The Unsplittable Stable Marriage Problem

Brian C. Dean, Michel X. Goemans, and Nicole Immorlica

[1] Department of Computer Science, Clemson University. bcdean@cs.clemson.edu
[2] Department of Mathematics, M.I.T. goemans@math.mit.edu
[3] Microsoft Research. nickle@microsoft.com

Abstract. The Gale-Shapley "propose/reject" algorithm is a well-known procedure for solving the classical stable marriage problem. In this paper we study this algorithm in the context of the many-to-many stable marriage problem, also known as the stable allocation or ordinal transportation problem. We present an integral variant of the Gale-Shapley algorithm that provides a direct analog, in the context of "ordinal" assignment problems, of a well-known bicriteria approximation algorithm of Shmoys and Tardos for scheduling on unrelated parallel machines with costs. If we are assigning, say, jobs to machines, our algorithm finds an unsplit (non-preemptive) stable assignment where every job is assigned at least as well as it could be in any fractional stable assignment, and where each machine is congested by at most the processing time of the largest job.

1 Introduction

In the United States, a medical school graduate is required to complete a residency program at a hospital before entering the workforce as a doctor. Since the 1950s, the medical field has turned to a centralized mechanism, called the *National Residency Matching Program* (NRMP), to aid this marketplace [10]. In this program, final-year medical students and hospitals each submit preferences over possible matches, and an algorithm determines which matches will take place. In order for the system to be successful, it is essential that the computed matches be *stable*. That is, there should be no (student, hospital) pair that both prefer each-other to their assigned partners — such a pair would have an incentive to withdraw from the centralized matching system and to make its own plans on the side. Computing a stable matching is a classic problem in economics and computer science, and can be solved in polynomial time by the deferred acceptance algorithm of Gale and Shapley [3].[1]

For many years the NRMP proved to be quite successful. However, in the late 1990s it was observed that many matches were being formed outside the NRMP [12]. The problem stemmed from the fact that many medical students were getting married to one another during medical school, and so had complicated preferences that were ignored by the NRMP. In particular, married

[1] For a discussion of this problem and related questions, see the books by Gusfield and Irving [4] and Roth and Sotomayor [14], or the lecture notes by Knuth [8].

students had strong preferences for hospitals in similar geographical locations. The NRMP was redesigned to accommodate such preferences [13]; currently, the NRMP permits married students to submit a joint preference list over *pairs* of hospitals and guarantees that, if they are matched, they will be matched to a pair in their list. Unfortunately, in a matching market with couples like the NRMP, a stable matching might not exist [10] and determining whether one exists is computationally difficult, in fact NP-hard [9].

Motivated by the issue of couples in the NRMP, we study a marketplace in which agents on one side of the market have non-uniform demands and agents on the other side have non-uniform quotas, or capacities. Demanding agents have a preference list over capacitated agents and prefer to be satisfied by a lexicographically maximal set of these agents. This problem is known as the *stable allocation* or *ordinal transportation* problem, and is a many-to-many generalization of the classical stable marriage problem, introduced originally by Baiou and Balinski [1]. It surfaces naturally in scheduling or load balancing settings where only "ordinal" information (ranked preference lists) is known. When demands are all 1 or 2 and capacities are integral, as in the student/hospital setting, this restricted preference domain becomes a special case of *weakly responsive preferences* studied by Klaus and Klijn [6]. In such cases, Klaus and Klijn [6] proved that a stable matching always exists. Instances of this problem with generalized demands/capacities include the assignment of teaching assistants (TAs) to courses in academic departments: TAs rank courses, course instructors rank TAs, each course requires a certain number of TA hours, and different TAs are responsible for working different numbers of hours. Another example is the assignment of load to servers in a network – clients prefer servers geographically nearby and servers prefer clients with higher service types. Baiou and Balinski [1] study these generalized settings and prove that even in this case a stable allocation always exists.

