A. Omitted Proofs

Lemma 2.1. $\|\phi(x) - \phi(y)\|_F \leq 2\|x - y\|_1$.

Proof. It suffices to prove that for doubly-stochastic matrices U and V, we have $\|M^k(x_k)U - M^k(y_k)V\|_F \leq \|U - V\|_F + 2|x_k - y_k|$. The result follows when we set U and V to $M^{k-1}(x_{k-1}) \dots M^1(x_1)$ and $M^{k-1}(y_{k-1}) \dots M^1(y_1)$, respectively.

$$\begin{aligned}
&\|M^k(x_k)U - M^k(y_k)V\|_F \\
=&\|M^k(x_k)U - M^k(x_k)V + M^k(x_k)V - M^k(y_k)V)\|_F \\
\leq&\|M^k(x_k)(U - V)\|_F + \|(M^k(x_k) - M^k(y_k))V\|_F \\
\leq&\|U - V\|_F + \|(M^k(x_k)V - M^k(y_k))V\|_F. \qquad (19)
\end{aligned}$$

The first inequality uses the triangle inequality and the second inequality follows from the fact that M^k is a contraction. By using $V_{s,:}$ and $V_{t,:}$ to denote the s_kth and t_kth rows of V, we obtain

$$\begin{aligned}
&\|(M^k(x_k)V - M^k(y_k))V\|_F^2 \\
&= 2\|(x_k - y_k)V_{s,:} - (x_k - y_k)V_{t,:}\|_F^2 \\
&= 2(x_k - y_k)^2\|V_{s,:} - V_{t,:}\|_F^2 \\
&\leq 2(x_k - y_k)^2(\|V_{s,:}\|_F^2 + \|V_{t,:}\|_F^2) \\
&\leq 4(x_k - y_k)^2, \qquad (20)
\end{aligned}$$

where the final equality uses the fact that the ℓ_2-norm of each row of a doubly-stochastic matrix is at most 1. The claim follows from combining (19) and (20). □

Proposition 2.2. *Suppose the objective $g(x;\mu)$ in Problem (6) is coordinatewise strongly concave. Then any point $\hat{x}$ that is coordinatewise minimal leads to a permutation matrix $\phi(\hat{x})$. Any $\phi(x^*)$ corresponding to a global minimum x^* is a solution to the original Problem (1).*

Proof. Any x that is not binary is not coordinatewise minimal, so all coordinatewise minimal points must be binary and correspond to a permutation matrix. Also, the regularization term penalizes every permutation equally and hence a global minimal x^* gives an optimal permutation $\phi(x^*)$ for Problem (1). □

Proposition 3.1. *Suppose $g(\cdot;\mu)$ is coordinatewise strongly concave. Then cyclic coordinate descent with exact minimization reaches a binary x in finitely many steps.*

Proof. The minimizer along each coordinate is always either 0 or 1 so within a single cycle of coordinate descent, x is binary. After that, x remains binary and at each step corresponds to an unvisited permutation since $f(x)$ is monotonically decreasing. Hence, the algorithm terminates within $n! + m$ coordinate iterations. □

B. Gradient and Hessian Computation.

In describing the computation of the full gradient, we focus on the term (13), which is one of the terms in the partial derivative with respect to x_k. We can rearrange this term to obtain

$$\left\langle AM^m \dots M^{k+1}v^k, \phi(x)BM^1 \dots M^{k-1}v^k \right\rangle. \qquad (21)$$

We can maintain matrices $U^k := AM^mM^{m-1} \dots M^{k+1}$ and $V^k := \phi(x)BM^1M^2 \dots M^{k-1}$, so that (21) can be written as $\langle U^kv^k, V^kv^k\rangle$. The derivative with respect to the next component x_{k+1} can be written as $\langle U^{k+1}v^{k+1}, V^{k+1}v^{k+1}\rangle$, where the new W and Z matrices are obtained from the $O(n)$ update formulas $U^{k+1} = U^k(M^{k+1})^{-1}$ and $V^{k+1} = V^kM^k$.

If $x_k = 0.5$, we have $M^k(x_k)$ is not invertible. As for cyclic coordinate descent, we can store the two columns that were affected by $M^k(x_k)$ and restore them when needed.

