# AC⊕DC search: behind the winning solution to the Flywire graph-matching challenge

**Anonymous authors**
**Paper under double-blind review**

## Abstract

This paper describes the alternating continuous and discrete combinatorial (AC⊕DC) optimizations behind the winning solution to the Flywire Ventral Nerve Cord Matching Challenge. The challenge was organized by the Princeton Neuroscience Institute and held over three months, ending on January 31, 2025. During this period, the challenge attracted teams of researchers with expertise in machine learning, high-performance computing, graph data mining, biological network analysis, and quadratic assignment problems. The goal of the challenge was to align the connectomes of a male and female fruit fly, and more specifically, to determine a one-to-one correspondence between the neurons in their ventral nerve cords. The connectomes were represented as large weighted graphs, and the challenge was posed as a problem in graph matching: how does one find a permutation that maps the nodes of one graph onto the nodes of another? The winning solution to the challenge alternated between two complementary approaches to graph matching—the first, a combinatorial optimization over the symmetric group of permutations, and the second, a continuous relaxation of this problem to the space of doubly stochastic matrices. For the latter, the doubly stochastic matrices were optimized by combining Frank-Wolfe methods with a fast preconditioner to solve the linear assignment problem at each iteration. We provide a complete implementation of these methods with a few hundred lines of code in MATLAB. Notably, this implementation obtains a winning score to the challenge in less than 15 minutes on a laptop computer.

## 1 Introduction

The problem of graph matching is to find the best permutation that maps the nodes and edges of one graph onto those of another. The problem arises in many areas of science and engineering where graphs are used to encode similarity, co-dependence, or the flow of information (Conte et al., 2004; Mamano & Hayes, 2017; Haller et al., 2022). The problem is solved in practice by maximizing an objective function that scores each permutation by quantifying the similarity of nodes and edges that it brings into correspondence. Graph matching is in general an NP-hard problem; one can find a globally optimal solution by exhaustively considering all permutations, but this is only possible for very small graphs. For larger problems, the best solvers rely on approximate search algorithms that attempt to find a high-scoring match.

A particular instance of this problem was at the heart of a recent challenge[1] posed by the FlyWire consortium at the Princeton Neuroscience Institute. The goal of the challenge was to align the connectomes of a male and female fruit fly, where each connectome was represented by a large sparse graph. The nodes in these graphs represented neurons, and the edges indicated which neurons were connected by synapses. These graphs were also directed and weighted: each edge counted the number of synapses observed in a particular direction between two neurons. The determination of these connectomes—down to the level of individual synapses—was the culmination of many years of painstaking research (Dorkenwald et al., 2024; Devineni, 2024), and the Flywire consortium has since issued a stream of open challenges to further process these results. This particular challenge was posed in the winter of 2024, and its purpose was to study how gender differences in male and female fruit flies are manifested in the connectomes of their ventral nerve cords (VNCs).

[1]`https://codex.flywire.ai/app/vnc_matching_challenge`

There were two features of this challenge that made for an especially compelling problem in graph-matching. First was the problem size: each VNC connectome was represented by a graph with $n = 18524$ nodes, far larger than many previous problems in this space. By comparison, for example, the hermaphrodite *C. elegans* connectome consists of only 302 neurons (Chen et al., 2016; Varshney et al., 2011). At the same time, the graphs in the VNC challenge were small enough that soft matches, represented by dense $n \times n$ matrices, could fit into the memory of a modestly equipped computer. It was therefore possible to explore approaches that took advantage of this ability and did not rely on purely combinatorial techniques.

The second distinguishing feature of the challenge was the particular way that different matches were scored. Let $\pi$ denote the permutation of the indices $\{1, 2, \ldots, n\}$ that maps $i$ to $\pi_i$, and let $A$ and $B$ denote, respectively, the sparse matrices whose nonzero elements record edge weights in the male and female connectomes. The alignment score for the VNC matching challenge was computed as

$$\mathcal{S}(\pi) = \sum_{ij} \min\bigl(A_{ij}, B_{\pi_i \pi_j}\bigr)\,. \tag{1}$$

Eq. (1) is a variant on the weighted Jaccard distance, and it was chosen by the challenge organizers because it appeared to correlate better with known biological isomorphisms. In particular, they found that eq. (1) was more robust with respect to these isomorphisms than scores based on simple correlation or cosine distance. The score in eq. (1) can be computed efficiently by restricting the sum to nonzero elements of $A$ and $B$.

The challenge ran for three months, and it attracted teams of researchers with expertise in machine learning, high-performance computing, graph data mining, biological network analysis, and quadratic assignment problems. The challenge organizers provided a baseline match that was determined from neuron cell types, and this benchmark solution (though not the cell type metadata) was made available for teams to use as a warm start. Nearly all teams continued to submit improved solutions up to the deadline on January 31, 2025, and the scores of these submissions were independently verified by the challenge organizers. Throughout the challenge, scores were not disclosed, and different teams did not share code or details of their solutions.

In this paper, we reveal the methods behind the winning solution to the challenge. The solution combined two complementary approaches to graph matching: the first was a combinatorial optimization over the space of permutation matrices (Mamano & Hayes, 2017), and the second was a continuous relaxation of this problem to the space of doubly stochastic matrices (Vogelstein et al., 2015). Both of these approaches were pursued individually by other teams, but it was the combination of these approaches that led to a winning solution. We refer to this alternation of continuous and discrete combinatorial methods as AC$\oplus$DC search.

The continuous relaxation of this problem is obtained by extending the score in eq. (1) to the convex set of doubly stochastic matrices. The relaxed objective is given by

$$\mathcal{S}(P) = \sum_{ijk\ell} \min(A_{ij}, B_{k\ell}) P_{ik} P_{j\ell}, \tag{2}$$

where $P$ is an $n \times n$ nonnegative matrix whose rows and columns sum to one. Note that the objective in eq. (2) is quadratic *but not concave* in its argument; also, when $P$ is dense, it appears naively to require $O(n^4)$ operations to perform its quadruple sum. The winning solution optimized eq. (2) by adapting Frank-Wolfe methods for constrained convex optimization (Frank & Wolfe, 1956) alongside a fast preconditioner to solve the linear assignment problem at each iteration. This type of relaxation has been used successfully for other quadratic assignment problems (Vogelstein et al., 2015). The main novelties described in this paper are highly optimized routines for computing the gradient of eq. (2) and projecting this gradient into the space of permutation matrices.

With these techniques, we show how to obtain a winning score to the challenge in under 15 minutes, on a laptop computer, with a few hundred lines of code in MATLAB. Though not revealed at the time—because teams on the leaderboard were listed by the date of their most recent submission—the AC$\oplus$DC methods held the top score for the final forty days of the challenge. These methods should be of direct interest to other researchers in biological network analysis, and they should also be of general interest to researchers in machine learning whose problems require optimizations over the permutation group.

