the pedigree, we will know the number of generations in each lineage. The main difference is that each lineage has *non-decreasing* cardinality of d-splits as we move backwards in time. The missing information, now, is in not knowing which d-split was generated by the parent versus a more distant ancestor. For the example we gave above, if $X = \{1, 2\}$, then $D_9(X)$ and $D_7(X)$ are indistinguishable.

Proof. of Lemma 2
Again, since we have a d-split for every individual, the algorithm will either assign founder status or parents to every individual. Now, if we look at a single step in the algorithm, each individual will be assigned some parents, due to the non-decreasing cardinality of d-splits as we consider d-splits for individuals in older generations. However, the construction will be different for re-orderings of the d-splits. This means that we cannot resolve the correct labels for individuals $I \setminus X$ in the interior of the pedigree. Even though we cannot resolve the interior, if we run the algorithm once for each possible ordering of the d-splits, we obtain a collection of pedigrees that are consistent with the d-splits. The actual pedigree will be contained in this collection. $\square$

Notice that this idea of descendant splits is a general property of any directed acyclic graph with fixed in-degree. This idea provides a core description of the pedigree reconstruction algorithm that Thatte and Steel proposed [21]. They focused on the scenario where the individuals of interest, referred to in this paper as X, were leaf individuals in some unknown pedigree. They proposed an artificial generative model of inheritance in which every individual contributed some unique alleles with high probability. These unique alleles then became markers for each d-split, allowed them to piece together the lineages of multiple leaf individuals, and allowed them to analyze the amount of genetic material required for successful prediction of the pedigree structure.

3 Definition of Pedigree Edit Distance

To evaluate reconstruction methods, we want to be able to determine how good a predicted pedigree is by comparing it to a correct pedigree graph. Informally, given two arbitrary pedigree graphs, $P = (I(P), E(P))$ and $Q = (I(Q), E(Q))$, we want to find the minimum number of parent-child edge additions/deletions required to convert Q into P. We call this the *edge edit distance* between P and Q. Notice that it is not necessary that $|I(P)| = |I(Q)|$ because addition/deletion of edge-less nodes is free.

Let us define edge edit distance more formally. For an individual i, let $s(i) \in \{m, f\}$ indicate male and female gender, respectively. Define a *matching of pedigrees* between P and Q to be a subset M of $I(P) \times I(Q)$ such that $(i, j) \in M$ if and only if $s(i) = s(j)$, and for each i, there is at most one j such that $(i, j) \in M$. Given such a matching M and some $i \in I(P)$, we define $M(i) = j$ if $(i, j) \in M$ and $M(i) = \lambda$ otherwise, where λ is a special symbol reserved for nodes that are not matched. We will abuse notation and use $M(j)$ to denote the analogous function for individuals in $I(Q)$ as well. With this notation, we define the *match distance* incurred by M as

$$d(M) = d_{P,Q}(M) + d_{Q,P}(M),$$

where $d_{G,G'}(M) = |\{(i,j) \in E(G) | (M(i), M(j)) \notin E(G')\}|$. Here, $d_{G,G'}(M)$ is the number of parent-child edges in the pedigree G that do not correspond under the matching M to parent-child edges in G'.

Now, we can define a distance between pedigrees. Let the *edit distance* between pedigrees P and Q be defined as the minimum matching distance:

$$D_{P,Q} = \min_M d(M).$$

Notice that the dual optimization problem is that of maximizing the number of matched edges. When convenient, we will convenient, we occasionally deal with the dual.

Assume that we have a set of labeled individuals X, as before, and that the set of labeled individuals is the same in both pedigrees P and Q. Recall that this is a set of distinguishable individuals; for instance we may have data for them. In this case then the matching M will have several fixed matches, where for $i \in X$, $(i,i) \in M$. This forces the matching to be consistent with respect to the labeled individuals.

The problem of finding the pedigree distance is a specific case of the well-studied problem of inexact graph matching [5]. Inexact graph matching is not only NP-hard in general, but MAX SNP-hard even when restricted to trees [23], and there is a extensive literature on heuristics and algorithms for inexact graph matching. However, the hardness results do not apply to pedigree distance–perhaps the hard cases of inexact graph matching are non-pedigrees–and the algorithms do not take advantage of the structure of pedigrees. In Section 4, we show that computing pedigree distance is APX-hard, and in Section 5, we present several heuristics and algorithms for calculating pedigree distance in general and in specific cases.

4 Hardness of Computing Pedigree Edit Distance

In this section, we show that calculating the edge edit distance between two pedigrees is APX-hard for monogamous out-bred pedigrees (i.e. tree-like pedigrees), by reducing from Minimum Common Integer Partition. For clarity, when applied to trees, we call the edge edit distance *cut-and-paste* distance (cut/paste for short). A cut/paste operation involves deleting (cutting) an edge $x \to y$ and adding (pasting) a new edge $z \to y$.