Algorithm 3 details the process for incrementally computing term (13) for all x_k. (The process for computing (14) is similar.) Computation of the full gradient is thus also an $O(nm)$ operation. Using this technique, we can apply full-gradient first-order methods efficiently, including gradient projection and Frank-Wolfe. With an appropriate line-search method, gradient projection is guaranteed to converge to a stationary point.

Algorithm 3 Computing the gradient term (13) for all x_k

```
Input: comparison coefficients $x$ and matrices $A, B \in \mathbb{R}^{n\times n}$
$U \leftarrow A$
for $i = m$ to $1$ do
    $U \leftarrow UM^i$
end for
$V \leftarrow$ compute $\phi(x)B$
for $i = 1$ to $m$ do
    $U \leftarrow U(M^i)^{-1}$
    $g_i \leftarrow \langle Uv^i, Vv^i\rangle$
    $V \leftarrow VM^i$
end for
```

Theorem B.1. *The gradient with respect to a sorting-network coordinate for the objective in (12) can be computed in $O(nm)$ time. The amount of additional memory used is $O(n^2 + nd)$, where $d = |\{i \mid x_i = 0.5\}|$.*

For the Hessian computation, the second partial derivative with respect to x_k and x_j is

$$\begin{aligned}
2 \cdot \Big\langle &(v^j)^\top M^{j+1} \dots M^m AM^m \dots M^{k+1}v^k, \\
&(v^k)^\top M^{k-1} \dots M^1 BM^1 \dots M^{j-1}v^j \Big\rangle. \qquad (22)
\end{aligned}$$

Using matrices $U^k := AM^m M^{m-1} \dots M^{k+1}$ and $W^k := M^{k-1}M^{k-1} \dots M^1 B$, so that (22) can be written as $\langle (v^j)^\top M^{j+1} \dots M^m U^k v^k, (v^k)^\top W^k M^1 \dots M^{j-1} v^j \rangle$, we can compute U^k and W^k from U^{k-1} and W^{k-1} respectively in $O(n)$ operations.

We can compute the kth row of the Hessian in $O(m)$ operations, given U^k and W^k. We first compute $u^{k,0} := M^1 M^2 \dots M^m U^k v^k$ and $w^{k,0} := (v^k)^\top W^k$, and recursively define the terms $u^{k,j} := (M^j)^{-1} u^{k,j-1}$ and $w^{k,j} = w^{k,j-1} M^j$. The (k, j) entry of the Hessian is now given by

$$2((v^j)^\top u^{k,j})(w^{k,j} v^j).$$

Computing each entry of the Hessian takes just $O(1)$ operations, due to the sparsity of the v vectors. As with the cyclic coordinate descent algorithm and gradient computation, when $x_k = 0.5$, we need to store the corresponding columns or elements to reverse $M^k(x_k)$.

Algorithm 4 describes this process in detail.

Algorithm 4 Computing the Hessian

```
Input: comparison coefficients $x$ and matrices $A, B \in \mathbb{R}^{n \times n}$
$U \leftarrow A$
for $i = m$ to $1$ do
  $U \leftarrow UM^i$
end for                    // $U = A\phi(x)$ at the end of the loop
$W \leftarrow B$
for $i = 1$ to $m$ do
  $U \leftarrow U(M^i)^{-1}$
  $u \leftarrow Uv^i$
  $w^\top \leftarrow (v^i)^\top W$
  for $j = m$ to $1$ do
    $u \leftarrow M^j u$
  end for
  for $j = 1$ to $m$ do
    $u \leftarrow (M^j)^{-1} u$
    $H_{kj} \leftarrow ((v^j)^\top x) \cdot (w^\top v^j)$
    $w^\top \leftarrow w^\top M^j$
  end for
  $W \leftarrow M^i W$
end for
```

Theorem B.2. *The Hessian can be computed in $O(m^2)$ time and uses $O(n^2 + nd)$ additional memory (if we do not need to store the Hessian), where $d = |\{i \mid x_i = 0.5\}|$.*

C. Additional Experiment Details

C.1. QAPLIB Experiments: Details

We now assess the effects of the various features and heuristics in our coordinate descent framework. In this section, we use CD to refer to the variant of coordinate descent with no continuation and a fixed regularization term of $\mu = 0$. The timings in this section are relative to the average performance of CD.