The organization of this paper is as follows. In section 2, we describe a greedy search algorithm for the combinatorial optimization of eq. (1) over the space of permutation matrices. This approach has the advantage of simplicity, and it also directly optimizes the score in eq. (1). But it improves the score slowly in the early stages of optimization, and in the later stages, it is prone to getting stuck. In section 3, we describe a first-order method for the continuous optimization of eq. (2). This method has the advantage that it makes rapid initial progress, but it does not converge to a permutation matrix that maximizes eq. (1). In section 4, we describe the winning solution that is found by alternating these approaches, and we also discuss further techniques (albeit with diminishing returns) for optimizing the scores in eqs. (1) and (2). Finally, in section 5, we conclude with a discussion of open problems and directions for future work.

## 2 Discrete search

There are many ways to search for a permutation that maximizes the score in eq. (1). Arguably the simplest is a hill-climbing approach that makes local moves in the space of $n!$ permutations. This approach can equivalently be viewed as an optimization over the space of $n \times n$ permutation matrices—that is, matrices whose elements are equal to zero or one and whose rows and columns sum to one. Within this approach, there are also many types of local moves that can be considered, but the simplest are those that swap exactly one pair of indices (viewing permutations as shuffles) or exactly one pair of rows (viewing them as matrices). In this section, we describe an efficient way to evaluate these moves and illustrate the strengths and weaknesses of this approach.

### 2.1 Evaluation of pairwise swaps

Our first goal is to evaluate how the score in eq. (1) is changed by single pairwise swaps. The following notation will be useful. Let $P^\pi$ denote the $n \times n$ permutation matrix corresponding to the permutation $\pi$, whose elements are given by $P^\pi_{ik} = \delta(\pi_i, k)$, where $\delta(\cdot,\cdot)$ is the Kronecker delta function. Similarly, let $\sigma_{ij}$ denote the permutation that swaps $i$ and $j$ while leaving all other indices intact, and let $\pi \circ \sigma_{ij}$ denote the composition (from right to left) of these two permutations—that is, the permutation obtained by first swapping $i$ and $j$ and then permuting the indices according to $\pi$.

We begin by computing the $n \times n$ symmetric matrix $\Delta^\pi$ whose elements record the difference in scores between permutations that are related by a single pairwise swap of indices. In particular, let

$$\Delta^\pi_{ij} = \mathcal{S}(\pi \circ \sigma_{ij}) - \mathcal{S}(\pi), \tag{3}$$

so that the diagonal elements of $\Delta^\pi$ are zero, while the nonzero elements indicate those local moves in the space of permutations that change the score. Note that for this challenge, with $n = 18524$ nodes per graph, each matrix $\Delta^\pi$ records the effect of over 171 million pairwise swaps.

Let $\mathcal{P}$ denote the convex set of doubly stochastic $n \times n$ matrices. Since the score in eq. (2) is quadratic in the matrix $P \in \mathcal{P}$, the differences in eq. (3) can also be expressed in terms of the gradient and Hessian of this score. The gradient of eq. (2) is given by the $n \times n$ matrix with elements

$$\big[\nabla \mathcal{S}(P)\big]_{j\ell} = \sum_{ik} \Big[\min(A_{ij}, B_{k\ell}) + \min(A_{ji}, B_{\ell k})\Big] P_{ik}, \tag{4}$$

and it is generally a *dense* matrix even when the matrices $A$, $B$, and $P$ in eq. (4) are sparse. Of particular interest is the form of this gradient at the permutation matrix $P^\pi$. We denote this gradient by $G^\pi = \nabla \mathcal{S}(P^\pi)$, and its elements are given by

$$G^\pi_{j\ell} = \sum_{i} \Big[\min(A_{ij}, B_{\pi_i \ell}) + \min(A_{ji}, B_{\ell \pi_i})\Big]. \tag{5}$$

The gradient in eq. (5) can be computed in $O(n^2)$ operations by exploiting the sparsity of the connectome weights in the matrices $A$ and $B$. In particular, for the matrices of size $n = 18524$ in this challenge, this gradient takes about 15 seconds to compute on a Macbook Pro (M1 Max) laptop.

**Algorithm 1** Given connectome weights $A, B \in \mathbb{R}^{n\times n}$ and a base permutation $\pi$, evaluate the score differences in eq. (8) obtained by a single pairwise swap of indices.

```
procedure $\Delta$ = EvaluateSwaps($A$,$B$,$\pi$)
    ▷ Compute gradient at $\pi$ and permute rows and columns of $B$
    for $i \leftarrow 1$ to $n$ do
        for $j \leftarrow 1$ to $n$ do
            $G_{ij} \leftarrow \sum_{k=1}^{n}[\min(A_{ki}, B_{\pi_k j}) + \min(A_{ik}, B_{j\pi_k})]$
            $B^{\pi}_{ij} \leftarrow B_{\pi_i \pi_j}$
        end for
    end for
    ▷ Evaluate difference in scores due to pairwise swaps
    for $i \leftarrow 1$ to $n$ do
        for $j \leftarrow 1$ to $n$ do
            $hA_{ii} \leftarrow \min(A_{ii}, B^{\pi}_{ii}) + \min(A_{ii}, B^{\pi}_{jj}) - \min(A_{ii}, B^{\pi}_{ij}) - \min(A_{ii}, B^{\pi}_{ji})$
            $hA_{jj} \leftarrow \min(A_{jj}, B^{\pi}_{ii}) + \min(A_{jj}, B^{\pi}_{jj}) - \min(A_{jj}, B^{\pi}_{ij}) - \min(A_{jj}, B^{\pi}_{ji})$
            $hA_{ij} \leftarrow \min(A_{ij}, B^{\pi}_{ii}) + \min(A_{ij}, B^{\pi}_{jj}) - \min(A_{ij}, B^{\pi}_{ij}) - \min(A_{ij}, B^{\pi}_{ji})$
            $hA_{ji} \leftarrow \min(A_{ji}, B^{\pi}_{ii}) + \min(A_{ji}, B^{\pi}_{jj}) - \min(A_{ji}, B^{\pi}_{ij}) - \min(A_{ji}, B^{\pi}_{ji})$
            $\Delta_{ij} \leftarrow G_{i\pi_j} + G_{j\pi_i} - G_{i\pi_i} - G_{j\pi_j} + hA_{ii} + hA_{jj} - hA_{ij} - hA_{ji}$
        end for
    end for
end procedure
```

Next we consider the way in which the Hessian of the score in eq. (2) enters into the calculation of differences in eq. (3). To this end, we introduce a new matrix $B^{\pi}$, whose elements are obtained by permuting the rows and columns of $B$ according to the permutation $\pi$; in particular,

$$B^{\pi}_{ij} = B_{\pi_i \pi_j}. \tag{6}$$

The elements of $B^{\pi}$ appear in certain combinations of Hessian elements that arise repeatedly in the calculation of the matrix $\Delta^{\pi}$. As further shorthand, we define the functions

$$h^{\pi}_{ij}(a) = \min(a, B^{\pi}_{ii}) \;+\; \min(a, B^{\pi}_{jj}) \;-\; \min(a, B^{\pi}_{ij}) \;-\; \min(a, B^{\pi}_{ji}), \tag{7}$$

where in practice we will take the argument $a$ to be a particular connectome weight from the matrix $A$. For example, when $a = A_{ii}$, the right side of eq. (7) expresses a particular linear combination of elements from the Hessian of eq. (2) evaluated at the matrix $P^{\pi}$.

