Title
An O(n3 [square root of] log n) algorithm for the optimal stable marriage problem

Author
Ng, Cheng

Publication Date
1990

Peer reviewed

An $O(n^3\sqrt{\log n})$ Algorithm for the Optimal Stable Marriage Problem

Cheng Ng

Technical Report No. 90-22

Abstract

We give an $O(n^3\sqrt{\log n})$ time algorithm for the optimal stable marriage problem. This algorithm finds a stable marriage that minimizes an objective function defined over all stable marriages in a given problem instance.

Irving, Leather, and Gusfield have previously provided a solution to this problem that runs in $O(n^4)$ time [ILG87]. In addition, Feder has claimed that an $O(n^3 \log n)$ time algorithm exists [F89]. Our result is an asymptotic improvement over both cases.

As part of our solution, we solve a special *blue-red matching problem*, and illustrate a technique for simulating Hopcroft and Karp's maximum-matching algorithm [HK73] on the transitive closure of a graph.

1. Introduction

An instance of the stable marriage problem involves two disjoint sets of equal cardinality n, the men denoted by m_i's and the women denoted by w_i's. Each individual ranks, in decreasing order of preference, all members of the opposite sex in a *preference list*. The set of preference lists determines completely the men and women's ranking functions, denoted by mr and wr respectively, as follows:

$$mr(m_i, w_j) = k \quad \text{if man } i \text{ ranks woman } j \text{ in position } k,$$
$$wr(w_i, m_j) = k \quad \text{if woman } i \text{ ranks man } j \text{ in position } k.$$

Note that a lower value of k indicates a higher ranking.

A *pair* (m_i, w_j) consists of a man and a woman. A *stable marriage* is a complete matching of men and women that does not result in an unmatched pair (m_i, w_j) such that m_i and w_j each ranks the other higher than his or her partner.

Most problem instances admit more than one stable marriage. However, the traditional algorithm, first proposed by Gale and Shapley [GS62], gives only the *male-optimal* solution. In this solution, every man has the best partner possible under any stable marriage; simultaneously, every woman has the worst partner possible.

The problem of finding more equitable stable marriages has been raised by Knuth [K76] and others [MW71] [W76]. The *optimal stable marriage problem* results from responding to such calls. Given a marriage $M = \{(m_1, w_1), \ldots, (m_n, w_n)\}$, define its *value* $c(M) = \sum_1^n mr(m_i, w_i) + \sum_1^n wr(w_i, m_i)$. The optimal stable marriage problem is to find a stable marriage with minimum value. Irving, Leather, and Gusfield provide a solution to this problem that runs in $O(n^4)$ time [ILG87]. Feder has claimed that an algorithm that runs in $O(n^3 \log n)$ time is available [F89].

Most steps in Irving, Leather, and Gusfield's solution require $O(n^2)$ time. The only exception is a bottleneck step that requires $\Theta(n^4)$ time. In this paper, we give an $O(n^3\sqrt{\log n})$ time algorithm for this step, thus reducing the overall time complexity to $O(n^3\sqrt{\log n})$. We transform the bottleneck step into a specialized matching problem in directed graphs, which we name the *blue-red matching problem*. This problem, to be defined later in this section, is solved via a simulation of Hopcroft and Karp's maximum-matching algorithm for bipartite graphs [HK73].

We conclude this section with some additional definitions. Section 2 reviews Hopcroft and Karp's techniques. Section 3 summarizes the work of Irving, Leather, and Gusfield. We develop the main ideas of this paper and give an $O(n^3 \log n)$ time algorithm in Section 4. In Section 5, we describe the modification necessary for this algorithm to run in $O(n^3\sqrt{\log n})$ time. Section 6 consists of some concluding remarks.

The *maximum-matching problem* is solved on an undirected graph $G = (V, E)$. The graph is *bipartite* if V can be partitioned into two subsets X and Y such that every edge in E joins a vertex in X with a vertex in Y. $M \subseteq E$ is a *matching* if no vertex $v \in V$ is incident on more than one edge in M. An edge (u, v) is *matched* if it is in M, and *unmatched* otherwise. A vertex v is *matched* if it is incident on an edge in M, and *unmatched* otherwise. The maximum-matching problem is to find a matching with maximum cardinality.

A *path* is a sequence of vertices $v_1, v_2, \ldots, v_k$ such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq k-1$. A path P is an *augmenting path* relative to the matching M if (i) k is even, (ii) v_1 and v_k are not matched, and (iii) the edges (v_i, v_{i+1}) are not matched for odd i's and matched for even i's. The *length* of P, denoted $|P|$, is the number of edges in P. $|P|$ is always odd for an augmenting path, and is equal to $k-1$ in the above example.

The *blue-red matching problem* is solved on a directed acyclic graph $G = (V, E)$. A *directed path* is a sequence of vertices $v_1, v_2, \ldots, v_k$ such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq k-1$. A vertex v is *reachable* from u if there is a directed path (including zero-length paths) that starts at u and ends at v. B and R are two distinguished vertex subsets such that $B \cap R = \emptyset$. For convenience, we refer to a vertex $b \in B$ as a *blue vertex* and a vertex $r \in R$ as a *red vertex*. A *blue-red matching* of size k is a set of ordered pairs of vertices $\{(b_1, r_1), (b_2, r_2), \ldots, (b_k, r_k)\}$ such that (i) all vertices are distinct, (ii) all b_i's are blue vertices and all r_i's are red vertices, and (iii) for every i, r_i is reachable from b_i. The blue-red matching problem is to find a blue-red matching of maximum size.

The *maximum-flow problem* is solved on a *flow network*, which is a directed graph $G = (V, E)$ with two distinguished vertices, a *source* s and a *sink* t, and a positive real-valued capacity $c(v, w)$ for every arc $(v, w) \in E$. A *flow* f on G is a real-valued function on E satisfying the constraints: (i) $0 \leq f(v, w) \leq c(v, w)$ for all $(v, w) \in E$, and (ii) $\sum_{u \in V} f(u, v) = \sum_{w \in V} f(v, w)$ for all $v \in V - \{s, t\}$. The value $|f|$ of a flow f is the net flow into the sink, i.e., $|f| = \sum_{v \in V} f(v, t)$. The maximum-flow problem is to find a flow with maximum value.

