Let us consider applying this algorithm to a sample multicast spanning tree (Figure 10.8a). In the upward pass (Figure 10.8b), every node \( i \) computes \( m_i \), the marginal value connecting \( i \) to the network, given that its parent is connected. This is the maximum amount the agents on the subtree rooted at \( i \) would be willing to pay to join the multicast. In the downward pass (Figure 10.8c), every node \( i \) computes \( s_j \) for each child node \( j \). \( s_j \) is the actual total surplus generated by connecting \( j \) to the multicast tree. If \( m_j \) or \( s_j \) is negative, the efficient allocation does not connect \( j \). Thus, \( s_j \) can also be seen as the maximum amount by which agents in the subtree rooted at \( s_j \) could reduce their joint value declaration while remaining connected. Each connected node \( j \) is charged \( \max(\hat{v}_j - s_j, 0) \), meaning that his surplus is equal to the amount he could have under-reported his value without being disconnected. These payments are illustrated in Figure 10.8d.

### 10.6.4 Two-sided matching

So far in this chapter we have concentrated on mechanism design in quasilinear settings, meaning that we have assumed that money can be transferred between agents and the mechanism. However, there exist many interesting settings where such transfers are impossible, for example, because of legal restrictions. Examples of such problems include kidney exchanges, college admissions, and the assignment of medical interns to hospitals. **Two-sided matching** is a widely studied model that can be used to describe such domains. Under this model, each agent belongs to one of two groups. Members of each group are matched up, based on their declared preferences over their candidate partners. The mechanism design problem is to induce agents to disclose these preferences in a way that allows a desirable matching to be chosen, despite the restriction that payments cannot be imposed.

We will use the running example of a cohort of graduate students who must align with thesis advisors. Each student has a preference ordering over advisors (depending on their research interests, personalities, etc.), and likewise each potential advisor has a preference ordering over students. In this setting a social choice function is a decision about which students should be assigned to which advisors, given their preferences; as always, the mechanism design concern is how to implement a desired social choice function.

We now define the setting more formally. Let \( A \) be a set of advisors and let \( S \) be a set of graduate students. We do not assume that \( |A| = |S| \); thus, some students and/or advisors may remain unpaired. We assume that each student can have at most one advisor and each advisor will take at most one new student.\(^{13}\) Each student \( i \) has a preference ordering \( \succ_i \) over the advisors, and each advisor \( j \) has a preference ordering \( \succ_j \) over the students. We write \( a \succ_s a' \) to mean that student \( s \) prefers advisor \( a \) to advisor \( a' \), and \( \emptyset \succ_s a \) to mean that \( s \) prefers not finding a supervisor to aligning with advisor \( a \). In the latter case we say that advisor \( a \) is unmatchable to student \( s \). Similarly, we write \( s \succ_a s' \) and \( \emptyset \succ_s a \). Note that we

---

\(^{13}\) Many but not all of the results in this section can also be extended to the case where advisors can take
have assumed that all preferences are strict,\textsuperscript{14} but that each agent can identify a set of partners with whom he would prefer not to be matched, effectively expressing a tie among unacceptable partners. In what follows, we adopt the convention that all advisors are female and all students are male. The resulting problem of finding good male–female pairings pays homage to the problem introduced in the two-sided matching literature a half-century ago, so-called stable marriage.

**Definition 10.6.7 (Matching)** A matching $\mu : A \cup S \rightarrow A \cup S \cup \{\emptyset\}$ is an assignment of advisors to students such that each advisor is assigned to at most one student and vice versa. More formally, $\mu(s) = a$ if and only if $\mu(a) = s$. Furthermore, $\forall s \in S$, either $\exists a \in A, \mu(s) = a$ or $\mu(s) = \emptyset$ (the student is unpaired), and likewise $\forall a \in A$, either $\exists s \in S, \mu(a) = s$ or $\mu(a) = \emptyset$. Note that it is always possible that some student $s$ has the same match under two different matchings $\mu$ and $\mu'$, that is $\mu(s) = \mu'(s)$. In this case, $s$ must be indifferent between matchings $\mu$ and $\mu'$. A similar argument is true for advisors. Therefore, we use the operator $\succeq$ as well as $\succ$ when describing an agent’s preference relation over matchings. More formally, $\mu(s) \succ_s \mu'(s)$ means that either $\mu(s) \succ a \mu'(s)$ or $\mu(s) = \mu'(s)$. Similarly, $\mu(a) \succeq_a \mu'(a)$ means that either $\mu(a) \succ a \mu'(a)$ or $\mu(a) = \mu'(a)$.