With the above definitions, we can express the effects of swaps in eq. (3) in terms of the connectome weights and the gradient and Hessian of the score. In terms of these quantities, the elements of $\Delta^{\pi}$ are given by

$$\Delta^{\pi}_{ij} \;=\; G^{\pi}_{i\pi_j} + G^{\pi}_{j\pi_i} - G^{\pi}_{i\pi_i} - G^{\pi}_{j\pi_j} + h^{\pi}_{ij}(A_{ii}) + h^{\pi}_{ij}(A_{jj}) - h^{\pi}_{ij}(A_{ij}) - h^{\pi}_{ij}(A_{ji}), \tag{8}$$

and again, all $n^2$ matrix elements in this equation can be computed in $O(n^2)$ operations for a given permutation $\pi$. For the matrices of size $n = 18524$ in this challenge, it takes about 8 additional seconds to compute the elements in eq. (8) on top of the gradient in eq. (5). We provide pseudocode for a procedure (EvaluateSwaps) to compute these elements in Algorithm 1.

It is possible for none of the matrix elements $\Delta^{\pi}_{ij}$ in eq. (8) to be positive. When this is the case, it indicates that the permutation $\pi$ cannot be improved by a single pairwise swapping of indices. Otherwise, the largest (i.e., most positive) element of $\Delta^{\pi}$ indicates the pairwise swap that most improves the scoring function in eq. (1). This suggests a simple algorithm for greedy local search which we describe in the next section.

## 2.2 Greedy search with pairwise swaps

Starting from an initial permutation $\pi$, one can attempt to optimize the score in eq. (1) by alternating two procedures; the first evaluates the effect of each pairwise swap by computing its corresponding element in $\Delta^{\pi}$,

**Algorithm 2** Given connectome weights $A, B \in \mathbb{R}^{n\times n}$ and an initial permutation $\pi_0$, perform a greedy search with up to $\tau$ pairwise swaps (per iteration) to find a local optimum in the alignment score of eq. (1).

```
procedure $\pi$ = GreedySearch($A$,$B$,$\pi_0$,$\tau$)
    $\pi \leftarrow \pi_0$
    $\Delta \leftarrow$ EvaluateSwaps($A, B, \pi$)
    while $(\max_{ij}(\Delta_{ij}) > 0)$ do
        $\pi \leftarrow$ MakeSwaps($A, B, \pi, \Delta, \tau$)
        $\Delta \leftarrow$ EvaluateSwaps($A, B, \pi$)
    end while
end procedure

procedure $\pi$ = MakeSwaps($A$,$B$,$\pi$,$\Delta$,$\tau$)
    $\mathcal{S} \leftarrow \sum_{ij} \min(A_{ij}, B_{\pi_i\pi_j})$
    while $((\tau > 0)$ AND $(\max_{ij}(\Delta_{ij}) > 0))$ do
        $(i, j) \leftarrow \mathrm{argmax}_{ij}(\Delta_{ij})$
        $\pi' \leftarrow \pi \circ \sigma_{ij}$
        $\mathcal{S}' \leftarrow \sum_{ij} \min(A_{ij}, B_{\pi'_i\pi'_j})$
        if $(\mathcal{S}' > \mathcal{S})$ then
            $(\pi, \mathcal{S}, \tau) \leftarrow (\pi', \mathcal{S}', \tau - 1)$
        end if
        $(\Delta_{ij}, \Delta_{ji}) \leftarrow (0, 0)$
    end while
end procedure
```

and the second performs those swaps that seem likely to increase the score by the largest amount. We give the pseudocode for a greedy search based on these two procedures in Algorithm 2. The search terminates when the first procedure, sketched in Algorithm 1, returns a matrix $\Delta^\pi$ with no positive elements.

We use a threshold $\tau$ to adjust the balance of time spent in these two procedures. The MakeSwaps procedure in Algorithm 2 performs up to $\tau$ swaps that increase the score while skipping over swaps that do not. The threshold is only needed in the early stages of optimization, when the permutation $\pi$ is very far from optimal; in this case, the matrix $\Delta^\pi$ returned by the first procedure (EvaluateSwaps) may have an inordinately large number of positive elements. The MakeSwaps procedure considers pairwise swaps in descending order of their corresponding elements in $\Delta^\pi$. If the maximal element of $\Delta^\pi$ is positive, then the first such swap is guaranteed to yield a permutation with a higher score. However, successive swaps are *not* guaranteed to increase the score, even if they correspond to positive elements of $\Delta^\pi$, due to possible interference with previously executed swaps. (A trivial example of such interference arises from the symmetry of the matrix $\Delta^\pi$: a swap $j \leftrightarrow i$ will exactly negate the gain from an immediately preceding swap $i \leftrightarrow j$.)

The left panel of Fig. 1 shows results from the greedy search in Algorithm 2 for different thresholds on the maximum numbers of pairwise swaps per iteration. All of these runs were initialized from the benchmark solution with score 5154247 provided by the challenge organizers. The algorithm converges to different solutions for different thresholds $\tau$, but all of these solutions have scores around 5818K. These solutions are evidence of the large number of local maxima in this problem: there are many permutations whose scores cannot be improved by any pairwise swaps of indices (there are over 171 million possible pairwise swaps).

There are many ways to augment the greedy search in Algorithm 2 so that it discovers higher-scoring permutations. One is to introduce an element of randomness, sometimes performing a pairwise swap that decreases the score, as is done in simulated annealing (Mamano & Hayes, 2017). Another is to evaluate and perform higher-order moves that swap three or more indices at a time. While these approaches may require more resources, one can also optimize them aggressively, in faster languages than MATLAB, while exploiting opportunities for parallelism (e.g., multi-core, GPUs) (Koblentz, 2025).

Multiple teams experimented with these ideas in the days and weeks leading up to the deadline. But even with considerably longer runs, these more elaborate forms of discrete search were not able to reach the dashed line in Fig. 1, indicating a score (at 5850K) that was high enough to win the challenge. As mentioned earlier, however, this winning score can be obtained in under 15 minutes by combining discrete and continuous approaches to the optimization of eq. (1). With this goal in mind, we now turn to the latter approach.