An arc (v, w) is *saturated* relative to a flow f if $f(v, w) = c(v, w)$. No additional flow can use a saturated arc. A flow f is a *blocking flow* if every path from s to t contains a saturated arc. Many fast maximum-flow algorithms work by finding a succession of blocking flows, a technique first introduced by Dinic [D70]. Our algorithm for the blue-red matching problem also requires solving a succession of blocking-flow problems.

2. Maximum Matching Algorithm

Augmenting paths are central to most matching algorithms. It is well known that a matching M can be increased if and only if there exists an augmenting path P relative to M [B57]. Given such a path P, the matching is increased by exchanging matched and unmatched edges along P.

Hopcroft and Karp's contribution is their observation that it is more efficient to always use the shortest available augmenting path when increasing a matching.

Their algorithm uses a sequence of augmenting paths $P_0, P_1, \ldots, P_k$ to compute a sequence of matchings $M_0 = \emptyset, M_1, \ldots, M_{k+1}$. Each P_i is a shortest augmenting path relative to M_i and it is used to increase the matching from M_i to M_{i+1}.

Theorem 2.1. [HK73] If $P_0, P_1, \ldots, P_k$ are nondecreasing in length, then $|P_i| = |P_j|$ implies that P_i and P_j are vertex disjoint.

Hopcroft and Karp's algorithm operates in *phases* on a bipartite graph with vertex partitions X and Y. Since equal-length augmenting paths are vertex disjoint, a maximal set is found in a single phase as follows. Execute a breadth-first search, starting with the set of unmatched vertices that are in X, and adding unmatched and matched edges at alternate levels to a graph H, until an unmatched vertex in Y is reached. A depth-first search of H then gives the required set of augmenting paths.

Theorem 2.2. [HK73] Suppose a matching M has cardinality r and the maximum matching has cardinality $s > r$. Then there exists an augmenting path relative to M of length $\leq 2\lfloor r/(s-r) \rfloor + 1$.

Theorem 2.2 implies that the length of shortest available augmenting paths cannot get very large until the cardinality of the matching is near the maximum possible. This idea is captured in Theorem 2.3. We reproduce its proof here since it is relevant to the analysis of our algorithm.

Theorem 2.3. [HK73] Suppose the maximum matching has cardinality s. Finding all augmenting paths requires $O(\sqrt{s})$ phases.

Proof. Consider the matching M_r of cardinality $r = \lfloor s - \sqrt{s} \rfloor$ that results from applying the sequence of augmenting paths $P_0, P_1, \ldots, P_{r-1}$. By Theorem 2.2,

$$|P_r| \quad \leq \quad 2\lfloor s - \sqrt{s} \rfloor / (s - \lfloor s - \sqrt{s} \rfloor) + 1 \quad \leq \quad 2\lfloor \sqrt{s} \rfloor + 1.$$

Since the algorithm finds all equal-length augmenting paths in one phase, only $O(\sqrt{s})$ phases are required to find all augmenting paths up to P_{r-1}. Moreover, there are only $s - r = O(\sqrt{s})$ augmenting paths remaining. Therefore, the total number of phases is $O(\sqrt{s})$. ∎

3. Optimal Stable Marriage Problem

A *stable pair* in an instance of the stable marriage problem is a pair that appears in some stable marriage. Gusfield demonstrated that it is possible to identify all stable pairs in $O(n^2)$ time [G87]. The remaining pairs do not serve any useful purpose, and may be discarded from the preference lists. We refer to the resulting abbreviated lists as *stable lists*.

An important device known as *rotation* is derived from the stable pairs. Rotations were first introduced by Irving and Leather [IL86], who used them to obtain a crucial understanding of the structure underlying the set of stable marriages. This understanding is the basis of Irving, Leather, and Gusfield's efficient algorithm for the optimal stable marriage problem.

Definition. A sequence $\rho = (m_0, w_0), \ldots, (m_{r-1}, w_{r-1})$ is a rotation if there is a marriage M such that, for all i, (i) (m_i, w_i) is matched in M, and (ii) w_{i+1} is the next woman after w_i in m_i's stable list ($i+1$ is taken modulo r). Every stable pair appears in at most one rotation [IL86]; so, the total number of rotations is $O(n^2)$.

Lemma 3.1. [G87] The set of rotations can be found in $O(n^2)$ time.

We summarize Irving, Leather, and Gusfield's main results and refer readers to [ILG87] for details. Related details are found in [IL86] [G87] [GI89]. Consider the rotation ρ in the above definition. The process of *eliminating the rotation* ρ involves switching the partner of every m_i from w_i to w_{i+1}, the next woman in m_i's stable list. Eliminating ρ results in a new stable marriage. However, every woman that m_i ranks higher than w_i places a constraint on the timing of ρ's elimination, regardless of whether she forms a stable pair with m_i. These constraints impose a partial order $\leq$ on the set of rotations P. The *rotation poset* that results, denoted by $(P, \leq)$, specifies the ordering in which rotations can be eliminated.

Definition. A subset $C \subseteq P$ is a *closed subset* if it has the property that for all $\rho \in C$ and for all $\pi \in P$, $\pi \leq \rho$ implies that $\pi \in C$.

Theorem 3.2. [IL86] [ILG87] The stable marriages of a given problem instance are in one-to-one correspondence with the closed subsets of the rotation poset.

Definition. Given a rotation $\rho = (m_0, w_0), \ldots, (m_{r-1}, w_{r-1})$, define its *weight*

$$w(\rho) = \sum_{0}^{r-1}(mr(m_i, w_i) - mr(m_i, w_{i+1})) + \sum_{0}^{r-1}(wr(w_i, m_i) - wr(w_i, m_{i-1})),$$

where $i-1$ and $i+1$ are taken modulo r. If M is the marriage before eliminating ρ, $w(\rho)$ is the net change in the value $c(M)$ due to the elimination. Define the *weight of a closed subset* as the sum of the weights of all rotations in the subset.