For many settings, a stable allocation in which the demand of a single agent is satisfied fractionally is undesirable. Although a couple may prefer hospital a to b and thus a pair of placements (a, b) to a pair of placements (b, b), such an arrangement imposes strain on the matching. As often happens in labor markets with two-body problems, the couple may negotiate with hospital a to create an extra position, beyond the quota, for the extra member of the couple. In some sense, a fractional stable assignment is not stable. Thus, we seek a stable matching in which all the demand of a single agent is satisfied integrally. Clearly, such a matching may not exist, and so we relax our feasibility constraints and allow capacitated agents to be over-capacitated by at most the maximum demand. With a correspondingly appropriate modification of the definition of stability, we prove that a stable matching *always* exists, and give a modification of the Gale-Shapley algorithm to find it. Applied to the NRMP setting, our results compute a student-optimal (or hospital-optimal) stable matching where the number of students assigned to each hospital exceeds its quota by at most one position.

A close relative of the stable allocation problem is the well-studied *transportation* problem, where there are linear costs associated with every possible pairing and our objective is to compute a fractional assignment of minimum cost rather than a stable assignment. The stable allocation problem is also known as the ordinal transportation problem since it differs only in that we express the desirability of an assignment in an "ordinal" fashion using ranked preference lists. Unsplittable variants of the transportation problem have been previously considered in the literature, and a celebrated result of Shmoys and Tardos [15] states that from a fractional assignment (where all agents are fully assigned), we can construct an unsplit assignment of no greater cost where each agent is over-capacitated (or *congested*) by at most the maximum demand. Our results can viewed as a direct analog of this result for the ordinal case.

2 The Model

Consider assigning a set $[n] := \{1, 2, \ldots, n\}$ of items to a set $[m]$ of bins. To be somewhat more concrete, let us employ scheduling terminology and assume we are assigning "jobs" to "machines". Job i requires p_i units of processing time, machine j has a capacity of c_j units, and at most u_{ij} units of job i can be assigned to machine j. If $u_{ij} = p_i$ for all (i, j), we follow the terminology of Baiou and Balinski [1] and say our problem is *unconstrained.* All problem data is assumed to be integral.

2.1 Fractional Assignment

We first define a fractional setting where a job may be processed on multiple machines. A fractional assignment x is *feasible* if it satisfies

$$\begin{aligned} &\sum_{j\in[m]} x_{ij} \le p_i \;\; \forall i \in [n] \\ &\sum_{i\in[n]} x_{ij} \le c_j \;\;\; \forall j \in [m] \\ &0 \le x_{ij} \le u_{ij} \;\; \forall (i,j) \in [n] \times [m]. \end{aligned} \tag{1}$$

In the traditional *transportation* problem (a many-to-many generalization of the bipartite assignment problem), we designate a weight w_{ij} for assigning one unit of job i to machine j, then maximize $\sum w_{ij}x_{ij}$ over (1) using linear programming or network flow techniques (another popular objective is to minimize $\sum w_{ij}x_{ij}$ while insisting that all jobs must be fully assigned). In the *stable allocation problem*, however, we indicate the desirability of an assignment in an "ordinal" fashion by having each job (machine) submit a ranked preference list over all machines (jobs).

Thus, each job $i \in [n]$ has a strict, transitive, and complete preference relation $\pi(i)$ over the set $[m] \cup \emptyset$ where $\{\emptyset\}$ indicates a preference for remaining unmatched. If $\pi(i) = (j_1, \ldots, j_{k-1}, \emptyset = j_k, j_{k+1}, \ldots, j_{m+1})$, then i prefers j_a

