Linear Programming Heuristics for the Graph Isomorphism Problem

Reza Takapoui Stephen Boyd

November 1, 2016

Abstract

An isomorphism between two graphs is a bijection between their vertices that preserves the edges. We consider the problem of determining whether two finite undirected weighted graphs are isomorphic, and finding an isomorphism relating them if the answer is positive. In this paper we introduce effective probabilistic linear programming (LP) heuristics to solve the graph isomorphism problem. We motivate our heuristics by showing guarantees under some conditions, and present numerical experiments that show effectiveness of these heuristics in the general case.

1 Graph isomorphism problem

1.1 Problem statement

Consider two weighted undirected graphs, each with n vertices labeled $1, \ldots, n$, described by their adjacency matrices $A, \tilde{A} \in \mathbf{R}^{n\times n}$, where A_{ij} is the weight on the edge in the first graph between vertices i and j, and zero if there is no edge between vertices i and j (and similarly for $\tilde{A}$). Since the graphs are undirected, the adjacency matrices are symmetric. The two graphs are isomorphic if there is a permutation of the vertices of the first graph that makes the first graph the same as the second. This occurs if and only if there is a permutation matrix $P \in \mathbf{R}^{n\times n}$ (*i.e.*, a matrix with exactly one entry in each row and column that is one, with the other zero) that satisfies $PAP^T = \tilde{A}$. We will say that the permutation matrix P transforms A to $\tilde{A}$ if $PAP^T = \tilde{A}$, or equivalently, $PA = \tilde{A}P$.

The graph isomorphism problem (GIP) is to determine whether such a permutation matrix exists, and to find one if so. GIP can be formulated as the (feasibility) optimization problem

$$\begin{array}{ll} \text{find} & P \\ \text{subject to} & PA = \tilde{A}P \\ & P\mathbf{1} = \mathbf{1}, \quad P^T\mathbf{1} = \mathbf{1} \\ & P_{ij} \in \{0,1\}, \quad i,j = 1,\ldots,n, \end{array} \tag{1}$$

with variable $P \in \mathbf{R}^{n\times n}$, where $\mathbf{1}$ is the vector with all entries one. The data for this problem are the adjacency matrices of the two graphs, A and $\tilde{A}$. The constraints on the last two lines

enforce that P is a permutation matrix. This problem has n^2 scalar variables (*i.e.*, P_{ij}), and all constraints except the last one (which requires the variables to take on Boolean values 0 or 1) are linear in P. The problem (1) is an integer (or Boolean) linear program (LP).

If P transforms A to $\tilde{A}$, the two matrices are similar, and therefore have the same spectrum, *i.e.*, the same eigenvalues including multiplicity. This observation gives a very simple spectral condition for isomorphism: The spectra of A and $\tilde{A}$ must be the same. Since this condition is readily checked, we will henceforth assume that this is the case.

Assumption 1. *Adjacency matrices A and $\tilde{A}$ have the same eigenvalues including multiplicity.*

Assumption 1 does not imply that the graphs are isomorphic. But if assumption 1 does not hold, the graphs are surely not isomorphic.

GIP arises in several applications including chem-informatics, mathematical chemistry, electronic design automation, and network theory. For example it can be used in determining if two descriptions of a molecule are the same [Irn05], or whether the physical layout of an electronic circuit correctly reflects the given circuit schematic diagram [Col81].

1.2 Computational complexity

GIP is one of the few problems in NP that has so far resisted all attempts to be classified as NP-complete, or within P. In 1979, Garey and Johnson [GJ79] listed 12 such problems and the GIP is one of only two on that list whose complexity remains unresolved. For many restricted graph classes, polynomial time algorithms are known. This is, for example, the case for trees [K$^+$57], planar graphs [HW74], interval graphs [LB79], partial k-trees [Bod90], graphs of bounded degree [Luk80], or graphs with bounded eigenvalue multiplicity [BGM82]. Recently, Babai announced a quasipolynomial time algorithm for all graphs, *i.e.*, one with running time $2^{O((\log n)^c)}$ for some fixed $c > 0$ [Bab15].

This paper does not advance the discussion of computational complexity of GIP; instead we describe effective heuristics for it. However, many of the ideas we will encounter (*e.g.*, compact graphs, graph spectrum) are closely connected to those that arise in research on its computational complexity.

1.3 A simple relaxation

One relaxation of problem (1) can be obtained by replacing constraints $P_{ij} \in \{0, 1\}$ with interval relaxation $0 \leq P_{ij} \leq 1$. The relaxed problem can be written as

$$\begin{array}{ll} \text{find} & P \\ \text{subject to} & PA = \tilde{A}P \\ & P\mathbf{1} = \mathbf{1}, \quad P^T\mathbf{1} = \mathbf{1} \\ & P \geq 0, \end{array} \tag{2}$$

where the last constraint is entry-wise. The constraint $P \leq 1$ is removed because it is implied by $P\mathbf{1} = P^T\mathbf{1} = \mathbf{1}$ and $P \geq 0$. This problem has n^2 scalar variables, $n^2 + 2n$ scalar

linear equality constraints (some of which are redundant), and n^2 scalar linear inequality constraints. While (1) is a Boolean LP, the relaxation (2) is an LP, and readily solved in polynomial time using different methods such as interior-point methods or ellipsoid method [BV04].

If the relaxed problem (2) is infeasible, then A and $\tilde{A}$ are obviously not isomorphic. Therefore, from this point of the paper on, we assume that this problem is feasible.