Given a closed subset C of maximum weight, Theorem 3.2 implies that an optimal stable marriage can be obtained by eliminating all rotations in C. Unfortunately, there is no known efficient way of finding a maximum-weight closed subset directly from $(P, \leq)$. The key to Irving, Leather, and Gusfield's solution is their demonstration that a succint representation is always available for $(P, \leq)$. This representation takes the form of a sparse directed acyclic graph P', which can be constructed in $O(n^2)$ time from the preference lists. P' has vertex set P and transitive closure equivalent to $\leq$, so it preserves the closed subsets of P.

A maximum-weight closed subset of P is obtained by solving a special maximum-flow problem based on P'. Details of this reduction are in [ILG87]. Each step in the reduction runs in $O(n^2)$ time, except the maximum-flow computation, which runs in $O(n^4)$ time. Moreover, the flow network—and more importantly for us, P'—has $O(n^2)$ vertices and $O(n^2)$ arcs.

Lemma 3.3. [ILG87] Let W^+ and W^- denote the sums of the weights of all rotations in P with positive and negative weights respectively. $W^+ \leq n^2$ and $|W^-| \leq n^2$.

The first part of Lemma 3.3 concerning W^+ is proved in [ILG87]. The second part has a similar proof.

4. Blue-Red Matching

Instead of reducing the maximum-weight closed subset problem to the maximum-flow problem, we reduce it to the blue-red matching problem. The blue-red matching problem is solved on a graph G constructed by adding vertices and arcs to Irving, Leather, and Gusfield's special directed acyclic graph P'. For each $\rho \in P'$ such that $w(\rho) = k < 0$, we add $|k|$ blue vertices, and an arc from each added vertex to ρ. For each $\rho \in P'$ such that $w(\rho) = k > 0$, we add k red vertices, and an arc from ρ to each added vertex. These added vertices are the only blue and red vertices.

The construction is illustrated in Figures 1 and 2. Figure 1 shows an example of P', and Figure 2 shows the corresponding graph G.

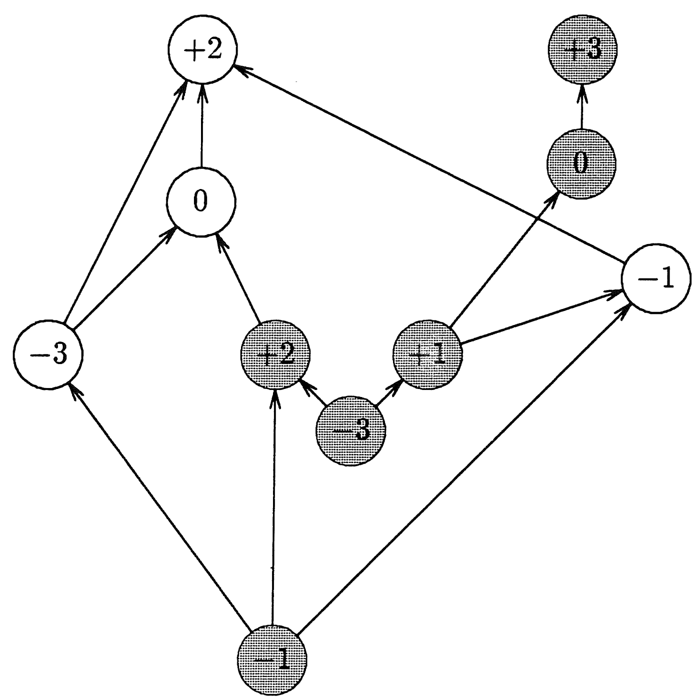

Figure 1. An example of P'. Numbers shown are weights of the vertices. The set of shaded vertices is a maximum-weight closed subset.

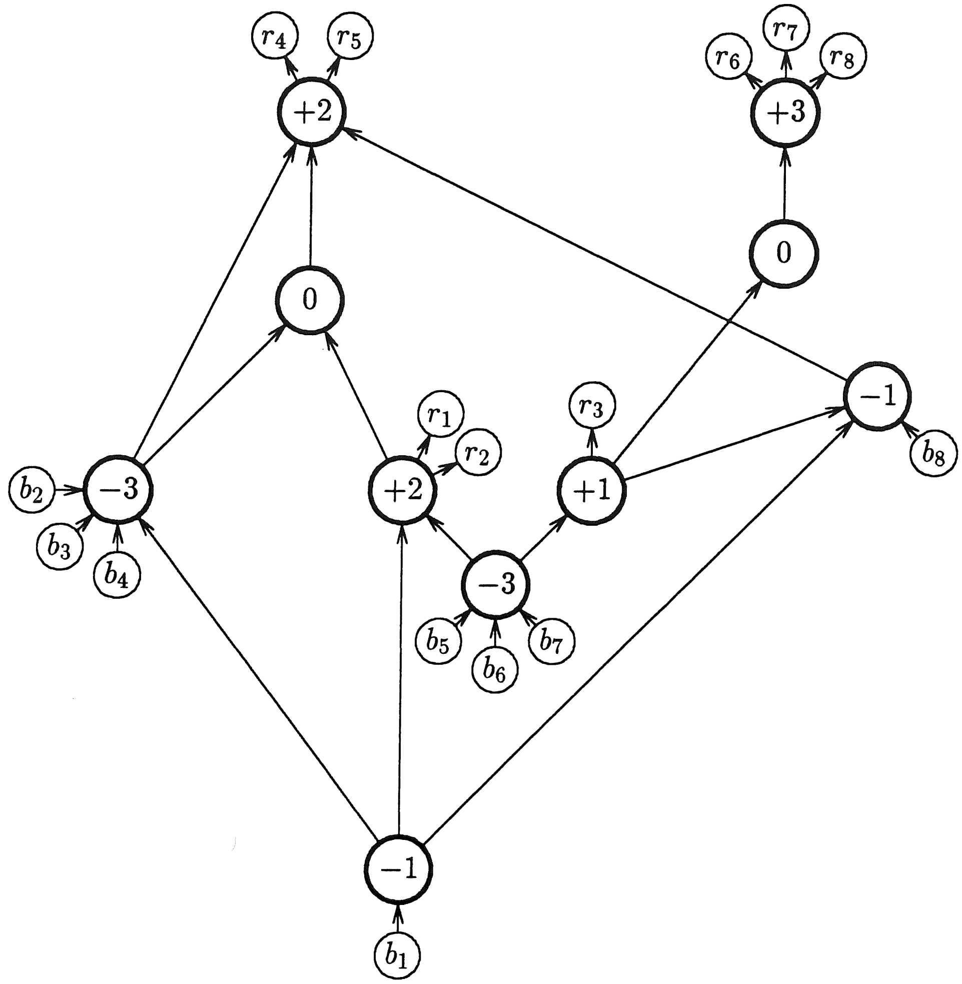

Figure 2. The graph G constructed from P' of Figure 1. The b_i's are blue vertices and r_i's are red vertices.