We use Table 2 to study the effect of the type of sorting network in the sorting network heuristic for local optimality, and Table 3 to study the effects of continuation and of the greedy pairwise swap heuristic.

From Table 2, we see that in general, using the $O(n \log^2 n)$ sized sorting network (RS) gives a significant improvement over having no sorting network, with the exception of `els` (this is likely due to the randomness inherent to the experiments). The full sorting network (FS) gives a slight improvement over RS in a majority of instances, especially in terms of the median objective. The running time of RS and FS are similar, and they are slightly higher than the running time for having no sorting network.

From the first part of Table 3, in terms of the objective value we see that applying just continuation (CD+C) is generally better than having just running CD, while applying continuation in the presence of a sorting network has mixed effects. We get mixed results when we compare the timing incurred by using just continuation (CD+C) with the timing from just using the sorting network heuristic (CD+FS).

The results from applying the greedy swap heuristic (second part of Table 3) show that applying the greedy swap heuristic can generally improve performance, at a much larger cost in terms of timing compared to the sorting network heuristic. Applying the greedy heuristic on the methods with the sorting network heuristic (i.e. CD+FS+G and CD+C+FS+G) can actually lead to a method that is faster that the methods without (i.e. CD+G and CD+C+G) on some problem families like `tho` and `sko`. This is in line with the fact that the sorting network heuristic can be viewed a continuous variant of the greedy swap heuristic and as such replaces some of the work the greedy swap heuristic can do. In general, greedy swaps can significantly improve the objective value obtained by the methods. Note that this results in a much smaller performance gap between CD+G and CD+C+FS+G than between plain CD and CD+C+FS. The difference between CD+C+FS and CD+C+FS+G is generally small.

Finally, we include the table of results for all the FAQ variants (Table 4) to better understand the effect of applying the greedy pairwise swap heuristic to FAQ. For problems like `bur`, `chr`, `els`, and `scr`, we see that the greedy heuristic is the primary factor behind the final objective values obtained, while for most other instances it helps to narrow the gap between FAQ10 and FAQ30.

			Average Percent Optimality Gap						Average Timing Ratio (compared to CD)		
			Best of 100 Runs			Median of 100 Runs					
Type	Num	n	CD+C	CD+C+RS	CD+C+FS	CD+C	CD+C+RS	CD+C+FS	CD+C	CD+C+RS	CD+C+FS
BUR	8	26	.06	.12	.03	.78	.78	.79	1.03	1.01	.97
CHR	14	12--25	11.92	6.84	7.34	44.50	39.92	38.48	1.92	2.57	2.20
ELS	1	19	.90	4.21	.00	20.59	27.60	27.60	1.93	2.73	2.55
ESC	20	16--128	.73	.08	.15	3.93	2.02	1.87	3.75	4.67	4.69
HAD	5	12--20	.00	.00	.00	.59	.39	.30	2.37	3.44	3.27
KRA	3	30--32	2.23	1.75	1.72	1.67	1.23	1.12	1.57	2.17	2.36
LIPA A	8	20--90	1.06	.95	.89	1.51	1.50	1.50	5.34	5.37	5.63
LIPA B	8	20--90	12.47	7.92	7.52	19.52	19.03	18.86	4.66	4.69	4.79
NUG	15	12--30	.77	.43	.39	4.29	3.46	3.39	1.82	2.49	2.16
ROU	3	12--20	1.01	.22	.31	5.33	4.67	4.67	3.06	3.15	2.81
SCR	3	12--20	1.24	.00	.00	7.60	7.04	6.62	2.31	2.70	2.43
SKO	13	42--100	1.30	1.03	.86	2.47	2.18	2.02	1.39	1.84	2.15
STE	3	36	4.63	3.63	4.69	16.93	14.77	13.85	1.42	1.87	1.99
TAI	28	10--256	1.51	1.30	1.31	6.09	5.50	5.42	2.37	2.91	2.90
THO	3	30--150	1.74	1.54	1.20	4.03	3.62	3.36	1.33	1.87	1.95
WIL	2	50,100	.45	.48	.39	1.35	1.20	1.13	1.35	1.83	2.11

Table 2. Performance of CD+C, CD+C+RS, and CD+C+FS on QAPLIB instances, aggregated by family of instances. These variants are compared to highlight the differences with modifying the choice of the sorting network for the local optimality heuristic.