Given two minimization problems Y and Z, an *L-reduction* from Y to Z is a pair of polynomial-time functions f, g and a pair of positive constants α, β meeting the following conditions.

(1) For every instance y of Y, $f(y)$ is an instance of Z with

$$opt(f(y)) \leq \alpha \cdot opt(y),$$

(2) For a feasible solution z to $f(y)$, $g(z)$ is a feasible solution to y such that

$$|opt(y) - val(g(z))| \leq \beta \cdot |opt(f(y)) - val(z)|.$$

Note that $opt(y)$ is the value of the optimal solution to an instance y, while $val(z)$ denotes the value of solution z. With the above two properties, it is easily seen that the following inequality on the relative errors of approximation holds:

$$\frac{|opt(y) - val(g(z))|}{|opt(y)|} \leq \alpha\beta \cdot \frac{|opt(f(y)) - val(z)|}{|opt(f(y))|}.$$

When the above conditions are satisfied, if it is NP-hard to approximate Y with a factor of $1 + \alpha\beta\epsilon$, then it is NP-hard to approximate Z with a factor of $1 + \epsilon$.

We reduce MCIP (Minimum Common Integer Partition) to MCPDT (Minimum Cut/Paste Distance between Trees) with an L-reduction.

A partition of an integer n is a multiset $\{n_1, n_2, ..., n_t\}$ such that $\sum_{1 \leq i \leq t} n_i = n$. For example, when $n = 8$, $\{3, 2, 2, 1\}$ is a partition of n.

A partition of a multiset $S = \{x_1, x_2, ..., x_p\}$ is a multiset union of all the partitions $P(x_i)$, i.e., $\cup_i P(x_i)$. A multiset X is a common partition of two multisets $S_1 = \{x_1, x_2, ..., x_p\}, S_2 = \{y_1, y_2, ..., y_q\}$ if there are partitions P, Q with $\cup_i P(x_i) = \cup_j Q(y_j) = X$. For example, given $S_1 = \{8, 5\}, S_2 = \{9, 4\}$, $X = \{5, 3, 2, 2, 1\}$ is a common partition of S_1, S_2.

MCIP (Minimum Common Integer Partition)

Instance: Two multisets of integers S_1, S_2, integer k.

Question: Do S_1, S_2 admit a common partition of size k?

It was shown in [4] that MCIP is APX-hard.

MCPDT (Minimum Cut/Paste Distance between Trees)

Instance: Two directed rooted unlabeled trees T_1, T_2, integer k.

Question: Can T_1 be converted into T_2 using k cut/paste operations?

Note that the question in MCPDT is also equivalent to: Can T_2 be converted into T_1 using k cut/paste operations? Or, Can T_1, T_2 be converted into a tree T using a total of k cut/paste operations? Or, Can T_1, T_2 be cut into a common forest F each using k edge cuts?

The reduction is simple. Given $S_1 = \{x_1, x_2, ..., x_p\}, S_2 = \{y_1, y_2, ..., y_q\}$, we construct two trees T_1, T_2 with roots r_1, r_2 such that the descendants of r_1 (resp. r_2) is composed of p (resp. q) paths, each is of size x_i (resp. y_j), for $1 \leq i \leq p$ (resp. $1 \leq j \leq q$). We need to cut T_1, T_2 into a common forest, in which each tree, except the ones containing r_1, r_2, is a path.

Let $opt(MCIP), A(MCIP)$ be the values of the optimal and approximate solution of MCIP. Let $\min = \min\{p, q\}$. Then the value of the optimal solution for MCPDT is $opt = opt(MCIP) - \min$. (In other words, we can conclude that S_1, S_2 has a solution of size k if and only if T_1, T_2 each can be cut into a common forest with $(k - \min)$ cuts.) Moreover, $A(MCPDT) = A(MCIP) - \min$ is the value of a feasible solution of MCPDT. Clearly we have

(1) $opt \leq \alpha \cdot opt(MCIP)$, by setting $\alpha = 1$.

(2) $|opt(MCIP) - A(MCIP)| \leq \beta \cdot |opt - A(MCPDT)|$, by setting $\beta = 1$.

To see (2), $|opt - A(MCPDT)| = |opt - (A(MCIP) - \min)| = |opt(MCIP) - A(MCIP)| \geq |opt(MCIP) - A(MCIP)|$.

Therefore, this reduction is an L-reduction. As MCIP is APX-hard, MCPDT is also APX-hard. This implies that unless P=NP, there is no way to obtain a PTAS for MCPDT. Since MCPDT reduces directly to the pedigree edit distance on monogamous outbred pedigrees, there is unlikely to be a PTAS for computing the edit distance. This proof works for labeled pedigrees provided that the labels are the same in the two pedigrees.