Assumption 2. *We assume the relaxed problem (2) is feasible.*

Assumption 2 does not imply that the graphs are isomorphic. But if assumption 2 does not hold, the graphs are surely not isomorphic. The necessary and sufficient conditions for feasibility of this LP has been studied in [SU11] and is closely related to common equitable partitions of two graphs.

The set of feasible points for problem (2) is a polytope in $\mathbf{R}^{n\times n}$. Permutation matrices transforming A to $\tilde{A}$ are extreme points or vertices of this polytope. (However, not every extreme point of this polytope is necessarily a permutation matrix.) Given assumption 2, GIP comes down to finding a permutation matrix among extreme points of the feasible set of problem (2). Our algorithms are all based on variations and extensions of this observation.

Surprisingly, under some conditions on the graphs (stated in §2.4), this relaxation is tight; that is, the set of feasible point for problems (1) and (2) are both singletons. Hence, it is sufficient to solve problem (2) to find a permutation matrix transforming A to $\tilde{A}$, or certify that A and $\tilde{A}$ are not isomorphic if problem (2) is infeasible.

1.4 Outline

We will describe the basic version of the algorithm in §2, then describe sparsity constraints in §3 to tighten the relaxation and increase the probability of success for the heuristic. An ADMM-based algorithm to solve the problem is described in §5, and finally in §6 we study some examples and numerical experiments.

2 Algorithm

2.1 Basic randomized subroutine

As we discussed in §1.3, the set of feasible points for problem (2) form a polytope and permutation matrices transforming A to $\tilde{A}$ are extreme points or vertices of this polytope. In order to encourage finding an extreme point of this polytope, we minimize a random linear objective over the set of feasible points. Hence our basic randomized subroutine is to generate a random matrix $W \in \mathbf{R}^{n\times n}$ with i.i.d. Gaussian entries and solve the LP

$$\begin{array}{ll} \text{minimize} & \mathbf{Tr}(W^T P) \\ \text{subject to} & PA = \tilde{A}P \\ & P\mathbf{1} = \mathbf{1}, \quad P^T\mathbf{1} = \mathbf{1} \\ & P \geq 0. \end{array} \tag{3}$$

If the solution to problem (3) found happens to be a permutation matrix we conclude that we solved this instance of GIP, since A and $\tilde{A}$ are isomorphic and the solution $P^\star$ is a permutation matrix that transforms A to $\tilde{A}$. On the other hand, if the solution found is not a permutation matrix, we cannot say anything about A and $\tilde{A}$.

Notice that with probability 1 the solution of problem (3) is unique. Let p^{succ} denote the probability under the distribution of W that the randomized subroutine finds a permutation matrix when A and $\tilde{A}$ are isomorphic. It is easy to see that p^{succ} is only a function of A (and does not depend on $\tilde{A}$). We briefly comment here that $p^{\text{succ}} > 0$.

Since scaling W does not change the optimization problem, without loss of generality we can assume that W is chosen uniformly from an (n^2-1)-sphere in an n^2 dimensional space. Consider the set of permutation matrices in $\mathbf{R}^{n\times n}$ that transform A to $\tilde{A}$. Centered around each such permutation matrix, there is a cone with nonzero solid angle such that if $-W$ is in that cone, the solution of problem (3) will be a permutation matrix. Probability of success p^{succ} is the probability of W being in one of these cones, and hence the probability that this algorithm successfully finds an acceptable permutation matrix.

Even though solving just one instance of problem (3) might look unsophisticated, it turns out to be an effective heuristic to solve the GIP in some cases. It is guaranteed to find a solution of GIP (with probability one) under some conditions which are discussed in next subsection.

Standard solvers based on the simplex method and interior-point methods can be used to solve problem (3), but can be inefficient in practice. However, we will show in §5 that the special structure of this problem enables us to use a custom method to solve this optimization problem more efficiently.

2.2 Polishing

After the basic randomized subroutine converges to a doubly stochastic matrix $P^* \in \mathbf{R}^{n\times n}$, we use can the Hungarian algorithm [Kuh55] to project P^* on the set of doubly stochastic matrices, which we denote by $\tilde{P}^*$. Finding the closest permutation matrix to P^* (in Frobenius norm) can be done in $O(n^3)$ time. If $\tilde{P}^*A = \tilde{A}\tilde{P}^*$, then a permutation mapping A to $\tilde{A}$ is found. This step, as will see in §6, can increase the probability of success of the subroutine.

2.3 Repeated randomized algorithm

As we discussed in §2.1, choosing a random instance of W and running the basic randomized algorithm will find a permutation that relates A and $\tilde{A}$ with probability p^{succ}. We also discussed in §2.2 that we can use Hungarian algorithm to (potentially) increase the probability of success. If we repeat this basic randomized algorithm for N independently chosen random instances of W the probability of success is $1-(1-p^{\text{succ}})^N$. For example if the probability of success for the basic randomized subroutine is 80%, after repeating the basic subroutine 4 times the probability of success will be 99.9%.

We evidently have $p^{\text{succ}} > 0$, meaning that by solving problem (3) there is a positive probability of finding a permutation matrix that that transforms A to $\tilde{A}$. In particular, by

repeatedly solving problem (3) with independent random choices of W, we will solve GIP with probability one. This probability can be extremely small, however, so the expected number of LPs we must solve to solve GIP can be extremely large. The benefit of this probabilistic method over a deterministic spectral-based heuristic is that when a deterministic heuristic fails, there is no hope to recover the permutation that relates A and $\tilde{A}$, but this probabilistic method can be used repeatedly to find the underlying permutation. This randomized algorithm can potentially have false negatives, *i.e.*, when A and $\tilde{A}$ are isomorphic, it might fail to find a permutation relating them.