C.2. Comparison To Other Methods for QAPLIB

For completeness, we include a discussion about the other methods to which FAQ has been compared in Vogelstein et al. (2015). These methods include

- PATH (Zaslavskiy et al., 2009), a convex-concave method for undirected graph matching that works by first solving the convex relaxation of the graph matching problem, then solving linear interpolations of convex and concave formulations of the graph matching problem to move towards a permutation matrix.
- EPATH (Liu et al., 2012), a directed-graph version of PATH.
- QPB (quadratic programming bound) (Anstreicher & Brixius, 2001; Schellewald et al., 2001), which solves a convex quadratic relaxation of the QAP and then formulates a linear assignment problem to find a permutation matrix.
- GRAD (graduated assignment algorithm) by Gold & Rangarajan (1996), an annealing-type approach.
- U (Umeyama, 1988)), which formulates the graph matching problem as a linear assignment problem using spectral information.

For QAPLIB, the majority of results reported for these algorithms has been on either a subset of 16 symmetric or 16 asymmetric instances. The median results of CD over 100 instances outperforms the results of the other methods from Zaslavskiy et al. (2009) (namely, PATH, QPB, GRAD, and U). Relative to the top quartile of results for FAQ10 and FAQ30 outperforms PATH for 13 and all 16 of the problems respectively. The asymmetric instances include 8 `lipa a` and 8 `lipa b` instances, and results for QPB, GRAD, EPATH, and U are reported in Liu et al. (2012). The top quartile for the FAQ methods outperform or tie with the other algorithms on all 16 instances. CD outperforms these algorithms on the `lipa a` instances, but for `lipa b`, GRAD and EPATH perform as well as the FAQ methods and significantly better than CD, which in turn significantly outperform QPB and U.

C.3. Synthetic QAP Experiments: Details

QAP generator and parameters. The class of problems come from Section 3.5 of Taillard (1995).

We first generate n points on the plane. We repeat the following process until there are no points left to place:

- Pick a cluster size s uniformly between 1 and 20.
- Pick a radius uniformly between 0 and 1000 and an angle uniformly at random. The corresponding point gives us the center of the cluster.
- Place s points around the center of the cluster by picking a radius uniformly between 0 and 10 and an angle uniformly at random for each point.

We let the A matrix be the Euclidean distance matrix corresponding to these n points. For the B matrix, we first generate a uniform random $[0, 1]$ real $n \times n$ matrix X. We then set the (i, j) component of B to be $\lfloor 10^{(6X_{ij}-4)} \rfloor$. Under this scheme, roughly two-thirds of the entries of B are zero.