5 Algorithms for Computing Pedigree Edit Distance

We showed in the previous section that no polynomial time algorithm exists to determine the pedigree distance unless $P = NP$. Indeed, even for monogamous pedigrees, there is probably no PTAS. In this section, we present a polynomial time randomized heuristic for calculating pedigree distance on general labeled pedigrees, a dynamic programming algorithm which provides an optimal solution for a restricted set of pedigrees in time which depends on distance between pedigrees, and we summarize a number of other algorithms for general and special cases.

Let a *regular* pedigree be one where every individual is monogamous and only individuals of the same generation mate with each other. Consider the case of two regular pedigrees both having the same set of labeled individuals, X.

Two-Generation Polynomial-Time Algorithm. When P and Q are both two-generation pedigrees where the labeled individuals X entirely determine the matching of the most recent generation, we have a polynomial-time algorithm. We construct two maximum-weight bipartite matching instances, one for each gender, whose solutions gives us the best matching. Notice that in doing this, we deal with the dual problem to the minimum edit distance, which is the maximum number of matched edges.

Let $I_M(S)$ and $I_F(S)$ be the male and female individuals of pedigree S, respectively. Let $G^M = (I_M(P) \cup I_M(Q), E(G_M))$ be the bipartite graph for males and $G_F = (I_F(P) \cup I_F(Q), E(G_F))$ the similar graph for females.

The edge weights for the male graph are $w_M(i,j)$ with $(i,j) \in E(G_M)$ and similarly, $w_F(i,j)$ for $(i,j) \in E(G_F)$. Let $s(i)$ be the gender of individual i. For every $(i,j) \in I(P) \times I(Q)$, we will have two nodes $i \in I_{s(i)}(P)$ and $j \in I_{s(i)}(Q)$. Edges are created as follows:

1. For all $i \in X$, create edge $(i,i) \in E(G_{s(i)})$.
2. For all $(i,j) \in I(P) \times I(Q)$ where $i \notin X$, $j \notin X$ and $s(i) = s(j)$, if i and j each have at least one common child $c \in X$, create $(i,j) \in E(G_{s(i)})$ with weight $w_{s(i)}(i,j)$ equal to the number of children having the same label.

When we compute the maximum-weight matching on these graphs G_M and G_F, we will obtain the matching of the parents of the labeled individuals that minimizes the number of mis-matched child edges. Notice that labeled individuals are forced to match with the corresponding individual with the same label in the other pedigree. This holds even for P and Q which are each collections of families.

Branch and Bound Algorithm. An obvious exact solution to that particular problem is to branch on all the possible matchings for each generation before continuing recursively to consider older generations. Indeed, this algorithm does not even require the generations in order to branch, since it can simply branch on every possible node-paring. We implemented a branch-and-bound algorithm that does this branching procedure and bounds the search if it becomes clear that the current matching will have a worse score than the best found so far. In the case of monogamy, we simply match monogamous couples rather than individual parents. In the case of regular pedigrees we can bound the search by finding the best possible matching via the two-generation algorithm. For each generation i and for some matching $i-1$ of the previous generations, construct the two-generation maximum-weight bipartite matching instance and solve it. The solution yields the maximum number of edges that can be shared by any pairing of the nodes at generation i given the existing node-pairings for generation $i-1$. Doing this improves the running-time of the branching process by eliminating unfruitful search branches.

Heuristic Random Matching Algorithm using D-Splits. The randomized matching algorithm for regular pedigrees is exceptionally simple. At each generation, among individuals of the same gender, choose a match-pair with probability proportional to the number of same-labeled individuals, $i \in X$, in the descendant sets of the two pedigrees. If there were separate paths from each leaf to the individual under consideration, then this algorithm would give equal weight to each path. However, these paths share edges, so it is difficult to analyze this heuristic.

This algorithm is polynomial, because at each generation it creates a $n \times n$ matrix of individual match probabilities for each gender (where there are $2n$ individuals per generation). The matches are then drawn iteratively from these probability matrices (without replacement of previously matched individuals).

Dynamic Programming Heuristic for Matched Generations. Suppose we have two pedigrees P and Q, each of which is made up of g generations with m males and m females in each generation and has no inter-generational mating. If the pedigrees P and Q are similar enough and the most recent generations of P and Q are entirely labeled with unique labels, we can use dynamic programming to find a good matching between them (and therefore their edit distance) in time exponential in the edit distance.