Lemma 4.1. The graph G has $O(n^2)$ vertices and $O(n^2)$ arcs, and is constructed in $O(n^2)$ time.

Proof. Initially, P' has $O(n^2)$ vertices and $O(n^2)$ arcs. The number of added vertices and arcs equals $W^+ + |W^-|$ of Lemma 3.3, which is $O(n^2)$. The time complexity follows immediately. ∎

In introducing the above reduction, we are motivated by a special relation that exists between maximum-flow and blue-red matching problems. This relation will be developed in Theorem 4.9 and Corollary 4.10, which we are able to prove only after describing the first part of the blue-red matching algorithm. It should be noted that the special relation is a generalization of the well-known equivalence between maximum-flow and maximum-matching problems [ET75] [LP86].

Algorithm Build_G'

Let $G' = (V', E')$. V' consists of two vertices s and t, plus a subset of the original vertices of G partitioned into layers $L_1, L_2, \ldots, L_x$. Two vertex subsets B_i and R_i in each L_i layer facilitate the construction of G'.

1. $E' \leftarrow \emptyset$.
2. $B_1 \leftarrow$ all unmatched blue vertices in G. For each $b \in B_1$, add an arc (s, b) to E' with capacity 1, i.e., $c(s, b) \leftarrow 1$.

For each i,

3a. $L_i \leftarrow$ all vertices in G reachable from a vertex in B_i and not already assigned to a layer. All arcs in the subgraph induced by L_i on G are added to E', with a capacity of ∞ for each arc.

3b. $R_i \leftarrow$ all red vertices in L_i. If R_i is empty, then $x \leftarrow i$, and *Build_G'* **terminates abnormally**.

3c. If all vertices in R_i are matched, then $B_{i+1} \leftarrow$ all vertices that are matched to vertices in R_i, and not already assigned to a layer. For each matched pair (r, b) such that $r \in R_i$ and $b \in B_{i+1}$, add an arc (r, b) to E' with capacity 1. Continue with layer L_{i+1}.

4. If some vertices in R_i are unmatched, then $x \leftarrow i$, i.e., L_i is the last layer. For each unmatched $r \in R_x$ add an arc (r, t) to E' with capacity 1. *Build_G'* **terminates normally**.

Figure 3.

and r_x are not matched. Therefore, $b_1, r_1, b_2, r_2, \ldots, b_x, r_x$ is an augmenting path. ∎

Corollary 4.3. Consider all arcs with capacity 1 along an s-t path P. If their endpoints, when sequenced from s to t, are given by $s, b_1, r_1, b_2, r_2, \ldots, b_x, r_x, t$; then the augmenting path corresponding to P is $b_1, r_1, b_2, r_2, \ldots, b_x, r_x$.

Proof. The arcs $(s, b_1), (r_1, b_2), \ldots, (r_x, t)$ in the proof of Lemma 4.2 account for all inter-layer arcs, which are exactly those arcs with capacity 1. ∎

Definition. If a vertex v is in L_i, then its *L-number*, denoted $L(v)$, is i.

Lemma 4.4. Suppose $P = b_1, r_1, b_2, r_2, \ldots, b_k, r_k$ is an augmenting path in G^*. If $L(b_i) = j$ and $j < x$, then (i) $L(r_i)$ exists and is no greater than j, and (ii) $L(b_{i+1})$ exists and is no greater than $j + 1$.

Proof. The edge (b_i, r_i) in G^* implies that r_i is reachable from b_i. Hence, r_i is in the same layer as b_i unless $r_i \in L_1 \cup L_2 \cup \cdots \cup L_{j-1}$. In any case, $L(r_i) \leq j$. In the augmenting path P, (r_i, b_{i+1}) is a matched edge. Since L_j is not the last