to j_b for any $a < b < k$, and prefers being unassigned to any machine j_c for $c > k$. If job i prefers machine j to machine j', we write $j >_i j'$. Job i prefers a fractional assignment x to another fractional assignment x' if x is lexicographically larger according to $\pi(i)$; that is, if $x_{ij} > x'_{ij}$ for the earliest machine j in $\pi(i)$ such that $x_{ij} \neq x'_{ij}$. In this case, we write $x >_i x'$. Similarly, each machine $j \in [m]$ has a strict, transitive, and complete preference relation $\pi(j)$ over the set $[n] \cup \emptyset$ where $\emptyset$ indicates a preference for being under-utilized. If $\pi(j) = (i_1, \ldots, i_{k-1}, \emptyset = i_k, i_{k+1}, \ldots, i_{n+1})$, then j prefers to accept load from job i_a to i_b for any $a < b < k$, and is unwilling to process load from any job i_c with $c > k$. We write $i >_j i'$ if machine j prefers job i to job i', and we write $x >_j x'$ if machine j prefers assignment x to assignment x'; again, this means that $x_{ij} > x'_{ij}$ for the first job i in $\pi(j)$ where $x_{ij} \neq x'_{ij}$.

A *blocking pair* is a familiar feature that is forbidden in any stable assignment: it is a pair (i, j) where $x_{ij} < u_{ij}$ and both i and j prefer each-other to at least some of their current assignments. In this case, job i and machine j would be "unhappy" with the current assignment and would prefer to increase x_{ij}. That is,

Definition 1. *Job i and machine j form a* blocking pair *if there is some job i' and machine j' such that $x_{ij} < u_{ij}$, $x_{ij'} > 0$, $x_{i'j} > 0$, and we have $i >_j i'$ and $j >_i j'$.*

A job i is saturated if all its load is assigned. Similarly, a machine is saturated if all its capacity is utilized.

Definition 2. *A job i is* saturated *if $\sum_j x_{ij} \geq p_i$. A machine j is* saturated *if $\sum_i x_{ij} \geq c_j$.*

Finally, a job i is said to be *popular* in an assignment if there is some machine j to which i is not assigned, but where j prefers i to at least some of the jobs currently assigned to it. We define a popular machine similarly.

Definition 3. *In an assignment x, we say job i is* popular *if there exists a machine j with $j >_i \emptyset$ and $x_{ij} < u_{ij}$ such that $i >_j i'$ for some job i' with $x_{i'j} > 0$. Likewise, we say machine j is popular if there exists a job i with $i >_j \emptyset$ and $x_{ij} < u_{ij}$ such that $j >_i j'$ for some machine j' with $x_{ij'} > 0$.*

If job i is popular due to machine j and i is not saturated, then our assignment is not stable since both i and j would be more satisfied if x_{ij} were increased.

Definition 4. *An assignment x is* stable *if (i) it admits no blocking pairs, and (ii) all popular jobs and machines are saturated.*

A feasible stable assignment x is said to be *job-optimal* if every job prefers x to any other feasible stable assignment x', i.e. $\forall\ i \in [n],\ x >_i x'$ (a *machine-optimal* assignment is defined analogously). In a job-optimal assignment, each job *simultaneously* receives at least as much of an allocation of its first-choice machine as it could in any feasible stable assignment, and it also receives at least as much of an allocation of its second-choice machine as it could in any feasible stable assignment with the same first-choice allocation, and so on. It is always possible to find a job-optimal feasible stable assignment for any problem instance using a strongly-polynomial algorithm of Baiou and Balinski [1].

2.2 Unsplit Assignment

We now consider the "unsplittable" unconstrained stable allocation problem where each job must be entirely assigned to a single machine. Thus the feasible assignments x are precisely the integral solutions to (1) where either $x_{ij} = 0$ or $x_{ij} = p_i$ for all (i, j). As the following simple example shows, an integral stable assignment may not exist.