We will need some notation to describe the algorithm. First, let the generations of pedigree P be numbered with the youngest generation being generation 1 and the oldest generation being generation g. For a pedigree P and a subset $S \subseteq [g]$, let $P|_S$ denote the sub-pedigree containing the i-th generation of P for every $i \in S$. Let $M(S)$ be the set of all matchings from $P|_S$ to $Q|_S$. For $M \in M(S)$ and i such that $j \leq i$ for all $j \in S$, let $B_i(M)$ be the best matching distance of $P|_{\{1,\ldots,i\}}$ to $Q|_{\{1,\ldots,i\}}$ which equals M when restricted to $P|_S$. We will abuse notation a bit by letting $M(i)$ denote $M(\{i\})$ and letting $P|_i$ denote $P|_{\{i\}}$. Finally, for matching $M \in M(S)$ and $T \subseteq S$, let $d_T(M)$ be the match distance of M restricted to $P|_T$.

Our algorithm rests on the following simple relation, whose proof we omit.

Lemma 3. *For every $M \in M(\{i, i-1\})$, let M_{i-1} be the restriction of M to $P|_{\{i-1\}}$. Then we have*

$$B_i(M) = B_{i-1}(M_{i-1}) + d_{\{i,i-1\}}(M) \tag{1}$$

The problem with this recursive algorithm is that its run-time is exponential in m, the number of males/females in each generation. For each value of i and each fixed matching M_{i+1}, there are $(m!)^2$ possible matchings M_i to process. Therefore, the run-time of this algorithm is $O((m!)^{2g})$.

But if we know that the two pedigrees under consideration are sufficiently similar at each generation, there is no need to consider all matchings. Suppose we are promised that the optimal matching M of P to Q has $d_{\{i,i-1\}}(M) < k$ for every $1 < i \leq g$. Then in the algorithm above we would only need to process, for each $M_{i-1} \in M(i-1)$, the matchings $M \in M(\{i, i-1\})$ such that M restricted to $P|_{i-1}$ equals M_{i-1} and $d_{\{i,i-1\}}(M) < k$.

The enumeration of the set $\{M \in M(\{i, i-1\})\colon M|_{i-1} = M_{i-1}, d_{\{i,i-1\}}(M) < k\}$ can be done recursively by choosing a node of $P|_i$, matching it to a node in the i-th generation of Q, and then recursively doing the same to another unlabeled node in $P|_i$, all the time keeping track of the accrued cost of the matching so far and not making any assignment that will push that cost above k. The pseudo-code for this enumeration is given below.

Algorithm 2 EnumerateMatchings($M_i, M_{i-1}, cost$)

```
Require: Access to $P, Q$, pedigrees with $g$ generations of $m$ males/females each.
Require: $i$ specifies which pair of generations is getting matched.
 1: if $|M_i| = 2m$ then
 2:     Process($M_i, M_{i-1}$)
 3:     return
 4: end if
 5: $u \leftarrow$ a node of $P|_i$ unmatched by $M_i$
 6: for $a \in Q|_i$ such that $s(a) = s(u)$ and $a$ is unmatched by $M_i$ do
 7:     add $u \mapsto a$ to $M_i$
 8:     if $d(M_i|_{\{i,i-1\}}) \leq k$ then
 9:         EnumerateMatchings($M_i, M_{i-1}, d_{\{i,i-1\}}(M_i)$)
10:     end if
11:     remove $u \mapsto a$ from $M_j$
12: end for
13: return
```

In the pseudo-code, the function Process carries out the dynamic programming implied by the recursion in Lemma 3. It takes as input a matching M of generations i and $i+1$ that is made up of a matching M_i of the i-th generation and a matching M_{i-1} of the $i-1$-th generation. With this matching, it does the following: 1) Calculates the distance d(M) on the sub-pedigrees consisting of only generations i and $i+1$. 2) Retrieves $B_{i-1}(M_{i-1})$ 3) Stores that $B_i(M_i)$ equals $B_{i-1}(M_{i-1}) + d(M)$.

Step (1) is easily done in $O(m^2)$ time. For steps (2) and (3) to be efficient, the values of $B(-)$ cannot be stored efficiently. To do this, we index the nodes of generation i from 1 to $2m$. Let $M_i(j)$ be the node in the second pedigree to which node j of generation i is mapped by M_i. Then, for each i, maintain a labeled tree that starts out empty and to which a matching M_i is added as follows: traverse the path with labels $(M_i(1), M_i(2), \ldots)$ until a node $v = M_i(j_0)$ is reached that does not have a child labeled $M_i(j_0+1)$, then add $M_i(j_0+1)$ as a child of $M_i(j_0)$, $M_i(j_0+2)$ as a child of $M_i(j_0+1)$, and so on. A matching is located in this tree by traversing the unique path with labels $(M_i(1), M_i(2), \dots, M_i(2m))$. This scheme allows the Process function to run in time $O(m)$ each time it is called.