## 3 Continuous relaxation

In this section we describe a complementary approach to this problem in graph-matching, one based on a continuous optimization over the convex set of doubly stochastic matrices. This type of relaxation has been

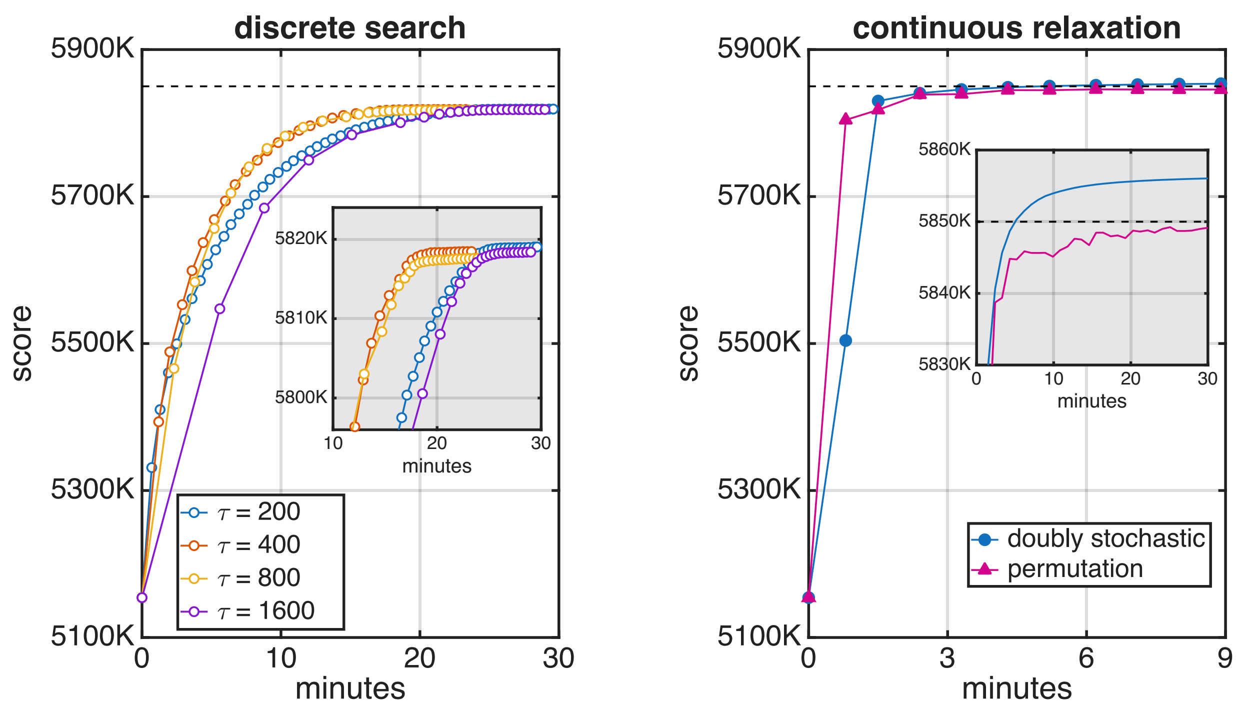

Figure 1: Alignment scores in eqs. (1–2) versus wall clock time starting from the benchmark solution at score 5154247. *Left.* Greedy discrete search in Algorithm 2 with up to $\tau$ pairwise swaps per iteration. *Right.* Frank-Wolfe updates in eqs. (11) and (13) to optimize the continuous relaxation in eq. (2). Neither method converges to a winning score for the challenge (indicated by the dashed line at 5850K).

studied previously for quadratic assignment problems (Vogelstein et al., 2015), but for the best results in the challenge this approach must be tailored specifically to the alignment score in eq. (2). This score is a quadratic function of $P$, but it is not concave, and therefore an iterative hill-climbing procedure is not guaranteed to find its global maximum. Here we show that an iterative procedure, based on the Frank-Wolfe algorithm for constrained convex optimization (Frank & Wolfe, 1956), can be adapted to this problem with extremely competitive results. One part of this procedure is to calculate a projected gradient, and we also describe a fast preconditioner for this calculation that is crucial to its overall efficiency.

### 3.1 Frank-Wolfe updates

The iterative procedure is shown in Algorithm 3, and it alternates between three steps. The first step computes the gradient in eq. (4). As mentioned previously, it takes about 15 sec to compute this gradient at a permutation matrix of size $n = 18524$. For this iterative procedure, we need to compute the gradient at doubly stochastic matrices, which in general can take much longer. As we shall see, however, the procedure converges very quickly, so that in practice—*if the search is initialized by a permutation matrix*—we only need to compute gradients for doubly stochastic matrices that are highly sparse. When this is the case, it takes only slightly longer to compute the gradient in eq. (4).

The second step of the iterative procedure projects this gradient back into the convex set of doubly stochastic matrices. In particular, this step computes

$$Q_t = \operatorname*{argmax}_{Q \in \mathcal{P}} \left( \text{trace}\left[ \nabla \mathcal{S}(P_t)^\top Q \right] \right). \tag{9}$$

Note that eq. (9) defines a linear program whose solution always lies at a *vertex* of the set $\mathcal{P}$; in other words, its solution $Q_t$ is not merely a doubly stochastic matrix, but also a permutation matrix. The optimization

**Algorithm 3** Given connectome weights $A, B \in \mathbb{R}^{n\times n}$ and an initial doubly stochastic matrix $P_0$, perform $T$ Frank-Wolfe updates to optimize the score in eq. (2), then return the doubly stochastic matrix $P_T$ and permutation matrix $\Pi_T$ found from these updates.

```
procedure $(P_T, \Pi_T) = \textsc{DoFrankWolfe}(A,B,P_0,T)$
    $P \leftarrow P_0$
    for $t \leftarrow 1$ to $T$ do
        $\triangleright$ Compute gradient
        for $i \leftarrow 1$ to $n$ do
            for $j \leftarrow 1$ to $n$ do
                $G_{ij} \leftarrow \sum_{k\ell} \big[\min(A_{ki}, B_{\ell j}) + \min(A_{ik}, B_{j\ell})\big] P_{k\ell}$
            end for
        end for
        $\triangleright$ Project gradient, compute step size, and interpolate
        $Q \leftarrow \mathrm{argmax}_{Q\in\mathcal{P}}\, \mathrm{trace}\big[G^\top Q\big]$
        $\alpha \leftarrow \mathrm{argmax}_{\alpha\in[0,1]}\, \big[\mathcal{S}((1-\alpha)P + \alpha Q)\big]$
        $P \leftarrow (1-\alpha)P + \alpha Q$
        $\triangleright$ Compute closest-matching permutation matrix
        $\Pi \leftarrow \mathrm{argmax}_{\Pi\in\mathcal{P}}\, \mathrm{trace}\big[P^\top \Pi\big]$
    end for
    $P_T \leftarrow P$
    $\Pi_T \leftarrow \Pi$
end procedure
```

in eq. (9) is most commonly known as the linear assignment problem, or the problem of perfect matching in a complete bipartite graph. It can be solved by the so-called Hungarian method (Kuhn, 1955) in polynomial time (Munkres, 1957; Edmonds & Karp, 1972; Tomizawa, 1971), but this is not the approach that was used in the winning solution to the challenge. We discuss this step more later, since without further acceleration it presents the biggest computational bottleneck.