2.4 Theoretical guarantees

In this subsection we show that under some conditions, solving problem (3) is guaranteed to find a permutation solution if A and $\tilde{A}$ are isomorphic, in other words, $p^{\text{succ}} = 1$. Specifically if A has distinct eigenvalues, and for every eigenvector v of A we have $\mathbf{1}^T v \neq 0$, the relaxation in problem (2) is tight.

In order to show this, assume $QA = \tilde{A}Q$, where Q is a permutation matrix and let P be a doubly stochastic matrix such that $PA = \tilde{A}P$. Also let $A = V\Lambda V^T$ be an eigendecompostion of A. Defining $R = V^T Q^T P V$, we have

$$R\Lambda = V^T Q^T P V \Lambda = V^T Q^T P A V = V^T Q^T \tilde{A} P V = V^T A Q^T P V = \Lambda V^T Q^T P V = \Lambda R.$$

Hence for every i, j we have $R_{ij}\Lambda_{ii} = \Lambda_{jj} R_{ij}$, or equivalently $R_{ij}\left(\Lambda_{ii} - \Lambda_{jj}\right) = 0$. Since $\Lambda_{ii} - \Lambda_{jj} \neq 0$ for $i \neq j$, all off-diagonal entries of R must be zero and the matrix R must be diagonal. Also $RV^T\mathbf{1} = V^T Q^T P \mathbf{1} = V^T Q^T \mathbf{1} = V^T \mathbf{1}$, which implies that $R_{ii}(V^T\mathbf{1})_i = (V^T\mathbf{1})_i$. Our second assumption on V enforces that $(V^T\mathbf{1})_i \neq 0$ and $R_{ii} = 1$ for all i and hence $P = QVRV^T = QVV^T = Q$.

For graphs specified above, relaxing the set of permutation matrices to the set of doubly stochastic matrices does not extend the set of feasible points of problem (1), which is a singleton.

A graph with adjacency matrix A is said to be a *compact graph* if the set of feasible points to problem (3) is the convex hull of the set of feasible points to problem (1). For example, all graphs with two aforementioned properties are compact. The concept of *compact graphs* was introduced by Tinhofer [Tin91], who proved that trees and cycles (which violate the two aforementioned assumptions) and the disjoint union of isomorphic copies of a compact graph are compact. If A is the adjacency matrix of a compact graph and A and $\tilde{A}$ are isomorphic, then $p^{\text{succ}} = 1$. It is not a hard problem to solve the graph isomorphism problem between two trees or two cycles (or some other compact graphs), but it is interesting to see that this problem can be solved (with probability 1) by only solving one LP.

3 Sparsity constraints

We discussed one possible relaxation of problem (1) in §1.3. However, we can find tighter relaxations by adding convex constraints about P that we know must be true. This will

create a tighter relaxation of the original problem which can potentially increase p^{succ} for the randomized algorithm. Specifically, we consider the following extension

$$\begin{array}{ll} \text{minimize} & \mathbf{Tr}(W^T P) \\ \text{subject to} & PA = \tilde{A}P \\ & P\mathbf{1} = \mathbf{1}, \quad P^T\mathbf{1} = \mathbf{1} \\ & P \geq 0 \\ & P_{ij} = 0 \quad (i,j) \in \mathcal{K}, \end{array} \tag{4}$$

where $\mathcal{K}$ is a set of pairs of indices. The difference between this extension and problem (3) is the last line of the constraints, which requires some entries in P to be zero. If $\mathcal{K} = \emptyset$, this problem reduces to problem (3). In general, problem (4) can be considered as a problem with $n^2 - \|\mathcal{K}\|$ scalar variables. Our efficient method for solving the LP relaxation (described in §5) can handle these constraints efficiently.

There are different ways to find a proper set $\mathcal{K}$. The simplest and least useful choice of $\mathcal{K}$ is the empty set. The maximal $\mathcal{K}$, denoted by $\mathcal{K}^{\max}$ is the set of pairs of indices i, j such that $P_{ij} = 0$ for every permutation matrix P transforming A to $\tilde{A}$. Any valid set $\mathcal{K}$ will satisfy $\emptyset \subseteq \mathcal{K} \subseteq \mathcal{K}^{\max}$.

We will show below how to find pairs of indices i, j such that for any permutation matrix P satisfying $PA = \tilde{A}P$, we must have $P_{ij} = 0$.

Lemma 1. *Let P be a permutation matrix. If $Pa = b$ for two given vectors $a, b \in \mathbf{R}^n$ and $P_{ij} = 1$, then we must have $a_j = b_i$. This implies that if $a_j \neq b_i$ then $P_{ij} = 0$.*

This simple lemma can be used to construct $\mathcal{K}$. If we know that $Pa = b$ for any permutation matrix that transforms A to $\tilde{A}$, we can include the pair (i, j) in $\mathcal{K}$. If all entries of vector a are distinct, then $\mathcal{K}$ contains $n^2 - n$ entries and permutation P is uniquely determined. At the other extreme, if all entries of vector a are equal, this lemma adds no new information.