Example 1. Suppose there are two jobs i_1 and i_2 with demands 1 and 2 respectively, and two machines j_1 and j_2, both with capacity 2. Let $\pi(i_1) = \pi(i_2) = (j_1, j_2)$ and $\pi(j_1) = \pi(j_2) = (i_1, i_2)$. Then the only stable assignment is $x_{i_1 j_1} = 1$, $x_{i_2 j_1} = 1$, and $x_{i_1 j_2} = 1$, but this is not an unsplit assignment.

We therefore consider a relaxation that is directly analogous to a result of Shmoys and Tardos [15] for the bipartite assignment problem with costs. Assuming existence of a feasible fractional assignment of cost C with all jobs fully assigned, Shmoys and Tardos show how to round this solution in polynomial time to obtain an unsplit solution of cost no more than C where each machine is congested (filled beyond its capacity) by at most $p_{max} = \max_i p_i$. Similar results have been achieved in literature on unsplittable flows (see [7, 2, 16] for more background), where our goal is generally to take a fractional solution to a network flow problem and round it to an unsplit flow (where the flow for each commodity follows a single path) without significantly raising the cost of the flow, and without causing excessive congestion on edges.

Definition 5. *An assignment x is* minimally congested *if for every machine j, removal of the least-preferred job (to j) currently assigned to j results in j being utilized at or below its capacity.*

Note that in a minimally congested assignment, each machine is over-capacitated by at most p_{max}. We show how a modified version of the GS algorithm can find, in polynomial time, a stable unsplit assignment that is *job-optimal* among all minimally congested stable unsplit assignments. Suppose x is a job-optimal feasible stable fractional assignment. We prove that in a job-optimal unsplit assignment, each job is assigned to at least the best of its fractional assignments in x (our analog of the condition that cost does not increase).

Our unsplit assignment is stable in that (i) it admits no blocking pairs and (ii) all popular machines are saturated. Note that one must take some care here with the definition of condition (ii). We define machine j to be saturated with respect to its *original* capacity, c_j, and not the inflated capacity $c_j + p_{max}$ according to which our unsplit solution is feasible, i.e. machine j is *saturated* if $\sum_i x_{ij} \geq c_j$. Otherwise, it might be impossible to satisfy (ii) by ensuring popular machines are saturated — for example, if c_j is odd but all p_i's are even. This definition makes intuitive sense because a machine beyond its capacity will not want any new jobs assigned to it.

3 The Gale-Shapley Algorithm

Gale and Shapley [3] devised a simple intuitive algorithm, now quite well known, for solving the classical "one-to-one" stable marriage problem. The algorithm is usually described in terms of men being assigned to women, although we continue to use job/machine terminology since it is less awkward once we advance to many-to-many matchings. The Gale-Shapley (GS) algorithm has each job i issue "proposals" to machines in the order of i's preference list. Each machine j tentatively accepts the best proposal received so far. If machine j is tentatively matched with job i and receives a more favorable proposal, it tentatively accepts the new proposal and rejects i, which then continues to propose to machines further down on its preference list. Remarkably, it can be shown that regardless of the order in which jobs propose, the GS algorithm always terminates with a job-optimal and machine-pessimal stable matching. Each job receives the most preferred partner it could receive in any stable matching, and each machine receives the least preferred partner it could receive in any stable matching. By symmetry, the reverse is true if the machines do the proposing.

Baiou and Balinski [1] mention that the GS algorithm can be generalized to solve the many-to-many stable allocation problem, although its running time in this case is only pseudo-polynomial. The generalized GS algorithm issues "aggregate" proposals: in each iteration a job i that is not fully assigned issues a proposal to the next machine j in its preference list and proposes all of its unassigned processing time (up to u_{ij}). Machine j accepts only as much as allowed by its capacity, current allocation, and preference list, possibly rejecting (fractionally) some of the jobs already assigned to it if they are less preferred than job i. Whenever a job is "split" due to a fractional acceptance or rejection, it remains split into two "virtual jobs" for the remainder of the algorithm, each of which carries out independent sequences of proposals. Just as with the classical unit stable matching problem, one can show that order of proposals and rejections does not matter — we always obtain a job-optimal feasible stable assignment. A similarly defined algorithm with machine proposals always finds the machine optimal assignment.