The third step of the iterative procedure is to find the convex combination of $P_t$ and $Q_t$ that maximizes the score in eq. (2). In particular, the update is given by

$$\alpha_t = \operatorname*{argmax}_{\alpha\in[0,1]} \Big[\mathcal{S}\big((1-\alpha)P_t + \alpha Q_t\big)\Big], \tag{10}$$

$$P_{t+1} = (1-\alpha_t)P_t + \alpha_t Q_t. \tag{11}$$

In practice, it is not necessary to perform a line search to compute the optimal convex combination in eq. (10). Instead one can simply calculate the point where the gradient of the score vanishes along the line connecting $P_t$ and $Q_t$. Since the score in eq. (2) is quadratic in its argument, this gradient vanishes at some point $(1-\lambda)P_t + \lambda Q_t$ where $\lambda\in\mathbb{R}$. In particular, $\lambda$ satisfies the linear equation

$$(1-\lambda)\,\mathrm{trace}\big[(Q_t-P_t)^\top \nabla\mathcal{S}(P_t)\big] = \lambda\,\mathrm{trace}\big[(Q_t-P_t)^\top \nabla\mathcal{S}(Q_t)\big]. \tag{12}$$

If $\lambda\in[0,1]$, then the weight $\alpha_t$ in eq. (10) is simply equal to $\lambda$. If $\lambda\notin[0,1]$, then there are two possibilities: either the score along the line from $P_t$ to $Q_t$ is *concave* with a *maximum* at $\lambda > 1$, or it is *convex* with a *minimum* at $\lambda < 0$. In both these cases, eq. (10) yields $\alpha_t = 1$. Finally, we note that the updates in eqs. (9–11) converge monotonically to a doubly stochastic matrix that is a stationary point (where the gradient has no component inside $\mathcal{P}$) of this procedure.

## 3.2 Application to graph-matching

While the Frank-Wolfe updates lead to monotonic improvement in the score of eq. (2), they converge in general to a doubly stochastic matrix and not a permutation matrix. But it is the latter that is needed to align two graphs with a score given by eq. (1). To rectify this problem, we also compute a permutation matrix $\Pi_t$ at each iteration of the updates in eqs. (9–11). This is done by projecting the doubly stochastic matrix $P_t$ into the space of permutation matrices:

$$\Pi_t = \operatorname*{argmax}_{\Pi \in \mathcal{P}} \left( \mathrm{trace}\left[ P_t^\top \Pi \right] \right). \tag{13}$$

Eq. (13) is a linear program whose solution is the closest-matching permutation matrix to $P_t$. Again this can be solved by the Hungarian method or any other algorithm for perfect matching in a complete bipartite graph. In practice the linear program in eq. (13) is much faster to solve than the one in eq. (9); the reason is that the doubly stochastic matrix $P_t$ in eq. (13) is highly sparse—expressible as a convex combination of a small number of permutation matrices—whereas the gradient $\nabla\mathcal{S}(P_t)$ in eq. (9) is dense.

Since the updates for $P_t$ in eqs. (9–11) converge to a point inside the convex set of doubly stochastic matrices, it is also true that their projections to $\Pi_t$ in eq. (13) converge to a permutation matrix at a vertex of this set. But while the scores $\{\mathcal{S}(P_t)\}_{t=0}^{T}$ of these doubly stochastic matrices increase monotonically as a result of these updates, the same is *not* true for the scores $\{\mathcal{S}(\Pi_t)\}_{t=0}^{T}$ of their closest-matching permutation matrices. The right panel of Fig. 1 plots the scores from these updates starting from the benchmark solution with score 5154247.

From the results in Fig. 1, we make several observations of interest. First, at the outset of the optimization, the continuous updates in the convex set of doubly stochastic matrices (shown right) increase the score much more rapidly than the discrete search based on pairwise swaps (shown left). Second, the scores of the permutation matrices $\Pi_t$ in eq. (13) generally track the scores of the doubly stochastic matrices $P_t$ in eq. (11), but the latter increase monotonically while the former do not. Third, the scores of the doubly stochastic matrices $P_t$ saturate around 5856K, while those of the permutation matrices $\Pi_t$ saturate just below 5850K. In particular, these updates by themselves do not obtain a handily winning score for the challenge.

## 3.3 Acceleration by preconditioning

In this section we describe how the winning entry to the challenge solved the linear program in eq. (9). As mentioned previously, this problem is equivalent to one of perfect matching, and it is more typically posed in terms of a cost matrix $C \in \mathbb{R}^{n\times n}$, where the goal is to find the permutation $\pi$ that minimizes the linear assignment cost

$$\mathrm{trace}(C^\top P^\pi) = \sum_i C_{i\pi_i}. \tag{14}$$

There is an internal (though not especially well-documented) routine in MATLAB that solves this problem by permuting large entries to the diagonal of a sparse matrix (Duff & Koster, 2001). It assumes that $C$ is stored as a dense matrix, and it is called as

$$\pi = \texttt{matlab.internal.graph.perfectMatching}(C). \tag{15}$$

We used this internal routine to solve the linear programs in eqs. (9) and (13) whose cost matrices had $n = 18524$ rows and columns. The routine is based on a polynomial-time algorithm, but it can be very slow if called in the above manner when $C$ is a dense matrix. For example, when $C = -\nabla\mathcal{S}(P_t)$, this routine requires 10-15 minutes per call on a MacBook Pro (M1 Max) with 64 GB of RAM. Of course it would not be possible to obtain a winning solution in less than 15 minutes if each iteration of Algorithm 3 required this much computation.

We discovered a heuristic that greatly accelerates this routine for perfect matching when it is called with the gradients $\nabla\mathcal{S}(P_t)$ that appear in eq. (9). The heuristic is based on three observations. First, the result in eq. (15) is unaffected if we shift any row or column of the cost matrix by a constant value. Second, the result is trivially equal to the identity permutation if $C$ has negative elements on the diagonal and nonnegative elements off the diagonal. Third, suppose that an approximate solution $\omega$ can be be guessed for eq. (15), where $\omega$ is a permutation that nearly solves the linear assignment problem. Then the matrix product $C \cdot (P^\omega)^\top$ should be closer than $C$ to a matrix whose smallest entries appear on the diagonal.