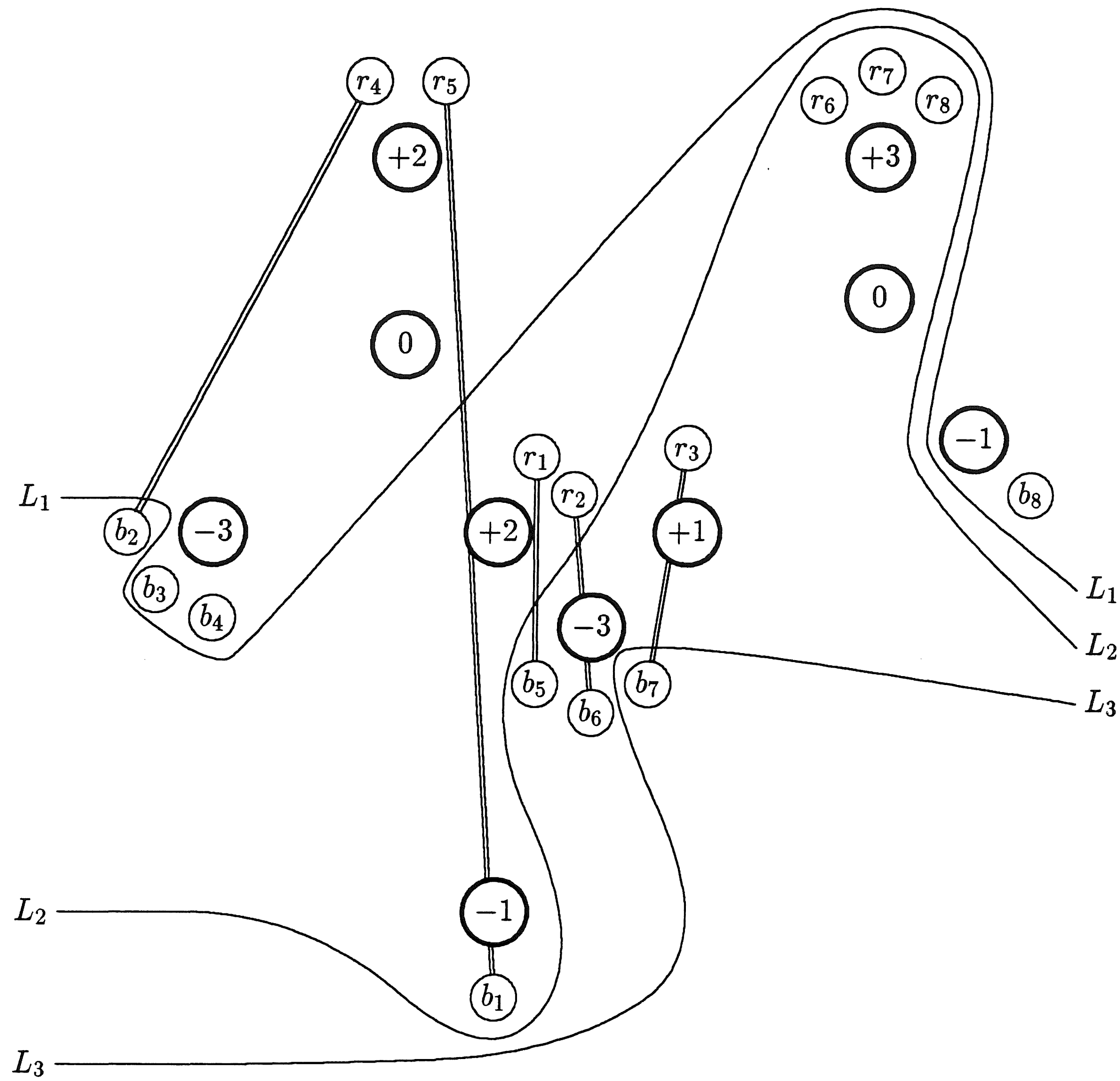

Figure 4. Layers in G, given $M = \{(b_1, r_5), (b_2, r_4), (b_5, r_1), (b_6, r_2), (b_7, r_3)\}$.

layer ($j < x$), b_{i+1} is placed in L_{j+1} by construction, unless it already has a lower L-number. ∎

Corollary 4.5. Suppose $b_1, r_1, b_2, r_2, \ldots, b_k, r_k$ is an augmenting path. For all $1 \leq i \leq x$, $L(b_i) \leq i$ and $L(r_i) \leq i$.

Proof. An augmenting path must start with an unmatched blue vertex b_1, which is placed in L_1 by construction. Therefore, $L(b_1) = 1$. The corollary then follows from Lemma 4.4 by induction. ∎

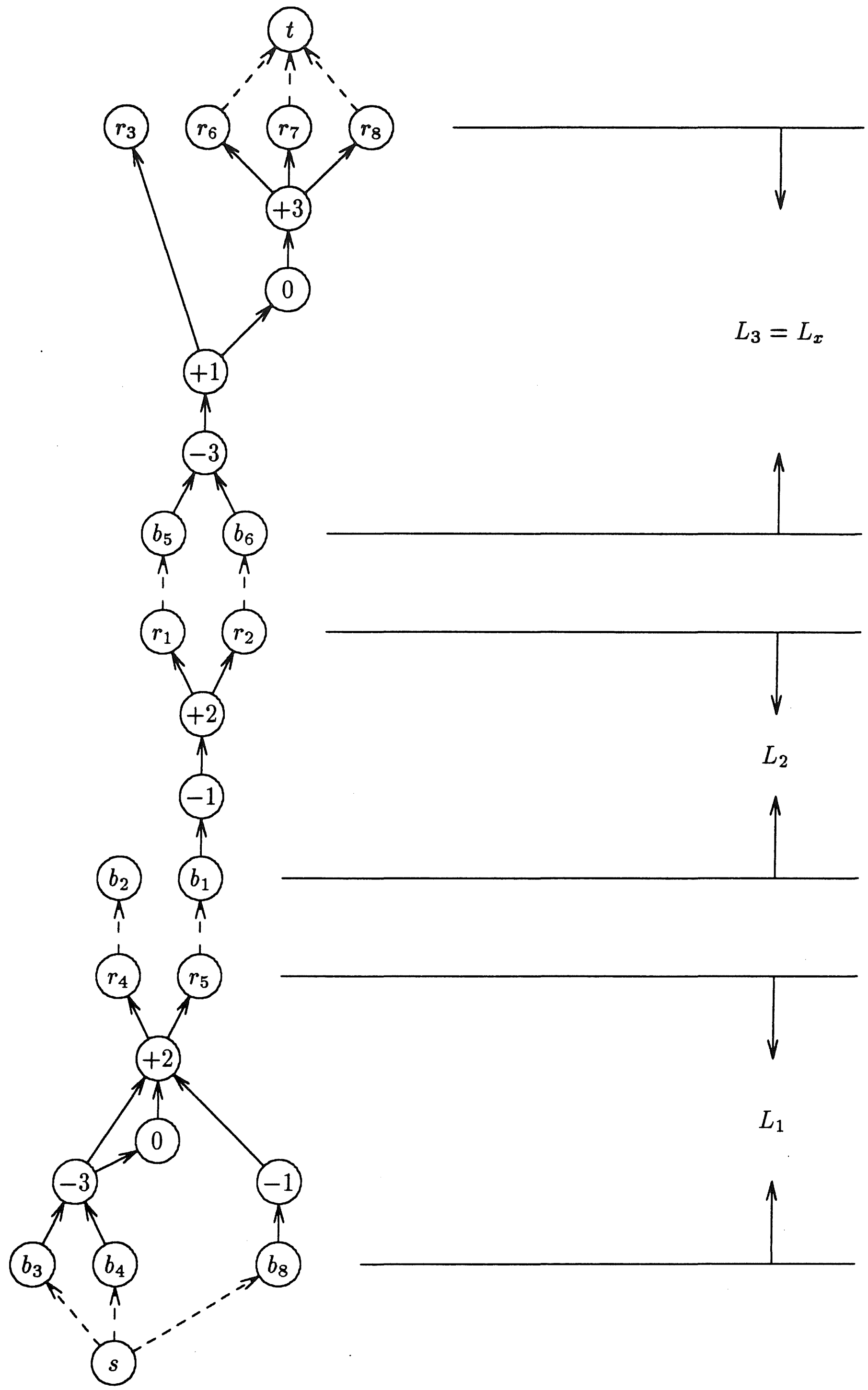

Figure 5. The graph G'. Dashed arrows denote arcs with capacity 1. All other arcs have infinite capacity.