Theorem 1. *For any order of proposals, the job-proposing GS algorithm computes the job-optimal fractional stable assignment.*

This theorem follows immediately from the fact that we can interpret the extended GS algorithm for the many-to-many stable allocation problem as nothing more than the standard "one-to-one" GS algorithm applied to an expanded instance where each job i is replaced with p_i unit-sized jobs (each with the same preference list) and each machine j is replaced by c_j unit-sized machines (each with the same preference list). The many-to-many algorithm is sped up by issuing proposals in batches, but it inherets from the one-to-one algorithm the property that the final solution must be job-optimal irrespective of the order of proposals. As an interesting remark, if problem data is irrational, then not only does this reduction to the one-to-one case fail, but it is also not known whether the GS algorithm terminates after a finite number of iterations. We comment on this issue further in the conclusion section.

4 Computing Unsplittable Stable Allocations

In this section we discuss our "ordinal" analog for the stable allocation problem of the result of Shmoys and Tardos for the minimum-cost bipartite assignment problem. Since the constraints $x_{ij} \leq u_{ij}$ do not make sense for an unsplittable stable allocation problem, we henceforth assume we are dealing with an unconstrained stable allocation problem.

Let us modify the GS algorithm as follows. Jobs issue proposals in sequence according to their preference lists, and in each iteration an arbitrary unassigned job i issues a proposal to the next machine j on its preference list. In this case, however, all proposals and rejections are "integral" in that either an entire job is accepted or rejected. Machine j accepts i's proposal, but then proceeds to reject in sequence the least favored jobs assigned to it (possibly including i) until j is at most over-congested by the processing time of a single job — that is, until rejecting the next job would leave the machine being utilized strictly below c_j units of load. Note that such an algorithm results in an assignment where each machine is congested by at most the maximum processing time of a job.

If each machine stores its accepted jobs in a heap based on preference list ranking, this integral variant of the GS algorithm runs in $O(mn \log n)$ time. We now prove some desirable properties of the algorithm. First we show that the assignment output by our algorithm is stable and *job-optimal*. The proof of the following theorem is similar to the traditional proof for the correctness and optimality of the one-to-one GS algorithm.

Theorem 2. *The integral job-proposing GS algorithm computes the* job-optimal *stable unsplit assignment among all minimally congested unsplit stable assignments.*

Proof. Let x^* be the solution output by the GS algorithm. Clearly, x^* is an unsplit assignment that congests each machine by at most p_{max}. Let $x^*(i)$ be the machine to which job i is assigned in x^* and $x^*(j)$ be the set of jobs to which machine j is assigned in x^* (i.e. $x^*(j) = \{i : x^*_{ij} > 0\}$). We also extend the preference notation such that for a set S, $S >_j i$ means $i' >_j i$ for all $i' \in S$ with $i' \neq i$.

We first show that x^* is stable. Suppose not. First note that once a machine is saturated, it never again becomes unsaturated. Thus, every popular machine j must be saturated since if j is popular due to i, then i must have proposed to j at some point and been rejected. This means that the instability in x^* must be caused by a blocking pair. Let (i, j) be a blocking pair. There are two cases. If i never proposed to j, then, since jobs propose in decreasing order of their preference list, $x^*(i) >_i j$ which contradictions the assumption that (i, j) is a blocking pair. On the other hand, if i proposed to j and was rejected, then $x^*(j) >_j i$ since machines only ever improve the set of jobs assigned to them.