Based on these observations, we discovered something akin to a preconditioner for the routine in eq. (15) when $C = -\nabla\mathcal{S}(P_t)$. We describe this preconditioner in detail because it yielded a significant speedup, reducing the time per call by a factor of 50-60x, or from minutes to seconds. As shorthand, let $\mathbb{1} \in \mathbb{R}^n$ denote the column vector of all ones, and let diag$(\cdot)$ denote the column vector of diagonal elements from its matrix argument. We start by observing that $\Pi_t$ in eq. (13) provides an approximate guess for $Q_t$ in eq. (9). With this and the previous observations in mind, we solve eq. (9) in the following way:

$$\text{(PERMUTE)} \qquad \Lambda = \nabla \mathcal{S}(P_t)\Pi_t^\top, \tag{16}$$

$$\text{(SHIFT)} \qquad \Omega = \Lambda + \mathrm{diag}(\Lambda)\mathbb{1}^\top + \mathbb{1}\,\mathrm{diag}(\Lambda)^\top - \mathbb{1}\mathbb{1}^\top\Lambda - \Lambda\mathbb{1}\mathbb{1}^\top, \tag{17}$$

$$\text{(MATCH)} \qquad \omega = \texttt{matlab.internal.graph.perfectMatching}(-\Omega), \tag{18}$$

$$\text{(UNPERMUTE)} \qquad Q_t = P^\omega \Pi_t. \tag{19}$$

Intuitively, the first of these steps (PERMUTE) ensures that $\Lambda$ has positive elements on the diagonal, the second (SHIFT) makes it more likely that $\Omega$ has negative elements off the diagonal, and the third (MATCH) is fastest when $\Omega$ has positive elements on the diagonal and none elsewhere, in which case $\omega$ is close to the identity permutation. We do not have a formal justification for this heuristic, but in practice it was essential, removing eq. (9) as the main bottleneck in Algorithm 3.

## 4 A winning solution (and beyond)

It is possible to combine the methods for search in the last two sections and reap the advantages of both. The discrete swaps in section 2 lead to slow but steady improvement until they reach a local maximum from which they cannot escape. The continuous Frank-Wolfe updates in section 3 lead to rapid improvement in the alignment score, but they plateau when the high-scoring doubly stochastic matrices in eq. (2) do not project to high-scoring permutation matrices in eq. (1). A winning solution can be quickly obtained by alternating these approaches, using each to offset the weaknesses of the other. This is the method of alternating continuous and discrete combinatorial (AC$\oplus$DC) search.

### 4.1 AC$\oplus$DC search

Fig. 2 shows the results from this alternating approach. First, we use ten Frank-Wolfe updates to climb from the benchmark score at 5154K to a score above 5845K in less than 10 minutes. Then we apply pairwise swaps until the score can no longer be further improved; in five additional minutes, these swaps produce a solution whose score exceeds 5850K, higher than all but the winning entry to the challenge. As shown in the figure, the score can be further improved by alternating these different types of search, with the Frank-Wolfe updates jumping out of the local maximum reached by the pairwise swaps, and the pairwise swaps reaching higher scores from wherever they are subsequently initialized. This combined approach reaches a score over 5852K in under one hour. In the supplementary material, we provide MATLAB code (less than 400 lines) to obtain this winning solution, as well as code to generate all the figures in this paper.

Fig. 2 also highlights the different role played by the continuous Frank-Wolfe updates in the later stages of optimization. In the first few iterations, before the five-minute mark, there is a high degree of correlation between the scores of the doubly stochastic matrices in eq. (11) and their closest-matching permutation matrices in eq. (13): when the former increase (shown in blue), so do the latter (shown in red). But this relationship no longer holds past the five-minute mark in Fig. (2). In this regime, we see that higher-scoring interior solutions often project to lower-scoring permutation matrices. Nevertheless these continuous updates still play a crucial role: they re-initialize the next stage of discrete updates in a basin of attraction where pairwise swaps can reach a higher maximum of the score in eq. (1). The overall result is the seesaw pattern of improvement between the red and yellow curves that we see in Fig. (2).

### 4.2 Further improvements

For the VNC matching challenge, we have shown that a score of over 5852K can be reached in under one hour by combining simple methods for discrete and continuous search. In this section, we give a brief overview of additional methods to further improve the score. At the outset, we note that above 5852K the optimization