Corollary 4.6. If *Build_G'* terminates normally, the shortest augmenting path has length $2x-1$.

Proof. If *Build_G'* terminates normally, the vertex t is in G'. Since G' is connected, an s-t path exists. By Lemma 4.2, there is a corresponding augmenting path of length $2x-1$. If a shorter augmenting path exists, consider its final vertex r, which must be unmatched by definition. $L(r) < x$ by Corollary 4.5. However, all vertices with L-number less than x are matched, a contradiction. ∎

Theorem 4.7. If *Build_G'* terminates normally, the s-t paths in G' are in one-to-one correspondence with the shortest augmenting paths in G^*.

Proof. Half of the proof is provided by Lemma 4.2. For the other half, consider an augmenting path $P = b_1, r_1, b_2, r_2, \ldots, b_x, r_x$ of length $2x-1$. By Corollary 4.5, all vertices in P have L-numbers and are therefore in G'. Moreover, $L(r_x) \leq x$. Since L_x is the only layer with unmatched vertices, $L(r_x) = x$. It follows that the constraints of Corollary 4.5 can only be satisfied by equality, that is, $L(b_i) = L(r_i) = i$, for all i.

The edge (b_i, r_i) in G^* implies that there is a path from b_i to r_i in G. This path is preserved in G' since both vertices are in the same layer L_i, and L_i is an induced subgraph of G. The matched edge (r_i, b_{i+1}) is used to construct a corresponding arc in G'. The arcs (s, b_1) and (r_x, t) are also in G' because b_1 and r_x are unmatched vertices. These arcs and paths combine to give the required s-t path. ∎

Definition. Given a closed subset $C \subseteq P$, a red or blue vertex is *associated* with C if it is adjacent to a vertex in C.

Lemma 4.8. If k red vertices remain unmatched in M, then the weight of any closed subset C is at most k.

Proof. A red vertex r can be matched with a blue vertex b only if r is reachable from b. Suppose r is adjacent to $\rho \in P$ and b is adjacent to $\pi \in P$. By construction, ρ is reachable from π if r is reachable from b. If ρ is in C, then π is also in C since C is a closed subset. Therefore, all red vertices associated with C can be matched only with blue vertices associated with C.

The weight of C is equal to the difference between the number of red vertices and the number of blue vertices associated with C. This difference cannot exceed k. Otherwise, the number of red vertices associated with C that remain unmatched in M exceeds k, the number of red vertices that remain unmatched overall. ∎

Theorem 4.9. Let $C = P - V'$, where V' is the vertex set of G'. If *Build_G'* terminates abnormally, then C is a maximum-weight closed subset of P.

Proof. Suppose ρ and π satisfy the conditions (i) $\rho \in C$, (ii) $\pi \in P$, and (iii) $\pi \leq \rho$. Condition (i) implies that $\rho \notin V'$ and condition (iii) implies that ρ is reachable from π. If $\pi \in V'$, then *Build_G'* assigns ρ to π's layer in V' (Step 3a), resulting in a contradiction. Therefore, $\pi \notin V'$, which implies that $\pi \in C$ and C is a closed subset.

All unmatched blue vertices are in V' (in the layer L_1) by construction; therefore, all blue vertices associated with C are matched. Moreover, they are matched only with red vertices associated with C since any blue vertex matched to a red vertex in V' ends up in V' (Step 3c). Therefore, C's weight is exactly the number of red vertices associated with C that remain unmatched in M. However, all unmatched red vertices are associated with C since all red vertices in L_1 to L_{x-1} are matched, by construction, and L_x has no red vertex because *Build_G'* terminated abnormally.

Therefore, C's weight is exactly the number of unmatched red vertices overall. By Lemma 4.8, this is the maximum possible weight of any closed subset. ∎

From a practical point of view, Theorem 4.9 in itself provides for the recovery of a maximum-weight closed subset. The next corollary is included mainly for theoretical interest, since it demonstrates that a maximum blue-red matching is reached simultaneously.

Corollary 4.10. If *Build_G'* terminates abnormally, then M is a maximum blue-red matching.

Proof. If M can be increased, the number of red vertices that remain unmatched can be decreased. By Lemma 4.8, no closed subset can then have the weight of C, a contradiction. ∎

Theorem 4.7 establishes the relation between s-t paths in G' and shortest augmenting paths in G^*. To push a unit of flow in G' clearly requires an s-t path. Consider all arcs with capacity 1 on this path. By Corollary 4.3, their endpoints specify completely the corresponding augmenting path in G^*. Since the flow saturates these arcs, they cannot be used for another unit of flow. Therefore, augmenting paths that correspond to two different units of flow must be vertex-disjoint.

The above discussion demonstrates that we can find vertex-disjoint augmenting paths by computing a flow in G'. If it is a blocking flow, every path from s to t has a saturated arc, and therefore, no additional s-t path is available. This implies that the set of vertex-disjoint augmenting paths is maximal when a blocking flow is reached. The *Find_Paths* algorithm (Figure 6) finds such a blocking flow.

Find_Paths is essentially an adaptation of Sleator and Tarjan's blocking-flow algorithm [ST83], with an added routine that recovers the augmenting paths simultaneously. Sleator and Tarjan use *dynamic trees* to achieve a highly efficient algorithm. The dynamic trees store information in a forest of vertex-disjoint rooted trees. Within each tree, every edge has a real-valued cost and is directed towards the root. These trees are maintained by an appropriate data structure that supports a rich set of operations efficiently. We list those operations required by *Find_Paths*.

root(v): Return the root of the tree containing v.

Algorithm Find_Paths (Adapted from Sleator and Tarjan [ST83])

Initialize each vertex as a separate tree.