But how many times is Process called? The following two lemmas together bound the number of matchings processed by EnumerateMatchings.

Lemma 4. *The number of matchings on which EnumerateMatchings will call the Process function is at most $T(m,k)^2$ where T is defined by the recurrence relation*

$$T(n,c) = T(n-1,c) + (n-1)T(n-1,c-2)$$

with initial conditions $T(1,c) = 1$ and $T(n,0) = T(n,1) = 1$.

Proof. First, suppose that there are only m individuals of one gender to match. Initially, EnumerateMatchings has m individuals whom it has to match and k "cost points" that it can use up. In the best case, there is only one person in $Q|_i$ to whom u can be matched without using any cost points (increasing the cost of the matching). This follows from the case with one child, where parent and child in one pedigree are matched to parent and child in the other pedigree. Besides this option there are at most $m-1$ other options, each of which will increase the cost of the matching by at least 2 (since at least one edge will have to be deleted and one edge will have to be added). This establishes the recurrence. The initial conditions follow from the following facts: 1) when one person is left to be matched, there is only one possible way to complete the matching being built, and 2) when the matching being built is already costing k (i.e. there are 0 cost points left), there is at best only one way to complete the matching.

This bounds the number of matchings of each gender by $T(m,k)$. Since this process occurs independently for each gender, the number of total matchings is at most $T(m,k)^2$. □

Lemma 5. *The recurrence T defined above satisfies $T(n,c) = O(n^C)$ where C is the greatest even integer less than or equal to c.*

Proof. We prove this by induction on c. The initial conditions of T give us our base case of $c = 0$. The general case follows from bounding the difference between successive values of $T(\cdot, c)$: the recurrence gives us that $T(n,c) - T(n-1,c) = (n-1)T(n-1,c-2)$, which is $nO(n^{C-2}) = O(n^{C-1})$ by the inductive hypothesis. □

The run-time of our algorithm is dominated by the last step, in which all the matchings between the first generations of P and Q are considered and the best one is chosen. By lemma 5, the number of these matchings is at most $O(m^{2k(g-1)}) = O(m^{2d})$ where d is the maximum possible distance between the two pedigrees. Thus, if k (the distance between pairs of successive generations) and g (the number of generations) are small, the algorithm can efficiently find a good edit distance.

Improvements to the DP Heuristic. For each generation and for a fixed labeling of the previous generation, similar to the two-generation matching problem, we can devise an instance of the minimum-weight perfect matching problem for which we can enumerate the k-best solutions. This can greatly improve the running-time of the EnumerateMatchings() method. For each generation i and for some matching $i-1$ of the previous generations, generate the two-generation maximum-weight bipartite matching instance for each gender with nodes $I_F(P) \cup I_F(Q)$ for the females and nodes $I_M(P) \cup I_M(Q)$ for the males. For sex S, we convert this into an instance of the minimum-weight perfect matching problem as follows. Create dummy vertices for half of the bipartite graph, where there is zero cost to match any vertex to a dummy in the opposite half of the graph. For vertices representing individuals in pedigree P we create $|I_S(Q)|$ dummy vertices which we call $D_S(Q)$, and respectively for Q we have $D_S(P)$ where $|D_S(P)| = |I_S(P)|$. So we have vertices $V = I_S(P) \cup D_S(Q) \cup I_S(Q) \cup D_S(P)$. with weights $w_S(i,j) = 0$ for $i \in D_S(Q), j \in V$ and for $i \in V, j \in D_S(P)$. Let $m = \max_{i,j} w_S(i,j)$ be the maximum weight edge in the graph. Now the weights $x_S(i,j)$ for the minimum-weight bipartite matching instance are then $x_S(i,j) = m - w_S(i,j)$ which yields non-negative weights.

Now, we have an instance of the minimum-weight perfect matching problem which we can solve. First, in the weight matrix, we delete any rows and columns where the nodes involved are two dummy nodes. Then we permute the columns of the weight matrix, so that the edges in the perfect matching are on the diagonal. Now, viewing the new matrix as the adjacency matrix of a di-graph, we notice that any near-best matchings must involve off diagonal edges. Enumerate the k-best cycles and disjoint combination of cycles to obtain the k-best perfect matchings. This can be done efficiently using Eppstein's k-shortest path algorithm [6]. Alternatively, Chegireddy and Hamacher give another algorithm for finding the k-best perfect matchings [3].