The idea here is to use graph invariants to find equalities of the form $Pa = b$ and then construct a set $\mathcal{K}$ from these equalities. By graph invariant, we mean any function of nodes that is independent of labeling the nodes. For instance, the number of paths with length k starting from a node is an invariant, where k is a positive integer. (We will discuss this in next subsection.) In principle, the method can be used with any invariant. But we are interested in invariants that are easy to compute, *e.g.*, degree invariants and spectral invariants.

3.1 Degree invariants

A simple linear equality that holds is $P(\mathbf{diag}\, A) = \mathbf{diag}\, \tilde{A}$. This means that vertices that are mapped to each other must have the same self-edge weights. If A and $\tilde{A}$ have no self-edges (*i.e.*, diagonal entries of A and $\tilde{A}$ are zero), this equality will add no information.

A more interesting equality that holds is $P(A^k\mathbf{1}) = \tilde{A}^k\mathbf{1}$ for every positive integer k. In other words, the number of paths of length k starting from a vertex must be the same for two vertices that map to each other.

For example, when the graphs are unweighted (*i.e.*, the edges have weight one) this equality with $k = 1$ means that the degrees of mapped vertices must be the same. In other words, all nodes of degree i must be mapped to each other. Hence, if the number of nodes with degree i is denoted by n_i, the original problem has $(n_1 + n_2 + \cdots)^2$ variables, but we know only $n_1^2 + n_2^2 + \cdots$ of them can potentially be nonzero. In the extreme case that all degrees are equal (or equivalently the graph is regular), this method does not eliminate any variables from P.

This equality is valid for every positive integer k. However, according to the Cayley-Hamilton theorem, A^k can be written as a linear combination of $A, A^2, \ldots, A^n$ for every positive k. Hence for a given pair of indices i, j if $(A^k)_i = (A^k)_j$ for $k = 1, \ldots, n$, we will have $(A^k)_i = (A^k)_j$ for every positive k. Therefore it is only enough to consider this equality for $k = 1, \ldots, n$.

In summary, we have two sets of equalities:

- $P(\mathbf{diag}\, A) = \mathbf{diag}\, \tilde{A}$.
- $P(A^k\mathbf{1}) = \tilde{A}^k\mathbf{1}$, for $k = 1, \ldots, n$.

We denote the set of pairs of indices constructed this way by $\mathcal{K}^{\text{degree}}$. Clearly, we have $\mathcal{K}^{\text{degree}} \subseteq \mathcal{K}^{\max}$.

3.2 Spectral invariants

As mentioned earlier, we assume that A and $\tilde{A}$ share the same spectrum; otherwise the graphs are evidently non-isomorphic. Let λ be an eigenvalue of A with multiplicity k and columns of $V \in \mathbf{R}^{n\times k}$ be an orthonormal basis for eigenspace associated with λ, hence we have $AV = \lambda V$ and $V^TV = I_k$, where I_k denotes the identity matrix in $\mathbf{R}^{k\times k}$. Similarly, assume that columns of $\tilde{V} \in \mathbf{R}^{n\times k}$ are an orthonormal basis of eigenspace associated with λ, hence we have $\tilde{A}\tilde{V} = \lambda\tilde{V}$ and $\tilde{V}^T\tilde{V} = I_k$. We have

$$\tilde{A}(PV) = (\tilde{A}P)V = (PA)V = P(AV) = P(\lambda V) = \lambda(PV).$$

In other words columns of PV are eigenvectors of $\tilde{A}$ associated with λ. Therefore we have $PV = \tilde{V}Q$ where $Q \in \mathbf{R}^{k\times k}$. We observe that

$$Q^TQ = Q^T\tilde{V}^T\tilde{V}Q = V^TP^TPV = V^TV = I_k.$$

Hence Q is an orthogonal in $\mathbf{R}^{k\times k}$. Therefore $PV = \tilde{V}Q$ implies that $P(VV^T)P^T = \tilde{V}\tilde{V}^T$, and we have the following equalities:

- $P(\mathbf{diag}\, VV^T) = \mathbf{diag}\, \tilde{V}\tilde{V}^T$.
- $P(VV^T\mathbf{1}) = \tilde{V}\tilde{V}^T\mathbf{1}$.

Notice that $P\left(\left(VV^T\right)^k\mathbf{1}\right) = (\tilde{V}\tilde{V}^T)^k\mathbf{1}$ adds no more information for $k > 1$, since $(VV^T)^k = VV^T$ and $(\tilde{V}\tilde{V}^T)^k = \tilde{V}\tilde{V}^T$.

These equalities hold for any eigenvalue λ. We denote the set of pairs of indices constructed this way by $\mathcal{K}^{\text{spectral}}$. Clearly, we have $\mathcal{K}^{\text{spectral}} \subseteq \mathcal{K}^{\max}$.

3.3 Constructing $\mathcal{K}$

In summary, here is how we construct $\mathcal{K} = \mathcal{K}^{\text{degree}} \cup \mathcal{K}^{\text{spectral}}$. We start with $\mathcal{K} = \emptyset$, and we sequentially consider the degree invariant equalities for $k = 1, \ldots, n$ and spectral invariant equalities for every eigenvalue λ. For each equality, we add the new pairs of disallowed i, j to the set $\mathcal{K}$.

When $\mathcal{K}$ is constructed, the number of variables that could possibly be nonzero is $n^2 - |\mathcal{K}|$. In §6 we will report the number of elements in $\mathcal{K}$ for our experiments, and we see how the sparsity constraints can increase the probability of success for our heuristic, for each LP solved.