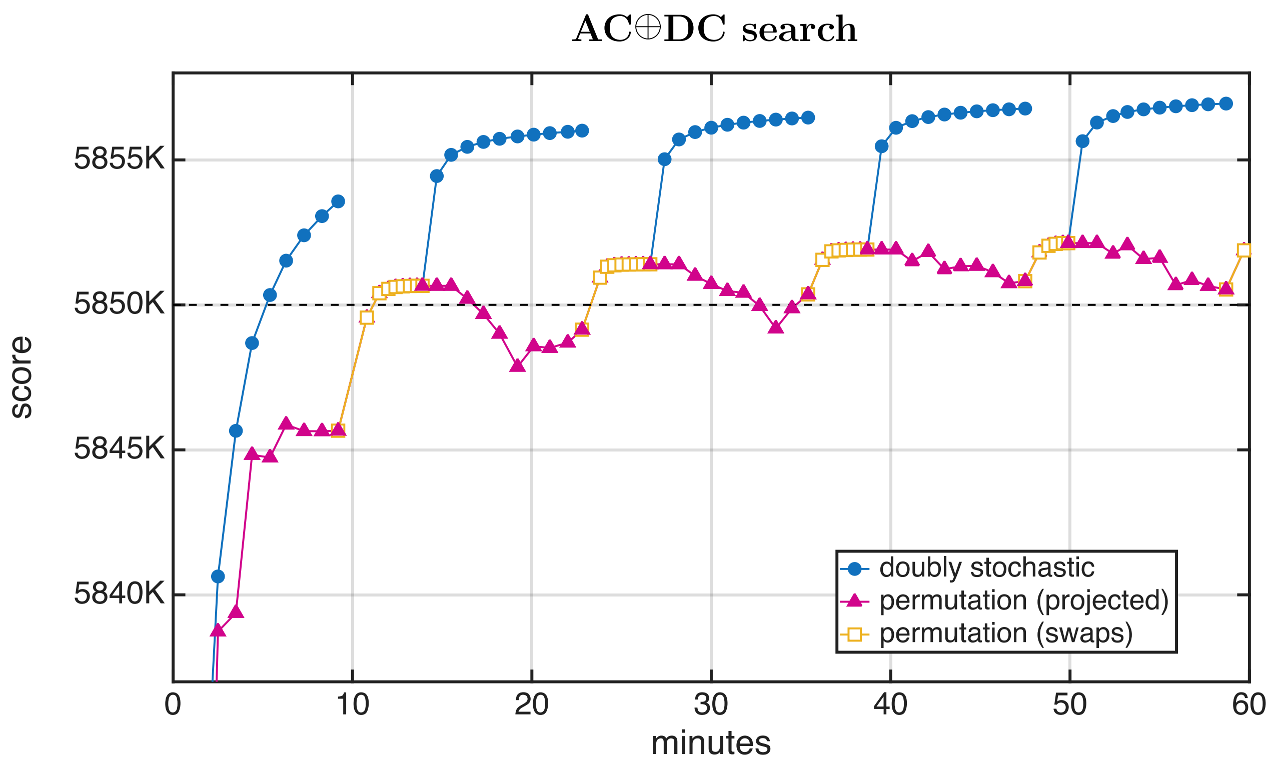

Figure 2: Alignment scores in eqs. (1–2) versus wall clock time starting from the benchmark solution at score 5154247. The scores were obtained by alternating updates for continuous and discrete combinatorial (AC⊕DC) search—in particular, Frank-Wolfe updates (in batches of ten) for the former and greedy pairwise swaps (repeated until no further swaps improved the score) for the latter. It takes less than 15 minutes for this method to produce a winning score for the challenge (indicated by the dashed line at 5850K).

appears to enter a regime of diminishing returns. As shown in Fig. 2, it takes only a few minutes to improve the benchmark score by nearly 700K, and then another hour after that to improve the score by an additional 10K. But beyond this regime it takes many additional hours—even for the more elaborate methods we discuss next—to obtain improvements that are orders-of-magnitude less. In light of this, we only provide a high-level sketch of these methods.

### 4.2.1 Higher-order swaps

The discrete search in Algorithm 2 quickly finds a solution that cannot be improved by further pairwise swaps. This search over permutation matrices can be extended by considering higher-order swaps that permute more than two indices at a time. For higher-order swaps, however, it is no longer feasible to evaluate all possible local moves before considering which ones to perform; there are, for instance, over one trillion different three-node swaps that can be performed in a graph with $n = 18524$ nodes. Instead one can evaluate a subset of higher-order moves that seem most likely to yield improvements. For example, we considered the subset of three-cycles $\{(i \to j \to k \to i)\}$ where the index $k$ was chosen greedily for all pairwise swaps $\{(i \leftrightarrow j)\}$ that did not reduce the score by a certain threshold. We also devised similar strategies for considering many different types of higher-order swaps. In total, our most sophisticated discrete search considered not only pairwise swaps, but also 3-cycles, 4-cycles, and 5-cycles, as well as 2x2, 3x2, 3x3, 4x2, 5x2, 2x2x2, 3x2x2, and 2x2x2x2 swaps. With these higher-order swaps, it takes another dozen hours to boost the score from 5852K to 5853K (amounting to a gain of less than 0.01%).

### 4.2.2 Multiplicative updates

The Frank-Wolfe updates in Algorithm 3 produce a sequence of doubly stochastic matrices that improve the score in eq. (2). When these updates are initialized from a permutation matrix, they produce a sequence of *sparse* doubly stochastic matrices. This sparsity has certain computational advantages: for example, it can be exploited to compute the gradient in eq. (4) much more efficiently. But it also has potential disadvantages; in particular, an optimization restricted to sparse solutions may not fully leverage the continuous search that is afforded by the relaxation to doubly stochastic matrices.

Recall that the updates in eq. (11) are *additive* updates in which the existing solution $P_t$ is linearly interpolated with the projected gradient $Q_t$. We also experimented with *multiplicative updates* that use the gradient in eq. (4) quite differently. These updates take the form

$$[P_{t+1}]_{ij} = [P_t]_{ij} \cdot \frac{[\nabla \mathcal{S}(P_t)]_{ij}}{u_i + v_j}, \tag{20}$$

where in the numerator of eq. (20) appear the elements of the gradient $\nabla\mathcal{S}(P_t)$ and in the denominator appear Lagrange multipliers $u, v \in \mathbb{R}^n$. This multiplicative update can be derived as a generalization of those for nonnegative and (singly) stochastic matrix factorization (Lee & Seung, 1999; Saul & Pereira, 1997). The main generalization is to introduce two sets of Lagrange multipliers into the update; one of these is to enforce sum-to-one constraints on the rows of doubly stochastic matrices, and the other is to enforce sum-to-one constraints on the columns. The resulting update is similar but not equivalent to the Sinkhorn-Knopp procedure for projecting a nonnegative matrix onto the set of doubly stochastic matrices (Sinkhorn & Knopp, 1967).

The multiplicative updates in eq. (20) can be used to optimize the score in eq. (2), and unlike the Frank-Wolfe updates, they do not involve the expense of computing a projected gradient, as in eq. (9). But to use these updates on dense doubly stochastic matrices, it is necessary to compute the score in eq. (2) and the gradient in eq. (4) when $P$ is dense. Naively this appears to require $O(n^4)$ operations, a prohibitive scaling for matrices of size $n = 18524$.

We devised a faster way to compute these gradients by exploiting the fact that the connectome weights are *quantized*. In particular, each nonzero weight records a positive number of synapses, and therefore not only are the elements of $A$ and $B$ quantized, but so are the possible values of $\min(A_{ij}, B_{kl})$ in eq. (4). Let $\mathcal{Q} = \{q_0, q_1, \ldots, q_M\}$ denote the set of these quantized values, with $q_0 = 0$ and $q_i < q_{i+1}$, and let $\Theta(\cdot)$ denote the step function defined by $\Theta(z) = 1$ if $z > 0$ and $\Theta(z) = 0$ otherwise. Then it follows that

$$\min(A_{ij}, B_{kl}) = \sum_{m=0}^{M-1} \Theta(A_{ij} - q_m)\, \Theta(B_{kl} - q_m)\, (q_{m+1} - q_m) \tag{21}$$

for all connectome weights $A_{ij}$ and $B_{kl}$. Note how this identity expresses the minimum as a sum over $M$ components. We now use this identity to more efficiently compute the score in eq. (2) and the gradient in eq. (4). To do so, for each interval $(q_m, q_{m+1})$, we define connectome *components* with weights

$$A_{ij}^{(m)} = \Theta(A_{ij} - q_m)\, \sqrt{q_{m+1} - q_m}, \tag{22}$$

$$B_{ij}^{(m)} = \Theta(B_{ij} - q_m)\, \sqrt{q_{m+1} - q_m}. \tag{23}$$

Note that each connectome component is a sparse matrix in its own right, one that is at least as sparse as the connectome from which it is derived. Finally, combining eqs. (21–23), we rewrite the score in eq. (2) as

$$\begin{aligned}
\mathcal{S}(P) &= \sum_{ijkl} \min(A_{ij}, B_{kl}) P_{ik} P_{jl} && (24)\\
&= \sum_{ijkl} \left[ \sum_m A_{ij}^{(m)} B_{kl}^{(m)} \right] P_{ik} P_{jl} && (25)\\
&= \sum_m \operatorname{trace} \left[ \left( A^{(m)} P \right) \left( P B^{(m)} \right)^\top \right]. && (26)
\end{aligned}$$