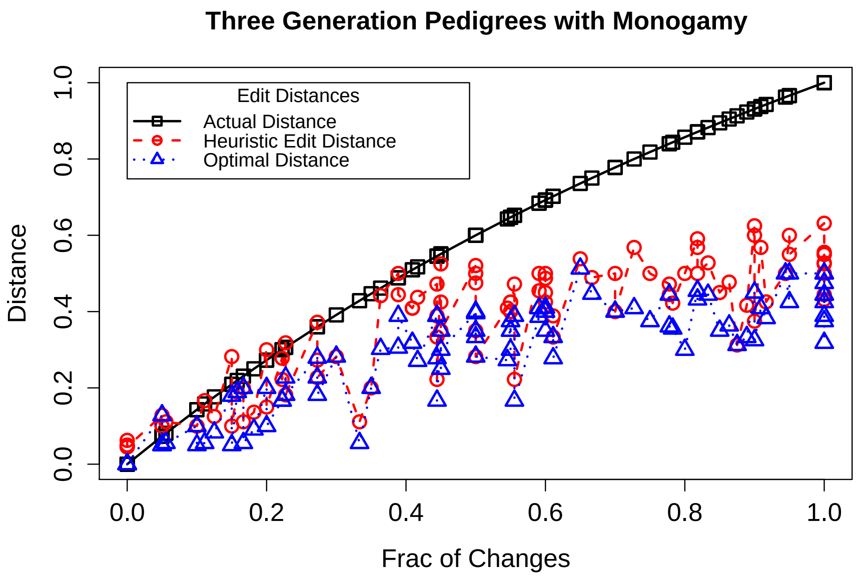
6 Simulation Results

We evaluated this algorithm on several random pairs of pedigrees at a range of distances. For each pair, the first pedigree graph was drawn from a Wright-Fisher simulation where every generation has a fixed number of individuals, n, and each monogamous parent-pair has a number of offspring drawn from the Poisson distribution with mean λ. The second pedigree was chosen based on the first in one of two ways. To generate a monogamous second pedigree, a proportion x of non-founders chose a couple uniformly at random to be their parents. To generate a non-monogamous second pedigree, a proportion x of non-founders chose a parent of each gender uniformly and independently at random. (Fig. 3 has one panel for each type). Notice that the simulation generated pedigrees in which individuals of the same generation mate only with each other.

We compared three different normalized distance estimates for pedigrees P and Q.

1. Actual Edit Path Distance: $A_{P,Q}/(|E(P)| + |E(Q)|)$
2. Random-Matching Edit Distance: $\hat{D}_{P,Q}/(|E(P)| + |E(Q)|)$
3. Optimal Edit Distance: $D^*_{P,Q}/(|E(P)| + |E(Q)|)$

where $A_{P,Q}$ is the actual edge changes in the simulation, $\hat{D}_{P,Q}$ is the random-matching edit distance, $D^*_{P,Q}$ is the optimal edit distance. These distances are plotted in Figure 3 against the actual number of changes. Figure 4 shows the difference between the random-matching and optimal edit distances. We see the random-matching heuristic performs reasonably well. Since this heuristic runs in polynomial time and performs reasonably well, we recommend it's use.



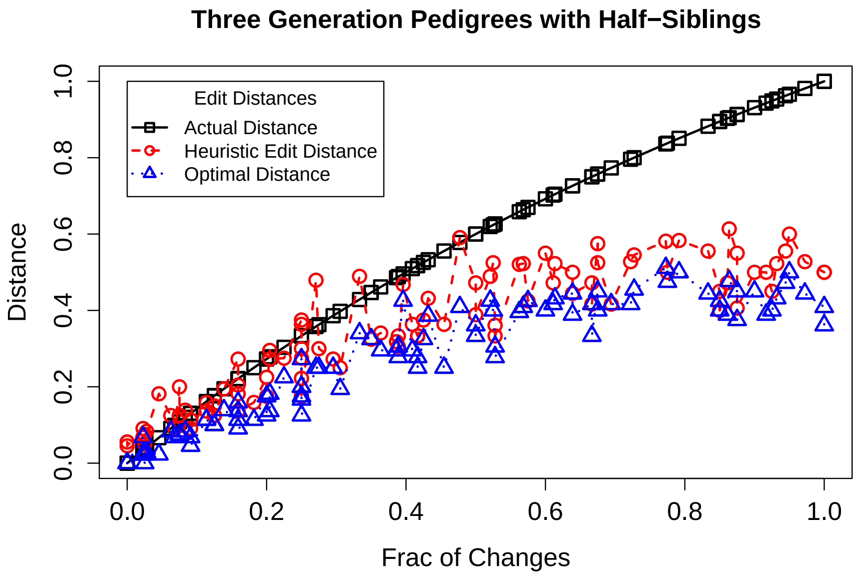
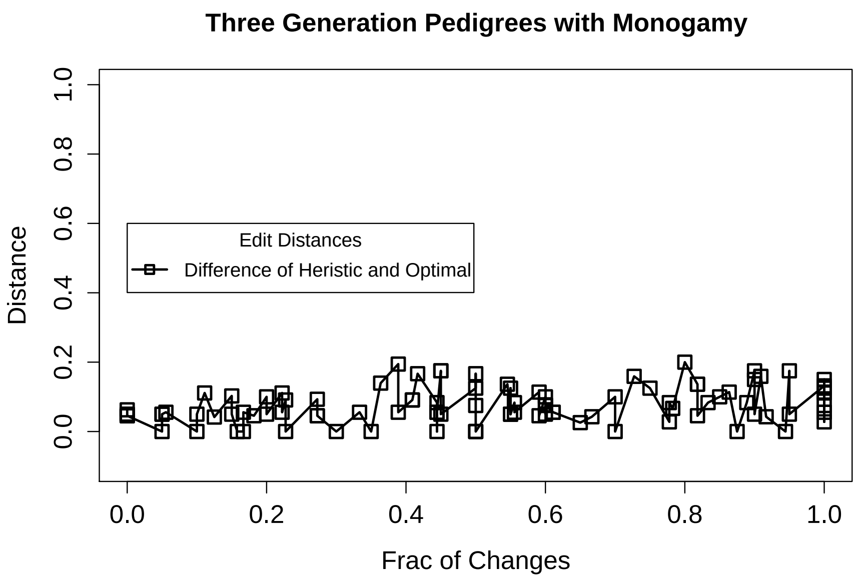