Simple examples show that $\mathcal{K}^{\text{degree}}$ and $\mathcal{K}^{\text{spectral}}$ are not necessarily subsets of each other. Therefore in general it is a good idea to include both of these constraints in the basic subroutine. Also, our examples show that *pruning* can be a helpful technique for making $\mathcal{K}$ larger. In pruning, we disallow pair i, j if no neighbor of i can be mapped to a neighbor of j.

One reasonable conjecture could be that choosing the maximal $\mathcal{K}$ would result in solving the problem with probability 1. In other words if $\mathcal{K} = \mathcal{K}^{\max}$ then $p^{\text{succ}} = 1$. This conjecture is wrong, however. It can be shown that for the Petersen graph $\mathcal{K}^{\max} = \emptyset$. Also $(1/3)A$ is a doubly stochastic matrix that commutes with A. If this conjecture were true, then A could be written as the convex combination of automorphisms of the Petersen graph. Each such automorphism Π has the property that $\Pi(v)$ is connected to v for all vertices v. But the only automorphism of the Petersen graph that has this property is the identity. Hence the Petersen graph disproves this conjecture.

4 Algorithm summary

A summary of the algorithm is presented below.

Algorithm 1 Randomized LP heuristic

```
Check whether assumptions 1, 2 hold. If not, declare A and Ã as non-isomorphic.
Construct 𝒦 as described in §3.3
for random instance W_1, ..., W_N do
    Solve problem (4) to find a solution P*
    Use Hungarian algorithm to find the closest permutation matrix P̃* to P*
    if P̃*A = ÃP̃* then
        Declare A and Ã as isomorphic and return P̃*
    end if
end for
```

Here N denotes the number of times problem (4) is solved and can be chosen beforehand based on an approximation of p^{succ}. As mentioned before, the probability of the success of this algorithm is $1 - (1 - p^{\text{succ}})^N$.

5 Solution method

5.1 ADMM algorithm

Any LP solver such as ones base on the simplex method or interior-point methods can be used to solve problem (4). With n^2 variables and n^2 equality constraints, using a general purpose interior-point solver would involve $O(n^6)$ flops. However, the special structure of the problem enables us to use a custom method which is more efficient. We use the alternating direction method of multipliers (ADMM) in consensus form [BPC+11] for solving problem (4) as follows. We write the problem as

$$\begin{array}{ll} \text{minimize} & f_1(P_1) + f_2(P_2) + f_3(P_3) \\ \text{subject to} & Z = P_i, \quad i = 1,2,3 \\ & ZA = \tilde{A}Z, \end{array} \tag{5}$$

where

$$\begin{aligned} f_1(P_1) &= (1/2)\,\mathbf{Tr}(W^T P_1) + \mathbf{I}(P_1\mathbf{1} = \mathbf{1}), \\ f_2(P_2) &= (1/2)\,\mathbf{Tr}(W^T P_2) + \mathbf{I}(P_2^T\mathbf{1} = \mathbf{1}), \\ f_3(P_3) &= \mathbf{I}(P_3 \geq 0) + \mathbf{I}\left((P_3)_{i,j} = 0, \quad (i,j) \in \mathcal{K}\right). \end{aligned}$$

Here $\mathbf{I}$ is the indicator function, and takes the value 0 if its argument is true, the value ∞ if its argument is false. Assuming that $A = V\Lambda V^T$ and $\tilde{A} = \tilde{V}\Lambda\tilde{V}^T$ are eigenvalue decompositions of A and $\tilde{A}$, the ADMM updates will be

$$\begin{aligned} P_1^{k+1} &= \underset{P_1\mathbf{1}=\mathbf{1}}{\operatorname{argmin}} \left(\mathbf{Tr}\left(\left(W/2 + Y_1^k\right)^T W\right) + (1/2)\|P_1 - Z^k\|_F^2\right) \\ P_2^{k+1} &= \underset{P_2^T\mathbf{1}=\mathbf{1}}{\operatorname{argmin}} \left(\mathbf{Tr}\left(\left(W/2 + Y_2^k\right)^T W\right) + (1/2)\|P_2 - Z^k\|_F^2\right) \\ P_3^{k+1} &= \Pi_{\{P|f_3(P)=0\}}(Z^k - Y_3^k) \\ Z^{k+1} &= \Pi_{\{Z|ZA=\tilde{A}Z\}}\left(\frac{P_1^{k+1} + P_2^{k+1} + P_3^{k+1}}{3}\right) \\ Y_1^{k+1} &= Y_1^k + P_1^{k+1} - Z^{k+1} \\ Y_2^{k+1} &= Y_2^k + P_2^{k+1} - Z^{k+1} \\ Y_3^{k+1} &= Y_3^k + P_3^{k+1} - Z^{k+1}, \end{aligned}$$

where $\Pi_{\mathcal{C}}$ denotes the projection operator onto $\mathcal{C}$. Without loss of generality, we chose $\rho = 1$ in ADMM, because scaling ρ is equivalent to scaling W. After generating a random direction W, we scale W such that the absolute value of elements of W averages to 1. This is only for the ADMM algorithm to converge faster, and it does not affect the solution found by the ADMM. (Remember that the solution to problem (5) is unique with probability 1.)