We now show that x^* is job-optimal. Suppose not and let i be the first job rejected by one of its stable machines (i.e. a machine assigned to i in some minimally congested stable unsplit assignment), and let j be the first stable machine to reject i. Call the minimally congested unsplit stable assignment in which i and j are matched x. When j rejected i, in the current tentative assignment x', $x'(j) >_j i$ and $\sum_{i' \in x'(j)} p_{i'} \geq c_j$. We now know that there must be some $i' \in x'(j) \backslash x(j)$; if this were not the case and $x'(j) \subseteq x(j)$, then x could not have been minimally congested (removal of job i and all other jobs j prefers less than i would still leave machine j saturated). Since i' has not yet been rejected by a stable machine, and since jobs propose in decreasing order of their preference list, $j >_{i'} x(i')$. But then (i', j) form a blocking pair in x, and so j could not have been a stable machine for i.

We now observe that this solution computed by the integral variant of the GS algorithm assigns each job to at least the best of its fractional assignments in the job-optimal fractional assignment. Thus, the jobs weakly prefer the solution output by the integral variant to the solution output by the fractional variant – i.e. the solution is both integral and lexicographically larger. Our proof uses the fact that the order of proposals does not affect the outcome of the GS algorithm. Thus, we can run the fractional variant of the GS algorithm using the order of proposals induced by the integral variant. During this process, we observe that jobs are assigned to the same machines in both variants. However, the fractional variant may have additional proposals to make after the integral variant completes. As jobs always propose to machines in decreasing order of their preference list, and as the fractional (integral) variant computes the job-optimal fractional (unsplit) stable solution, this coupling of the two algorithms shows that the unsplit solution must be preferred to the fractional solution by each job.

Let $x(i)$ be the set of machines to which i is partially assigned in assignment x, i.e. $x(i) = \{j : x_{ij} > 0\}$.

Theorem 3. *Consider any feasible fractional stable assignment x_{frac} and the* job-optimal *minimally congested unsplit stable assignment x_{int}. Then for all jobs i, $x_{int}(i) >_i x_{frac}(i)$.*

Proof. The proof follows from Theorem 1 and the fact that jobs propose in decreasing order of their preference list (and so as the algorithm runs the jobs' situations worsen). More formally, consider the sequence of proposals defined by the integral GS algorithm. Call this sequence $(i_1, i_2, \ldots, i_l)$ (note this list includes repetitions and l may be greater than n). Run the fractional GS algorithm with the same order of proposals. We prove by induction that after the proposal of i_k, the current assignment x in the integral variant and x' in the fractional variant satisfy $x(j) = x'(j)$ for all j and a machine is saturated in x if and only if it is in x'. This is clearly true after the proposal of i_1. Assume this is the case after the proposal of i_{k-1} and let j be the machine to which i_k proposes. By inductive assumption, j must be the same machine in both the integral and fractional variants of the algorithm. If j rejects i_k in the integral variant, then it must be that $x(j) >_j i_k$ and $\sum_{i \in x(j)} p_i + \geq c_j$. Thus, in the fractional variant, $\sum_{i \in x'(j)} x'_{ij} = c_j$ and $x'(j) >_j i_k$ so all of i_k's load is rejected. A similar argument holds if j rejects i_k in the fractional variant, and so the inductive hypothesis holds.

Therefore, after the l'th proposal in the integral variant, the final solution x_{int} of the integral variant is at least as preferable as the current solution x' of the fractional variant for each job. Furthermore, as jobs propose in decreasing order of their preference list, the final solution x_{frac} of the fractional variant cannot be preferred to the current solution x' by any job. This completes the proof.

We remark that all the theorems in this paper hold if we instead seek the machine-optimal solution. We merely need to run the Gale-Shapley algorithm with machine-proposals – a machine proposes to the next job on its preference list if it is currently under-utilized (it's load is currently less than its capacity). A job (fractionally) accepts a proposal if it is (fractionally) unassigned or if it prefers the proposing machine to (some of) its current machine(s), in which case it rejects (some of) its current machine(s).