Fig. 3. Comparing Different Distances Estimates. Three different accuracy measures, the actual edit accuracy, the estimated edit accuracy, and the kinship accuracy. There were 100 pedigrees simulated with $n = 6$ parent-pairs and $\lambda = 3$. Clearly the random matching algorithm yields an estimated edit accuracy which is fairly close to the actual and which has less variance than the kinship accuracy. The left panel matches two different monogamous pedigrees while the right panel matches a monogamous pedigree to a pedigree containing half-siblings.



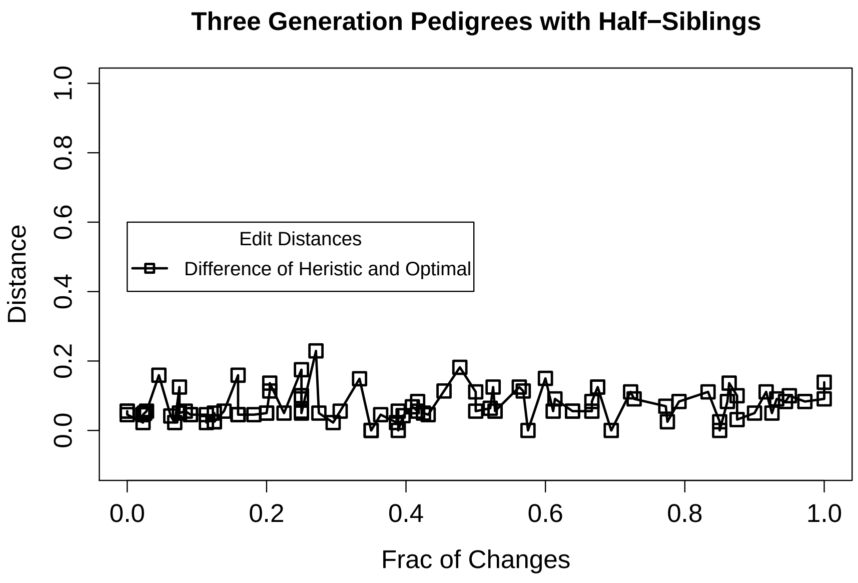


Fig. 4. Comparing Heuristic versus Optimal Distances. Three different accuracy measures, the actual edit accuracy, the estimated edit accuracy, and the kinship accuracy. There were 100 pedigrees simulated with $n = 6$ parent-pairs and $\lambda = 3$. Clearly the random matching algorithm yields an estimated edit accuracy which is fairly close to the actual and which has less variance than the kinship accuracy. The left panel matches two different monogamous pedigrees while the right panel matches a monogamous pedigree to a pedigree containing half-siblings.

7 Discussion

In this paper, we introduce a novel formulation for pedigree graphs that may be more robust than the standard parent-child formulation of a pedigree. We also introduce the problem of comparing pedigree graphs via an edge cut-and-paste distance, as well as show this problem to be APX-hard. We give several exact algorithms and several heuristics for this problem. The heuristics were implemented and compared with the optimal edit distance on simulated examples.

One interesting open problem is that of fractional matchings. Instead of allowing a one-to-one matching of nodes in the two pedigrees, suppose that fractional matchings were allowed. This instance would likely be easier computationally, although the biological interpretation is less clear. Perhaps, the computational easy of this instance would make this problem worth consideration.