In order to find the projection onto $\{Z|ZA = \tilde{A}Z\}$ we notice that $ZA = \tilde{A}Z$ is equivalent to $ZV\Lambda V^T = \tilde{V}\Lambda\tilde{V}^T Z$, which is equivalent to $\tilde{V}^T ZV\Lambda = \Lambda\tilde{V}^T ZV^T$, or equivalently $\tilde{V}^T ZV$

commutes with Λ. Hence, $ZA = \tilde{A}Z$ if and only if $(\tilde{V}^T Z V)_{ij} = 0$ for any i, j that $\Lambda_{ii} \neq \Lambda_{jj}$. After simplifying the steps above, the ADMM update steps will reduce to

$$\begin{aligned}
P_1^{k+1} &= Z^k - W/2 - Y_1^k - (1/n)\left(\left(Z^k - W/2 - Y_1^k\right)\mathbf{1} - \mathbf{1}\right)\mathbf{1}^T \\
P_2^{k+1} &= Z^k - W/2 - Y_2^k - (1/n)\mathbf{1}\left(\mathbf{1}^T\left(Z^k - W/2 - Y_2^k\right) - \mathbf{1}^T\right) \\
P_3^{k+1} &= \max\left(Z^k - Y_3^k, 0\right) \circ S \\
Z^{k+1} &= \tilde{V}\left(\left(\tilde{V}^T \frac{P_1^{k+1} + P_2^{k+1} + P_3^{k+1}}{3} V\right) \circ R\right) V^T \\
Y_1^{k+1} &= Y_1^k + P_1^{k+1} - Z^{k+1} \\
Y_2^{k+1} &= Y_2^k + P_2^{k+1} - Z^{k+1} \\
Y_3^{k+1} &= Y_3^k + P_3^{k+1} - Z^{k+1},
\end{aligned}$$

where $\circ$ denotes entry wise product, and R_{ij} is 1 if $\lambda_i = \lambda_j$ and 0 otherwise, and $S_{ij} = 1$ if $(i, j) \in K$ and 0 otherwise.

The first three update rules are the first proximal operator in ADMM algorithm and are, in fact, minimizing a quadratic function over an affine subspace. Hence, they are linear operators in P, and are $O(n^2)$ flops. The fourth equality is the projection of $(P_1^{k+1} + P_2^{k+1} + P_3^{k+1})/3$ onto $\{Z | ZA = \tilde{A}Z\}$, and is of $O(n^3)$ complexity. Finally, the last three equalities are dual updates, and can be done in $O(n^2)$ flops. Therefore each iteration of the ADMM algorithm is done in $O(n^3)$ flops.

The practical complexity of this heuristic is $O(n^3)$, which is basically the square root of the one obtained with interior-point methods. We observe that this algorithm usually converges within tens to hundreds of ADMM iterations.

6 Examples

We use several test cases to examine our heuristic. In the first subsection we discuss a specific example, the Frucht Graph, in order to give a qualitative description of the heuristic. Through this example, we will describe how this heuristic can benefit from using graph invariant constraints. Then we use several standard classes of graphs for benchmarking algorithms. We will report our experiment results for undirected random graphs, two dimensional grids, cubic Hype-Hamiltonian graphs, Kronecker Eye Filp graphs, Miayzaky Augmented graphs, and affine geometries graphs [SFSV03, JK11, JK07, LPAC11, LPA09, JK09]. These graphs are described in §6.2. We observe that this heuristic is more effective for some classes of graphs. We also notice that the time complexity of this heuristic is $O(n^3)$, and it is more effective when graph invariant constraints are taken into account. The code for our solver and our examples can be found at `https://github.com/cvxgrp/graph_isom`.

6.1 Frucht graph

The Frucht graph is an example of a graph with no nontrivial automorphisms, that violates the assumptions in §2.4. As shown in Figure 1, the Frucht graph is a 3-regular graph with

12 vertices and 18 edges. It was first described by Robert Frucht in 1939.

We permute nodes of the Frucht graph randomly and run the algorithm for $100,000$ times with and without sparsity mask. Each run uses a random instance of linear objectives (W). We observe that without using sparsity extension, in 25.2% of random instances the algorithm finds the permutation matrix. When we used sparsity mask (graph invariant constraints, or $\mathcal{K}$), which contains only 14 nonzeros, the algorithm found the correct permutation in every single instance of $100,000$ runs, so our guess is that $p^{\text{succ}} = 1$ when sparsity constraints are taken into account.

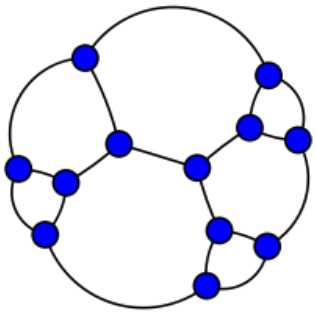

Figure 1: The Frucht graph is an asymmetric 3-regular graph with 12 vertices and 18 edges.

6.2 Database of graphs for benchmarking algorithms

We use several classes of graphs in [SFSV03, JK11, JK07, LPAC11, LPA09] to evaluate the effectiveness of this heuristic on a variety of graphs. In particular, our experiments were on the following classes of graphs.

- Undirected random graphs (R1N family) is taken from [SFSV03]. In these undirected Erdos-Renyi graphs, there is an edge between two vertices with probability 0.1.
- Cubic Hypo-Hamiltonian graphs clique-connected (CCH family) is proposed in [LPAC11], and is built using as basic block two non-isomorphic HypoHamiltonian graphs with 22 vertices.
- Affine geometries graphs (AG2 family) is a part of the benchmark of bliss [JK09]. It contains point-line graphs of affine geometries $AG_2(q)$.
- Two dimensional grids (G2N family) also comes from [SFSV03] benchmark and is discussed in [KBL07]. They are introduced for simulating applications dealing with regular structures as those operating at the lower levels of an image processing task.
- Kronecker Eye Flip graphs (KEF family) is a part of the benchmark of bliss [JK09].