| Submitted | Name | Score |
|---|---|---|
| 2025-02-18 | [redacted] | 5,853,925 |
| 2025-02-17 | D. A. Bader, H. A. Sriram, S. Chinthalapudi, and Z. Du | 5,853,910 |
| 2025-01-31 | [redacted] *(Winner)* | 5,853,779 |
| 2025-01-31 | D. A. Bader, H. A. Sriram, S. Chinthalapudi, and Z. Du | 5,849,534 |
| 2025-01-31 | Y. Ma, X. Zhu, and L. Zhu | 5,842,347 |
| 2025-01-31 | W. B. Hayes, M. Longo, and R. Longo | 5,841,041 |
| 2025-01-31 | Team FAQ | 5,838,188 |
| 2025-01-31 | D. Hashorva | 5,837,872 |
| 2025-01-31 | P. J. C. Duarte, C. Larsen, and R. Willemsen | 5,834,246 |
| 2025-01-31 | T. M. da Nóbrega | 5,824,339 |

Table 1: Top ten scores on the leaderboard of the VNC matching challenge as of the writing of this paper. The winning score on 2025-01-31 was obtained using the methods in this paper, and the top score on 2025-02-18 was obtained by combining the methods of the two leading teams.

Note that this final expression for the score in eq. (26) can be computed in $O(Mn^3)$ as opposed to $O(n^4)$. This savings is significant when $M \ll n$, and it is also inherited by the computation of the gradient. For the VNC matching challenge, there are $M = 617$ graph components that arise from the nonzero connectome weights of the male and female fruit fly. With this savings, and using a GPU, it takes less than 2 minutes to compute the gradient of eq. (26) and perform each multiplicative update in eq. (20).

The official winning score to the challenge was 5853779. This score was submitted on January 31, 2025 and achieved by alternating the additive updates for sparse doubly stochastic matrices in eq. (11) with the multiplicative updates for dense doubly stochastic matrices in eq. (20). A score of 5853925, higher by 0.0025%, was obtained on February 18, 2025 by combining the methods of the two top-scoring teams. Table 1 reproduces the top ten scores on the leaderboard as of the writing of this paper.

## 5 Discussion

In this paper we have described the AC⊕DC optimizations behind the winning solution to the VNC Matching Challenge. The graphs in this challenge were large enough to foil exhaustive methods, but small enough to experiment with many different approaches on a modestly equipped computer. The highest-scoring solution was obtained by combining continuous relaxations, additive and multiplicative updates, bespoke graph decompositions, and higher-order swaps. But most of the work was done by alternating simple (but aggressively optimized) methods for continuous and discrete combinatorial search and exploiting the particular structure of the alignment score in eq. (1).

We mention several directions for future work. First, not all of the methods in this paper scale gracefully to larger graphs with $n \gg 10^4$ nodes. For such graphs, it seems necessary to develop divide-and-conquer methods that do not require the storage of $n \times n$ matrices. Second, we expect the linear assignment problem in eq. (9) to remain a crucial subroutine for higher-order assignment problems (or at least for any problem whose score function can be linearized). We need to understand better why certain heuristics, such as the preconditioner in Section 3.3, lead to faster solutions, and then perhaps we can use this understanding to develop even faster approaches. Third, the winning solution to the VNC matching challenge was implemented in MATLAB, a relatively high-level programming language, but it is surely possible to produce faster implementations that are better at exploiting sparsity, managing high-speed memory, and harnessing GPUs. Indeed, to solve larger problems in graph matching, we are likely to need further progress in all of these directions.

### Acknowledgements

[redacted]

## References

L. Chen, J. T. Vogelstein, V. Lyzinski, and C. E. Priebe. A joint graph inference case study: the c. elegans chemical and electrical connectomes. *Worm*, 5(2):e1142041, 2016.

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*, 18:265–298, 2004.

A. V. Devineni. A complete wiring diagram of the fruit-fly brain. *Nature*, 634:35–36, 2024.

S. Dorkenwald, A. Matsliah, A. R. Sterling, P. Schlegel, S.-C. Yu, C. E. McKellar, A. Lin, M. Costa, K. Eichler, Y. Yin, W. Silversmith, C. Schneider-Mizell, C. S. Jordan, D. Brittain, A. Halageri, K. Kuehner, O. Ogedengbe, R. Morey, J. Gager, K. Kruk, E. Perlman, R. Yang, D. Deutsch, D. Bland, M. Sorek, R. Lu, T. Macrina, K. Lee, J. A. Bae, S. Mu, B. Nehoran, E. Mitchell, S. Popovych, J. Wu, Z. Jia, M. A. Castro, N. Kemnitz, D. Ih, A. S. Bates, N. Eckstein, J. Funke, F. Collman, D. D. Bock, G. S. X. E. Jefferis, H. S. Seung, M. Murthy, and The Fly Wire Consortium. Neuronal wiring diagram of an adult brain. *Nature*, 634:124–138, 2024.

I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM Journal on Matrix Analysis and Applications*, 22(4):973–996, 2001.

J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972.

M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956.

S. Haller, L. Feineis, L. Hutschenreiter, F. Bernard, C. Rother, D. Kainmüller, P. Swoboda, and B. Savchynskyy. A comparative study of graph matching algorithms in computer vision. In *Proceedings of the 17th European Conference on Computer Vision (ECCV-2002)*, pp. 636–653, 2022.

E. Koblentz. Fruit fly research led NJIT scientists and Edison teens to better AI habits on supercomputers. `https://davidbader.net/post/20250303-njit/`, 2025. Accessed: 2025-06-11.

H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2: 83–97, 1955.

D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401: 788–791, 1999.

N. Mamano and W. B. Hayes. Simulated annealing far outperforms many other search algorithms for biological network alignment. *Bioinformatics*, 33(14):2156–2164, 2017.

J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

L. Saul and F. Pereira. Aggregate and mixed-order markov models for statistical language processing. In *Proceedings of the 2nd Conference on Empirical Methods in Natural Language Processing*, pp. 81–89, 1997.

R. Sinkhorn and P. Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21:343–348, 1967.

N. Tomizawa. On some techniques useful for solution of transportation network problems. *Networks*, 1(2): 173–194, 1971.

L. R. Varshney, B. L. Chen, E. Paniagua, D. H. Hall, and D. B. Chklovskii. Structural properties of the caenorhabditis elegans neuronal network. *PLoS Computational Biology*, 7(2):e1001066, 2011.

J. T. Vogelstein, J. M. Conroy, V. Lyzinski, L. J. Podrazik, S. G. Kratzer, E. T. Harley, D. E. Fishkind, R. J. Vogelstein, and C. E. Priebe. Fast approximate quadratic programming for graph matching. *PLOS ONE*, 10(4):1–17, 2015.