Clearly, there are many possible matchings. The key question is which matching should be chosen, given students’ and advisors’ preference orderings. In other words, what properties does a desirable matching have? We identify two.

**Definition 10.6.8 (Individual rationality)** A matching $\mu$ is individually rational if no agent $i$ prefers to remain unmatched than to be matched to $\mu(i)$.

**Definition 10.6.9 (Unblocked)** A matching $\mu$ is unblocked if there exists no pair $(s, a)$ such that $\mu(s) \neq a$, but $a \succ_s \mu(s)$ and $s \succ a \mu(a)$.

Intuitively, a matching is individually rational if no agent is matched with an unacceptable partner; a matching is unblocked if there exists no pair that would prefer to be matched with each other than with their respective partners. Putting these two definitions together, we obtain the concept of a stable matching.

**Definition 10.6.10 (Stable matching)** A matching $\mu$ is stable if and only if it is individually rational and unblocked.

It turns out that in the setting we have defined above, no matter how many students and advisors there are and what preferences they have, there always exists at least one stable matching.

**Theorem 10.6.11 (Gale and Shapley, 1962)** A stable matching always exists.
Step 1: each student applies to his most preferred advisor.
repeat
  Step 2: each advisor keeps her most preferred acceptable application (if any) and rejects the rest (if any).
  Step 3: each student who was rejected at the previous step applies to his next acceptable choice.
until no student applied in the last step

Figure 10.9: Deferred acceptance algorithm, student-application version.

Proof. The proof is obtained by giving a procedure that produces a stable matching given any set of student and advisor preferences. Here, we describe the so-called “student-application” version of the algorithm (Figure 10.9). There is an analogous algorithm in which advisors apply to students. This algorithm must stop in at most a quadratic number of steps, since no student ever applies more than once to any advisor. The outcome is a matching, since at any step each student is paired with at most one advisor and vice versa. The matching is individually rational, since no student or advisor is ever matched to an unacceptable agent.

It only remains to show that the matching is unblocked. Let \( \mu \) be the matching produced by the algorithm. Assume for contradiction that \( \mu \) is blocked by some student \( s \) and advisor \( a \). Since \( s \) prefers \( a \) to his own match at \( \mu \), \( a \) must be acceptable to \( s \), and so he must have applied to her before having applied to his match. Since \( s \) is not matched to \( a \) in \( \mu \), he must have been rejected by her in favor of someone she liked better. Therefore, \( (s, a) \) does not block \( \mu \), a contradiction.

Thus, there always exists at least one stable matching. However, these matchings are not necessarily unique—given a set of student and advisor preferences, there may exist many stable matchings. Let us now consider how different matchings can be compared.

Definition 10.6.12 A stable matching \( \mu \) is student optimal if every student likes it at least as well as any other stable matching; that is, \( \forall s \in S \) and for every other stable matching \( \mu' \), \( \mu(s) \succeq_s \mu'(s) \).

Along the same lines, we can define advisor-optimal matching. Now we can draw the following conclusions about stable matchings.

Theorem 10.6.13 There exists exactly one student-optimal stable matching and one advisor-optimal stable matching. The matching produced by the student-application version of the deferred application algorithm is the student-optimal stable matching, and the matching produced by the advisor-application version of the deferred application algorithm is the advisor-optimal stable matching.

© Shoham and Leyton-Brown, 2008
Next, it turns out that any stable matching that is better for all the students is worse for all the advisors and vice versa.

**Theorem 10.6.14** If $\mu$ and $\mu'$ are stable matchings, $\forall s \in S, \mu(s) \succeq_s \mu'(s)$ if and only if $\forall a \in A, \mu'(a) \succeq_a \mu(a)$.

Say that an advisor $a$ is achievable for student $s$, and vice versa, if there is a stable matching $\mu$ that matches $a$ to $s$. Then we can state one implication of the above theorem: that the student-optimal stable matching is the worst stable matching from each advisor’s point of view, and vice versa.

**Corollary 10.6.15** The student-optimal stable matching matches each advisor with her least preferred achievable student, and the advisor-optimal stable matching matches each student with her least preferred achievable advisor.

Now let us move to the mechanism design question. If agents’ preferences are private information, can we find a mechanism that ensures that a stable matching will be achieved? As is common in the matching literature, we restrict our attention to settings in which neither the agents’ equilibrium strategies nor the mechanism itself are allowed to depend on the distribution over agents’ preferences. Thus, we must rely on either dominant-strategy or ex post equilibrium implementation. Unfortunately, it turns out that stable matchings cannot be implemented under either equilibrium concept.