- Miayzaky Augmented graphs (MZA family) are also taken from the benchmark of bliss [JK09].

For each graph, we ran the algorithm for 50 random instances of W and have summarized the results in Table 1 below. The first column states the graph type, and the second column states the ratio of possibly nonzero entries in P. In other words the number in the second column is $1 - \frac{|\mathcal{K}|}{n^2}$. The third and fourth column represent the percentage of runs that a permutation was successfully found with and without sparsity constraints ($\mathcal{K}$), respectively.

As we discussed in §5, each iteration of the ADMM algorithm is $O(n^3)$, hence we expect that the runtime of the heuristic be a cubic function of the number of vertices of the graphs. This is because we observe that the number of iterations that ADMM requires before it convergence does not change dramatically as the number of vertices changes. Figure 2 shows the scatter plot of runtime of the algorithm (denoted by t) versus the number of vertices (denoted by n) in log-scale for 50 runs over random undirected graphs (R1N family). This runtime excludes the polishing step, which is done using Hungarian algorithm. We observe that the average runtime is linear in the size of the graph when plotted in log-scale. For comparison we also used ECOS [DCB13], an interior-point method for second-order cone programming. We see that our ADMM solver is far more scalable than interior-point methods.

Graph type	n	sparsity ratio	success rate without $\mathcal{K}$	success rate with $\mathcal{K}$
R1N	20	0.050	1.0	1.0
R1N	40	0.025	1.0	1.0
R1N	60	0.017	1.0	1.0
R1N	80	0.012	1.0	1.0
R1N	100	0.010	1.0	1.0
R1N	200	0.005	1.0	1.0
R1N	400	0.003	1.0	1.0
R1N	600	0.002	1.0	1.0
R1N	800	0.001	1.0	1.0
R1N	1000	0.001	1.0	1.0
CHH	22	0.393	0.74	0.88
CHH	44	0.205	0.86	0.96
CHH	88	0.204	0.72	0.96
CHH	132	0.226	0.78	0.92
CHH	198	0.225	0.66	0.96
CHH	264	0.204	0.54	0.90
CHH	352	0.205	0.48	0.86
CHH	440	0.212	0.32	0.92
CHH	550	0.212	0.30	0.82
CHH	660	0.204	0.38	0.82
CHH	792	0.205	0.34	0.82
CHH	924	0.208	0.26	0.82
AG2	10	0.520	0.34	1.0
AG2	21	0.510	0.20	1.0
AG2	35	0.506	0.12	0.72
AG2	55	0.504	0	0
AG2	105	0.502	0	0
AG2	136	0.501	0	0
G2N	16	0.375	0.54	0.98
G2N	36	0.186	0.52	0.98
G2N	64	0.110	0.46	1.0
G2N	81	0.078	0.44	0.96
G2N	100	0.072	0.42	1.0
G2N	196	0.038	0.58	0.98
G2N	400	0.019	0.52	0.98
G2N	576	0.013	0.54	1.0
G2N	784	0.010	0.48	0.90
KEF	32	0.375	0.68	0.80
KEF	50	0.270	0.82	0.86
KEF	72	0.167	0.78	0.88
KEF	98	0.103	0.62	0.92
KEF	128	0.062	1.0	1.0
KEF	242	0.046	1.0	1.0
KEF	392	0.036	1.0	1.0
KEF	578	0.029	1.0	1.0
KEF	800	0.025	1.0	1.0
KEF	1058	0.021	1.0	1.0
MZA	40	0.159	1.0	1.0
MZA	80	0.085	0.62	0.76
MZA	120	0.057	0.22	0.28
MZA	160	0.044	0.24	0.32
MZA	200	0.035	0.24	0.32
MZA	240	0.029	0.08	0.16
MZA	280	0.025	0	0.08
MZA	320	0.022	0	0.08
MZA	360	0.019	0	0

Table 1: The outcome for 50 instances of our heuristic on different problems.