5 Conclusion

In this paper, we studied a natural integral variant of the stable allocation problem in which every job was unsplittably assigned and every machine was not excessively congested. Our results have implications for many economic settings where varying sized agents must be matched to each other. Our work leaves open a number of interesting questions:

Rural hospitals: It is well known that in one-to-one matching, the set of singles remains the same in every stable matching. Roth [11] extended this theorem and showed that in one-to-many matching, an agent not fully utilized in a stable matching always receives the exact same assignment in every matching.[2] It seems likely that similar statements might hold in a many-to-many matching as well. It would be interesting to learn whether the same machines are congested in every stable unsplit matching, and if so whether these machines are congested by the same amount in every stable unsplit matching, and/or that the uncongested machines have the same assignment in every stable unsplit matching.

Incentives: Centralized matching algorithms like the one proposed in this paper are often used in economic settings where agents are self-interested and might alter their submitted preference list in order to improve their match. It is known that no stable mechanism can be incentive-compatible for both jobs and machines. In a job-optimal mechanism, for example, machines have an incentive to lie. However, Immorlica and Mahdian [5] showed that, in a one-to-many matching, if preference lists of jobs are short and preferences are drawn randomly according to a particular class of distributions, then each agent has a unique stable partner with high probability, and thus has no incentive to lie. It would be interesting to prove a similar statement in the many-to-many setting studied here.

References

1. M. Baiou and M. Balinski. Erratum: The stable allocation (or ordinal transportation) problem. *Mathematics of Operations Research*, 27(4):662–680, 2002.
2. Y. Dinitz, N. Garg, and M.X. Goemans. On the single-source unsplittable flow problem. *Combinatorica*, 19:17–41, 1999.
3. D. Gale and L.S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69(1):9–14, 1962.
4. D. Gusfield and R. Irving. *The Stable Marriage Problem: Structure and Algorithms.* MIT Press, 1989.
5. N. Immorlica and M. Mahdian. Marriage, honesty, and stability. In *Proceedings of 16th ACM Symposium on Discrete Algorithms*, pages 53–62, 2005.
6. B. Klaus and F. Klijn. Stable matchings and preferences of couples. *Journal of Economic Theory*, 121:75–106, 2005.
7. J.M. Kleinberg. *Approximation algorithms for disjoint paths problems.* PhD thesis, M.I.T., 1996.
8. D.E. Knuth. Stable marriage and its relation to other combinatorial problems. In *CRM Proceedings and Lecture Notes, vol. 10, American Mathematical Society, Providence, RI. (English translation of Marriages Stables, Les Presses de L'Université de Montréal, 1976)*, 1997.
9. E. Ronn. Np-complete stable matching problems. *Journal of Algorithms*, 11:285–304, 1990.
10. A.E. Roth. The evolution of the labor market for medical interns and residents: a case study in game theory. *Journal of Political Economy*, 92:991–1016, 1984.

[2] This is known as the *rural hospital theorem* as it explains why rural hospitals, typically unpopular among students in the NRMP, always receive the same assignment in every stable matching.

11. A.E. Roth. On the allocation of residents to rural hospitals: a general property of two-sided matching markets. *Econometrica*, 54:425–427, 1986.
12. A.E. Roth. The national residency matching program as a labor market. *Journal of the American Medical Association*, 275(13):1054–1056, 1996.
13. A.E. Roth and E. Peranson. The redesign of the matching market for american physicians: Some engineering aspects of economic design. *American Economic Review*, 89:748–780, 1999.
14. A.E. Roth and M. Sotomayor. *Two-Sided Matching: A Study in Game-Theoretic Modeling and Analysis*. Cambridge University Press, 1990.
15. D.B. Shmoys and É. Tardos. Scheduling unrelated machines with costs. In *Proceedings of the 4th annual ACM-SIAM Symposium on Discrete algorithms (SODA)*, pages 448–454, 1993.
16. M. Skutella. Approximating the single source unsplittable min-cost flow problem. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2000.