			Average Percent Optimality Gap								Average Timing Ratio		
			Best of 100 Runs				Median of 100 Runs				(compared to CD)		
Type	Num	n	CD	CD+C	CD+FS	CD+C+FS	CD	CD+C	CD+FS	CD+C+FS	CD+C	CD+FS	CD+C+FS
BUR	8	26	.08	.06	.01	.03	.79	.78	.51	.79	1.03	2.02	.97
CHR	14	12--25	12.47	11.92	8.12	7.34	51.20	44.50	40.20	38.48	1.92	1.62	2.20
ELS	1	19	1.25	.90	1.31	.00	27.44	20.59	28.08	27.60	1.93	2.19	2.55
ESC	20	16--128	1.04	.73	.18	.15	5.96	3.93	4.16	1.87	3.75	2.00	4.69
HAD	5	12--20	.03	.00	.02	.00	1.37	.59	.46	.30	2.37	2.70	3.27
KRA	3	30--32	2.61	2.23	1.85	1.72	2.25	1.67	1.23	1.12	1.57	2.16	2.36
LIPA A	8	20--90	1.63	1.06	1.07	.89	2.38	1.51	1.92	1.50	5.34	2.75	5.63
LIPA B	8	20--90	11.69	12.47	2.60	7.52	20.01	19.52	18.90	18.86	4.66	2.99	4.79
NUG	15	12--30	.73	.77	.23	.39	5.12	4.29	3.41	3.39	1.82	1.87	2.16
ROU	3	12--20	.35	1.01	.05	.31	6.90	5.33	4.78	4.67	3.06	1.74	2.81
SCR	3	12--20	3.22	1.24	.01	.00	10.53	7.60	7.36	6.62	2.31	1.70	2.43
SKO	13	42--100	1.35	1.30	.90	.86	2.67	2.47	1.96	2.02	1.39	2.02	2.15
STE	3	36	3.60	4.63	2.91	4.69	18.32	16.93	13.16	13.85	1.42	1.78	1.99
TAI	28	10--256	1.67	1.51	1.17	1.31	6.87	6.09	4.95	5.42	2.37	2.03	2.90
THO	3	30--150	2.01	1.74	1.13	1.20	4.38	4.03	3.37	3.36	1.33	1.98	1.95
WIL	2	50,100	.66	.45	.44	.39	1.52	1.35	1.15	1.13	1.35	2.02	2.11

	Average Percent Optimality Gap								Average Timing Ratio			
	Best of 100 Runs				Median of 100 Runs				(compared to CD)			
Type	CD+G	CD+C+G	CD+FS+G	CD+C+FS+G	CD+G	CD+C+G	CD+FS+G	CD+C+FS+G	CD+G	CD+C+G	CD+FS+G	CD+C+FS+G
BUR	.02	.02	.01	.01	.49	.52	.47	.50	3.26	3.37	2.91	3.25
CHR	9.99	8.51	7.57	7.34	39.16	38.12	37.38	38.26	1.84	2.57	2.15	2.53
ELS	.99	.90	1.13	.00	19.15	19.38	28.07	27.60	2.39	2.94	2.89	2.92
ESC	.65	.73	.15	.00	3.96	2.83	3.97	1.71	1.64	4.32	2.55	5.19
HAD	.00	.00	.01	.00	.35	.41	.14	.30	2.30	2.94	3.51	3.69
KRA	1.91	1.34	1.76	1.64	1.25	1.44	1.03	1.12	2.51	2.56	2.88	2.86
LIPA A	.96	.82	.81	.67	1.69	1.31	1.44	1.31	3.53	7.83	5.24	7.67
LIPA B	10.03	10.05	2.59	7.52	19.01	18.94	18.67	18.79	4.10	11.86	5.85	9.96
NUG	.52	.48	.22	.39	3.66	3.76	3.27	3.38	2.09	2.58	2.39	2.56
ROU	.06	.21	.05	.31	4.84	4.59	4.24	4.58	1.77	3.70	2.21	3.19
SCR	.01	.01	.00	.00	6.90	6.33	6.85	6.59	1.68	2.87	2.13	2.78
SKO	.93	1.03	.84	.83	2.20	2.08	1.91	2.01	9.50	9.55	4.45	3.06
STE	3.34	3.21	2.52	4.53	13.56	13.88	12.79	13.85	3.49	3.10	2.53	2.43
TAI	1.19	1.15	1.13	1.22	5.31	5.24	4.79	5.04	3.97	5.23	3.63	4.11
THO	1.31	1.12	1.02	1.19	3.64	3.49	3.30	3.36	5.31	5.51	4.84	3.10
WIL	.38	.34	.44	.39	1.26	1.17	1.11	1.13	9.21	8.64	4.88	3.05

Table 3. Performance of CD, CD+C, CD+FS, CD+C+FS, and the variants of these with the greedy pairwise swap heuristic on QAPLIB instances, aggregated by family of instances. These variants are compared to highlight the effects of (1) continuation, and (2) the greedy swap heuristic.