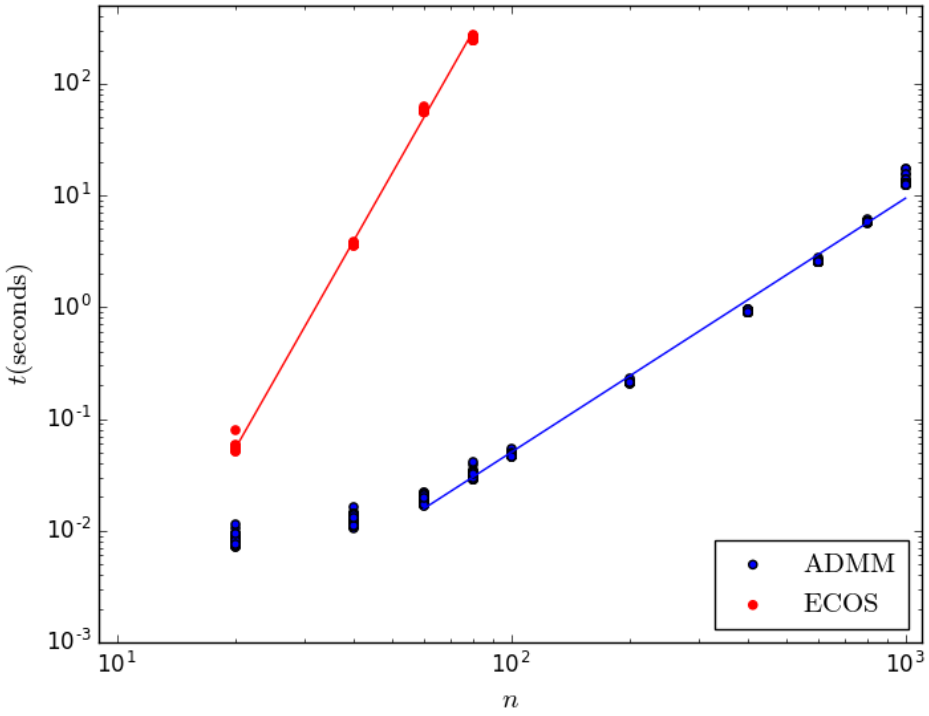

Figure 2: Runtime versus graph size in log-scale for 50 random runs.

References

[ABK14] Y. Aflalo, A. Bronstein, and R. Kimmel. Graph matching: Relax or not? *arXiv preprint arXiv:1401.7623*, 2014.

[ABK15] Y. Aflalo, A. Bronstein, and R. Kimmel. On convex relaxation of graph isomorphism. *Proceedings of the National Academy of Sciences*, 112(10):2942–2947, 2015.

[Bab15] L. Babai. Graph isomorphism in quasipolynomial time. *arXiv preprint arXiv:1512.03547*, 2015.

[BGM82] L. Babai, D. Y. Grigorye, and D. M. Mount. Isomorphism of graphs with bounded eigenvalue multiplicity. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 310–324. ACM, 1982.

[Bod90] H. L. Bodlaender. Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *Journal of Algorithms*, 11(4):631–643, 1990.

[BPC+11] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[Col81] C. J. Colbourn. On testing isomorphism of permutation graphs. *Networks*, 11(1):13–21, 1981.

[DCB13] A. Domahidi, E. Chu, and S. Boyd. ECOS: An SOCP solver for embedded systems. In *European Control Conference (ECC)*, pages 3071–3076. IEEE, 2013.

[ER63] R. Erdős and A. Rényi. Asymmetric graphs. *Acta Mathematica Hungarica*, 14(3):295–315, 1963.

[Fru49] R. Frucht. Graphs of degree three with a given abstract group. *Canadian J. Math*, 1:365–378, 1949.

[FS14] M. Fiori and G. Sapiro. On spectral properties for graph matching and graph isomorphism problems. *arXiv preprint arXiv:1409.6806*, 2014.

[GJ79] M. R. Garey and D. S. Johnson. Computers and intractability: A guide to np-completeness, 1979.

[HW74] J. E. Hopcroft and J. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 172–184. ACM, 1974.

[Irn05] C. M. Irniger. Graph matching–filtering databases of graphs using machine learning techniques. 2005.

[JK07] T. Junttila and P. Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX)*, 12:135–149, 2007.

[JK09] T. A. Junttila and P. Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. http://www.tcs.hut.fi/Software/bliss/benchmarks/index.shtml, April 2009.

[JK11] T. Junttila and P. Kaski. Conflict propagation and component recursion for canonical labeling. In *Theory and Practice of Algorithms in (Computer) Systems*, pages 151–162. Springer, 2011.

[K+57] P. J. Kelly et al. A congruence theorem for trees. *Pacific J. Math*, 7(0957):961–968, 1957.

[KBL07] A. Kandel, H. Bunke, and M. Last. *Applied graph theory in computer vision and pattern recognition*, volume 52. Springer, 2007.

[Kuh55] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[LB79] G. S. Lueker and K. S. Booth. A linear time algorithm for deciding interval graph isomorphism. *Journal of the ACM (JACM)*, 26(2):183–195, 1979.

[LFF+14] V. Lyzinski, D. Fishkind, M. Fiori, J. T. Vogelstein, C. E. Priebe, and G. Sapiro. Graph matching: Relax at your own risk. *arXiv preprint arXiv:1405.3133*, 2014.

[LPA09] J. L. López-Presa and A. F. Anta. Fast algorithm for graph isomorphism testing. In *International Symposium on Experimental Algorithms*, pages 221–232. Springer, 2009.

[LPAC11] J. L. López-Presa, A. F. Anta, and L. N. Chiroque. Conauto-2.0: Fast isomorphism testing and automorphism group computation. *arXiv preprint arXiv:1108.1060*, 2011.

[Luk80] E. M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. In *21st Annual Symposium on Foundations of Computer Science*, pages 42–49. IEEE, 1980.

[SFSV03] M. De Santo, P. Foggia, C. Sansone, and M. Vento. A large database of graphs and its use for benchmarking graph isomorphism algorithms. *Pattern Recognition Letters*, 24(8):1067–1079, 2003.

[SU11] E. R. Scheinerman and D. H. Ullman. *Fractional Graph Theory: A Rational Approach to the Theory of Graphs*. Courier Corporation, 2011.

[Tin91] G. Tinhofer. A note on compact graphs. *Discrete Applied Mathematics*, 30(2):253–264, 1991.

[Ume88] S. Umeyama. An eigendecomposition approach to weighted graph matching problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(5):695–703, 1988.