Step 1. $v \leftarrow root(s)$. If $v = t$, go to Step 4; otherwise, go to Step 2.

Step 2. ($v \neq t$; extend path). If no arc leaves vertex v, go to step 3. Otherwise, select an arc (v, w) and perform $link(v, w, c(v, w))$. Go to Step 1.

Step 3. (all paths from v to t are blocked). If $v = s$, stop. Otherwise, delete from G' every arc entering v. For each such arc (u, v) that is a tree edge, perform $cut(u)$. Go to Step 1.

Step 4. ($v = t$; an s-t path is found).
repeat
  $r \leftarrow mincost(s)$;
  output $parent(r)$ and r;
  delete the edge $(r, parent(r))$ from G';
  perform $cut(r)$
until $r = s$.
Go to Step 1.

Figure 6.

parent(v): Return the parent of v. This operation assumes that v is not a tree root.

mincost(v): Return the vertex w closest to $root(v)$ such that the edge $(w, parent(w))$ has minimum cost among all edges on the path from v to $root(v)$. This operation assumes that v is not a tree root.

link(v, w, x): Combine two trees by adding an edge (v, w) of cost x, making w the parent of v. This operation assumes that v and w are in different trees and v is a tree root.

cut(v): Delete the edge $(v, parent(v))$, thus dividing the tree containing v into two trees. This operation assumes that v is not a tree root.

If the maximum size of any tree is n, the dynamic trees implementation [ST83] runs in $O(\log n)$ time per operation. A simpler implementation based on splay trees is also available [ST85]. The latter implementation supports arbitrary ordering of operations in $O(\log n)$ amortized time per operation, which meets the requirements of *Find_Paths*.

Find_Paths inherits its correctness from Sleator and Tarjan's algorithm. The key idea is that an arc is only deleted from G' when it is no longer possible to use it in another s-t path. All edges in dynamic trees correspond to arcs that have not been deleted; therefore, when $v = t$ in step 1, an s-t path is found correctly.

Sleator and Tarjan's algorithm pushes flow in Step 4 and updates the capacities remaining in edges along the s-t path. It is possible to simplify this step in *Find_Paths* because there are only two types of tree edges: those edges with infinite capacity can be used an unlimited number of times and those edges with capacity 1 can be used only once. *Find_Paths* uses the *mincost* operation repeatedly to find all edges with capacity 1, outputs their endpoints for use in updating M, and deletes these edges from G' and the dynamic trees.

To obtain a more general analysis of the blue-red matching algorithm, we assume that G has $O(N)$ vertices and $O(E)$ arcs. At the end of this section, we substitute the actual values of N and E to get the time complexity for the optimal stable marriage problem.

Lemma 4.11. G' has $O(N)$ vertices and $O(E)$ arcs.

Proof. G' has at most two more vertices (s and t) than G. There are three types of arcs: $O(E)$ arcs are from G, $O(N)$ arcs from the matching M, and $O(N)$ arcs each with one endpoint either in s or t. ∎

Lemma 4.12. *Build_G'* runs in $O(E)$ time.

Proof. The bulk of the work is in Steps 3a, b, and c. In Step 3a, after obtaining L_i from B_i using breadth-first search, we may delete from G all arcs adjacent to vertices in L_i since these vertices will not be assigned to another layer. Therefore, every arc in G is visited and deleted at most once, and Steps 3a and 3b run in $O(E)$ time when summed over all i's.

Step 3c checks any red vertex at most once for the possibility that it is matched. It is easy to devise a data structure for M such that Step 3c runs in $O(N)$ time. ∎

Lemma 4.13. *Find_Paths* runs in $O(E \log N)$ time.

Proof. The $O(E \log N)$ time bound is inherited from Sleator and Tarjan's algorithm. Each *cut* operation must be preceded by a *link* of the same edge. Before a tree edge is *cut*, the corresponding arc in G' is deleted and is never visited again. Therefore, there are $O(E)$ *cuts*, $O(E)$ *links*, and $O(E)$ processing time for arcs in G'. Each *root* operation in Step 1 must be followed by either a *link* in Step 2 or a *cut* in Step 3 or 4. There are $O(1)$ *mincost* and *parent* operations per *cut* in Step 4.

All dynamic trees operations are accounted for in the above discussion, giving a total of $O(E)$ such operations requiring a total of $O(E \log N)$ time. ∎

Theorem 4.14. The blue-red matching algorithm runs in $O(\sqrt{N} E \log N)$ time on a graph with N vertices and E arcs.

Proof. The blue-red matching algorithm simulates Hopcroft and Karp's maximum-matching algorithm. By Theorem 2.3, there are $O(\sqrt{N})$ phases since $N/2$ is the maximum size of the blue-red matching. Each phase executes *Build_G'* and *Find_Paths* once, and requires $O(E \log N)$ time, according to Lemmas 4.12 and 4.13. ∎

Theorem 4.15. The optimal stable marriage problem has a worst-case time complexity of $O(n^3 \log n)$.

Proof. As noted earlier, blue-red matching is the bottleneck step in optimal stable marriage; all other steps require $O(n^2)$ time. By Lemma 4.1, we can substitute $N = O(n^2)$ and $E = O(n^2)$ in Theorem 4.14, giving an $O(n^3 \log n)$ overall time bound. ∎

5. Speeding up Blue-Red Matching

We use *Find_Paths* to find a maximal set of shortest augmenting paths. However, when only a single shortest augmenting path is needed, it is sufficient to perform a depth-first search on G' in $O(E)$ time. This observation is the key to speeding up blue-red matching.

The revised algorithm duplicates the original algorithm until the number of layers in G' exceeds $\sqrt{N/\log N}$. The remaining augmenting paths are found by repeated applications of *Build_G'*, each application followed by a depth-first search of the resulting graph G'.

Theorem 5.1. The revised blue-red matching algorithm runs in $O(E\sqrt{N \log N})$ time on a graph with N vertices and E arcs.