Another open problem of interest is how these algorithms work on non-regular pedigrees, i.e. with inter-generational mating. The simulations used here were based on the Wright-Fisher model and did not allow any inter-generational mating events. It may be of interest to model the pedigrees using a birth-death model such as the Moran model where inter-generational mating is allowed. Such a simulation would allow the evaluation of the distance heuristics on non-regular pedigrees.

Also of great interest is understanding pedigrees having missing genealogical information, where not all the parent-child edges or the descendant splits are known in one or both of the pedigrees. There are two issues of interest. First, how robust is the descendant-split formulation to missing information? Second, is there an edit distance between pedigrees that can allow missing information?

Algorithms for comparing pedigrees are useful to researchers who currently build pedigrees manually. In particular it is quite possible to have multiple genealogical sources that might produce different pedigrees. The ability to compare these pedigrees to find the differences is essential. This paper presents one such approach.

Acknowledgments. Many thanks go to Yun Song for stimulating discussions and to Eran Halperin for the random pedigree simulation. Thanks also go to the National Science Foundation for support through the Graduate Research Fellowship.

References

1. M. Boehnke and N. J. Cox. Accurate inference of relationships in sib-pair linkage studies. *American Journal of Human Genetics*, 61:423–429, 1997.
2. S. Browning and B.L. Browning. On reducing the statespace of hidden Markov models for the identity by descent process. *Theoretical Population Biology*, 62(1):1–8, 2002.
3. C. R. Chegireddy and H. W. Hamacher. Algorithms for finding the k-best perfect matchings. *Discrete Applied Mathematics*, 18:155–165, 1987.
4. X. Chen, L. Liu, Z. Liu, and T. Jiang. On the minimum common integer partition problem. *ACM Trans. on Algorithms*, 5(1), 2008.
5. D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 2004.
6. David Eppstein. Finding the k shortest paths. *SIAM J. Comput.*, 28(2):652–673, 1999.
7. M. Fishelson, N. Dovgolevsky, and D. Geiger. Maximum likelihood haplotyping for general pedigrees. *Human Heredity*, 59:41–60, 2005.
8. D. Geiger, C. Meek, and Y. Wexler. Speeding up HMM algorithms for genetic linkage analysis via chain reductions of the state space. *Bioinformatics*, 25(12):i196, 2009.

9. Anderson K.G. How well does paternity confidence match actual paternity? evidence from worldwide nonpaternity rates. *Curr. Anthropol.*, 47:513–520, 2006.
10. B. Kirkpatrick. Haplotype versus genotypes on pedigrees. *WABI 2010: Proceedings for the 10th Workshop on Algorithms in Bioinformatics*, 2010.
11. B. Kirkpatrick. Pedigree reconstruction using identity by descent. *Technical Report No. UCB/EECS-2010-43*, 2010.
12. S. L. Lauritzen and N. A. Sheehan. Graphical models for genetic analysis. *Statistical Science*, 18(4):489–514, 2003.
13. J. Li and T. Jiang. An exact solution for finding minimum recombinant haplotype configurations on pedigrees with missing data by integer linear programming. In *Proceedings of the 7th Annual International Conference on Research in Computational Molecular Biology*, pages 101–110, 2003.
14. X Li, X-L Yin, and J Li. Efficient identification of identical-by-descent status in pedigrees with many untyped individuals. *Bioinformatics*, 26(12):i191–i198, 2010.
15. M.S. McPeek and L. Sun. Statistical tests for detection of misspecified relationships by use of genome-screen data. *Amer. J. Human Genetics*, 66:1076 – 1094, 2000.
16. A. Piccolboni and D. Gusfield. On the complexity of fundamental computational problems in pedigree analysis. *Journal of Computational Biology*, 10(5):763–773, 2003.
17. C. Semple and M. Steel. *Phylogenetics.* Oxford University Press, 2003.
18. Leigh W. Simmons, Rene E C. Firman, Gillian Rhodes, and Marianne Peters. Human sperm competition: testis size, sperm production and rates of extrapair copulations. *Animal Behavior*, 68:297–302, 2004.
19. L. Sun, K. Wilder, and M.S. McPeek. Enhanced pedigree error detection. *Hum. Hered.*, 54(2):99–110, 2002.
20. Bhalchandra D. Thatte. Combinatorics of pedigrees, 2006.
21. Bhalchandra D. Thatte and Mike Steel. Reconstructing pedigrees: A stochastic perspective. *Journal of Theoretical Biology*, 251(3):440 – 449, 2008.
22. E. A. Thompson. *Pedigree Analysis in Human Genetics.* Johns Hopkins University Press, Baltimore, 1985.
23. K. Zhang and T. Jiang. Some max snp-hard results concerning unordered labeled trees. *Inf. Process. Lett.*, 49(5):249–254, 1994.