	Average Percent Optimality Gap								Average Timing Ratio			
	Best of 100 Runs				Median of 100 Runs				(compared to CD)			
Type	FAQ10	FAQ10+G	FAQ30	FAQ30+G	FAQ10	FAQ10+G	FAQ30	FAQ30+G	FAQ10	FAQ10+G	FAQ30	FAQ30+G
BUR	.08	.04	.06	.04	.27	.09	.19	.08	739.89	742.16	1940.95	1942.74
CHR	18.72	5.69	9.32	5.90	77.50	36.01	57.76	33.91	2.17	3.57	4.81	5.80
ELS	25.89	4.21	22.96	4.56	77.60	14.49	67.39	19.15	1078.06	1081.29	2612.64	2615.23
ESC	.73	.15	.05	.00	9.82	2.64	5.34	2.36	1.42	2.09	4.28	4.85
HAD	.39	.00	.15	.05	3.77	.77	.77	.37	3.45	5.65	10.35	11.32
KRA	2.46	1.67	.06	.00	4.10	1.50	1.59	1.19	5.79	7.30	19.57	14.11
LIPA A	1.79	1.21	1.02	.66	2.23	1.55	1.76	1.35	3.01	5.54	7.69	10.07
LIPA B	.00	.00	.00	.00	15.07	12.16	4.93	4.90	3.81	6.15	9.59	13.36
NUG	.62	.19	.03	.03	5.04	2.67	2.35	1.85	3.04	4.18	8.79	9.40
ROU	1.58	.77	.49	.48	8.45	4.47	4.68	3.74	2.33	3.19	6.98	7.56
SCR	3.94	.29	2.66	.24	19.36	7.66	15.24	8.16	2.34	3.26	6.63	7.41
SKO	.94	.61	.34	.28	2.18	1.64	1.23	1.15	3.67	11.66	9.68	12.00
STE	2.75	1.99	1.10	.53	13.37	8.87	8.78	7.80	4.20	6.17	11.44	12.41
TAI	2.90	1.78	1.53	1.08	9.23	5.89	6.01	4.71	51.47	54.17	193.50	195.77
THO	1.37	.96	.38	.35	3.39	2.58	2.04	1.88	4.51	8.85	12.80	15.06
WIL	.54	.31	.13	.09	1.39	.82	.61	.51	4.36	12.52	11.89	15.36

Table 4. Performance of FAQ10, FAQ10+G, FAQ30, and FAQ30+G on QAPLIB instances, aggregated by family of instances.

Additional results on CD. In Figure 3 we compare CD+C+RS, CD, and CD+C+FS in terms of the average run time and the gap between the average objective value obtained and the best objective value out of all runs of FAQ30+G. We also tested CD+C, but omitted the results since the run time was close to CD+C+RS and the objective values were slightly worse. CD and CD+C+RS scale similarly at larger values of n, while CD+C+FS starts increasing at a significantly faster rate. CD+C+FS can obtain an average objective that is sometimes as good as (or slightly better) than what FAQ30+G achieves, and the gaps for CD+C+FS and CD+C+RS are off by almost a constant.

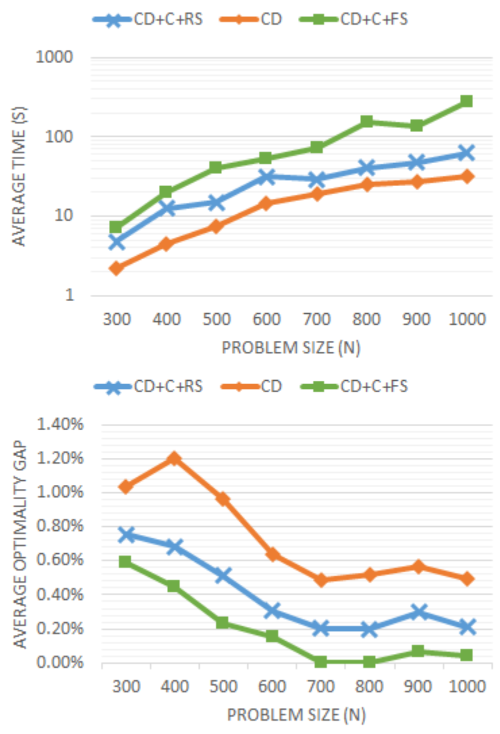

Figure 3. Comparison of CD+C+RS, CD, and CD+C+FS for synthetic QAP instances.