Proof. Suppose the maximum blue-red matching has size s, and suppose the size of the blue-red matching is r when the number of layers in G' first exceeds $\sqrt{N/\log N}$. By Theorem 2.2, the shortest augmenting path has length $\leq 2r/(s-r)+1$. By Corollary 4.6, this length is $> 2\sqrt{N/\log N} - 1$. Therefore,

$$\begin{aligned} 2\sqrt{N/\log N} - 1 &< \frac{2r}{s-r} + 1, \quad \text{which implies} \\ s - r &< \frac{s}{\sqrt{N/\log N}} \leq \frac{1}{2}\sqrt{N \log N}. \end{aligned}$$

The last inequality is due to $s \leq N/2$.

The number of augmenting paths remaining is $s - r = O(\sqrt{N \log N})$. These paths are found by repeated applications of *Build_G'* and depth-first search, for a total of $O(E\sqrt{N \log N})$ time. The first r augmenting paths have lengths $\leq 2\sqrt{N/\log N} - 1$. They are computed in $O(\sqrt{N/\log N})$ phases of *Build_G'* and *Find_Paths*, for a total of $O(E\sqrt{N \log N})$ time. ∎

Corollary 5.2. The optimal stable marriage problem has a worst-case time complexity of $O(n^3\sqrt{\log n})$.

Proof. Substitute $N = O(n^2)$ and $E = O(n^2)$ in Theorem 5.1. ∎

6. Remarks

We give an $O(n^3\sqrt{\log n})$ time algorithm for the optimal stable marriage problem. Asymptotically, this is an improvement over Irving, Leather, and Gusfield's $O(n^4)$ time algorithm [ILG87] and Feder's claim of an $O(n^3 \log n)$ time algorithm [F89].

The optimal stable marriage problem can be generalized to a weighted version where the ranking functions mr and wr are replaced by a general weight function c when computing the value of a marriage. The ***weighted optimal stable marriage problem*** is to find a stable marriage that minimizes $c(M) = \sum_{(m,w)\in M}\big(c(m,w) + c(w,m)\big)$. This weighted version allows each person to specify the structure of his/her preferences in more detail and hence may give more useful solutions.

Our algorithm is applicable to the weighted version when c is an integer function with small values. For example, if the value of $c(p,q)$ for each pair (p,q) is chosen to be within a constant multiple of the corresponding value of $mr(p,q)$ or $wr(p,q)$, the preference structure is still fairly flexible yet $W^+ + |W^-|$ in Lemma 3.3 remains $O(n^2)$. Let $U = W^+ + |W^-|$ and assume that its value is $\Omega(n^2)$. By Lemma 4.1, the graph G has U vertices and U arcs and by Theorem 5.1, our algorithm runs in $O(U^{3/2}\sqrt{\log U})$.

When $U = O(n^2)$, the time complexities for our algorithm and Irving, Leather, and Gusfield's algorithm remain unchanged. For larger U's, Irving, Leather, and Gusfield give an algorithm that runs in $O(n^4 \log n)$ time [ILG87]. Our algorithm is asymptotically faster whenever $U = o(n^{8/3}\sqrt[3]{\log n})$. However, Irving, Leather, and Gusfield's algorithm also works when c is a real-valued function whereas ours only works for integer functions. Due to the lack of details, it is not known if Feder's approach solves the weighted optimal stable marriage problem.

An obvious open question is whether blue-red matching can be improved. Hopcroft and Karp's matching algorithm runs in $O(\sqrt{N}E)$ time on a graph with N vertices and E arcs [HK73]. Therefore, any approach similar to ours cannot run faster than $O(n^3)$ time (recall that $N = O(n^2)$ and $E = O(n^2)$), unless it also improves on Hopcroft and Karp's result. Nevertheless, it is still interesting to investigate the possibility of removing the extra $\sqrt{\log n}$ factor from our algorithm. One possible approach is to look for an alternative to *Find_Paths* that does not use dynamic trees.

References

[B57] C. Berge, *Two theorems in graph theory*, Proc. Nat. Acad. Sci. U.S.A., 43(1957), pp. 842–844.

[D70] E. A. Dinic, *Algorithm for solution of a problem of maximal flow in a network with power estimation*, Sov. Math. Dokl., 11(1970), pp. 1277–1280.

[ET75] S. Even and R. E. Tarjan, *Network flow and testing graph connectivity*, SIAM J. Comput., 4(1975), pp. 507–518.

[F89] T. Feder, *A new fixed point approach for stable networks and stable marriages*, In Proceedings of the 21st Annual ACM Symposium on Theory of Computing, ACM SIGACT, Seattle, Washington, 1989, pp. 513–522.

[GS62] D. Gale and L. Shapley, *College admissions and the stability of marriage*, Amer. Math. Monthly, 69(1962), pp. 9–15.

[G87] D. Gusfield, *Three fast algorithms for four problems in stable marriage*, SIAM J. Comput., 16(1987), pp. 111–128.

[GI89] D. Gusfield and R. W. Irving, *The Stable Marriage Problem: Structure and Algorithms*, MIT Press, Cambridge, Massachusetts, 1989.

[HK73] J. Hopcroft and R. Karp, *An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs*, SIAM J. Comput., 2(1973), pp. 225–231.

[IL86] R. W. Irving and P. Leather, *The complexity of counting stable marriages*, SIAM J. Comput., 15(1986), pp. 655–667.

[ILG87] R. W. Irving, P. Leather and D. Gusfield, *An efficient algorithm for the "optimal" stable marriage*, J. Assoc. Comput. Mach., 34(1987), pp. 532–543.

[K76] D. E. Knuth, *Mariages Stables*, Les Presses de L'Université de Montréal, Montréal, 1976.

[LP86] L. Lovász and M. D. Plummer, *Matching Theory*, North-Holland, Amsterdam, 1986.

[MW71] D. G. McVitie and L. B. Wilson, *The stable marriage problem*, Comm. ACM, 14(1971), pp. 486–492.

[ST83] D. D. Sleator and R. E. Tarjan, *A data structure for dynamic trees*, J. Comput. System Sci., 26(1983), pp. 362–391.

[ST85] D. D. Sleator and R. E. Tarjan, *Self-adjusting binary search trees*, J. Assoc. Comput. Mach., 32(1985), pp. 652–686.

[W76] N. Wirth, *Algorithms + Data Structures = Programs*, Prentice-Hall, Englewood Cliffs, N.J., 1976.