**Theorem 10.6.16** No mechanism implements stable matching in dominant strategies.

**Proof.** By the revelation principle, if such a mechanism exists, then there also exists a direct truthful mechanism that selects matchings that are stable with respect to the declared preference orderings. Consider a setting with two students, $s_1$ and $s_2$, and two advisors, $a_1$, and $a_2$. Imagine that $s_1$, $s_2$, and $a_1$ declare the following preference orderings: $a_1 \succ s_1, a_2 \succ s_2, a_1$, and $s_2 \succ a_1$. Assume that $a_2$’s true preference ordering is the following: $s_1 \succ a_2, s_2$. If $a_2$ declares the truth, then (1) the setting will have two stable matchings, $\mu$ and $\mu'$, given by $\mu(s_i) = a_i$ for $i \in \{1, 2\}$, and $\mu'(s_i) = a_j$ for $i, j \in \{1, 2\}, j \neq i$, and (2) any stable matching mechanism must choose one of $\mu$ or $\mu'$. Suppose the mechanism chooses $\mu$. Observe that if $a_2$ declares that her only acceptable student is $s_1$, then $\mu'$ is the only stable matching with respect to the stated preferences and the mechanism must select $\mu'$—which $a_2$ prefers to $\mu$. Similarly, we can show that if the mechanism chooses $\mu'$ when the above preference orderings are stated, then in a setting where $a_2 \succ s_2, a_1$ is $s_2$’s true preference ordering, $s_2$ benefits by misreporting his preference ordering. Therefore, declaring the truth is not a dominant strategy for every agent.

Furthermore, it does not help to move to the ex post equilibrium concept, as can be proved along the same lines as Theorem 10.6.16.

*Multiagent Systems*, draft of August 1, 2008
Theorem 10.6.17 No mechanism implements stable matching in ex post equilibrium.

All is not hopeless, however—it turns out that we can obtain a positive mechanism design result for stable two-sided matching. The key is to relax our assumption that all agents are strategic. In our setting we will assume that advisors can be compelled to behave honestly. Under this assumption, it is enough to prove the following result.

Theorem 10.6.18 Under the direct mechanism associated with the student-application version of the deferred acceptance algorithm, it is a dominant strategy for each student to declare his true preferences.

Proof. This proof proceeds by contradiction. Suppose that the claim is not true and, without loss of generality, say that it is not a dominant strategy for student \( s_1 \) to state his true preference ordering. Then, there is a preference profile \( [\sim] = (\sim_{s_1}, \sim_{s_2}, \ldots, \sim_{s_{|A|}}, \sim_{a_{1}}, \ldots, \sim_{a_{|A|}}) \) such that \( s_1 \) benefits from reporting \( \succ_{s_1} \neq \succ_{s_1} \). Let \( \mu \) be the stable matching obtained by applying the student application version of the deferred acceptance algorithm to \( [\sim] \). By Theorem 10.6.13, \( \mu \) is student optimal with respect to \( [\sim] \). Let \( \mu' \) be the stable matching obtained by applying the same algorithm to \( [\sim'] = (\sim_{s_1}', \sim_{s_2}', \ldots, \sim_{s_{|A|}'}, \sim_{a_{1}'}, \ldots, \sim_{a_{|A|}'}) \). Note that except for \( s_1 \), all the other students and advisors declare the same preference ordering under \( [\sim] \) and \( [\sim'] \).

Let \( R = \{s_1\} \cup \{s: \mu'(s) \sim_s \mu(s)\} \) denote the set of students who strictly prefer \( \mu' \) to \( \mu \) (with respect to their declared preferences \( [\sim] \)). Note that we have included \( s_1 \) in \( R \) because, by assumption, \( \mu'(s_1) \succ_{s_1} \mu(s_1) \). Let \( T = \{a: \mu'(a) \in R\} \) denote the set of advisors who are matched with some student from \( R \) under \( \mu' \). In what follows we first show (Part 1) that any advisor \( a \in T \) is matched with an (always different) student from \( R \) under \( \mu \); that is, \( \{a: \mu'(a) \in R\} = \{a: \mu(a) \in R\} = T \). Then (Part 2) we show that there exist some \( a_t \in T \) and \( s_t \notin R \) such that \( (s_t, a_t) \) blocks \( \mu' \) at \( [\sim'] \) and therefore \( \mu' \) is not stable with respect to \( [\sim'] \). This contradicts our assumption that \( \mu' \) is a stable matching with respect to \( [\sim'] \).

Part 1: For any \( s \in R \), let \( a = \mu'(s) \). Stability of \( \mu \) with respect to \( [\sim] \) requires that advisor \( a \) be matched to some student under \( \mu \) (rather than being unpaired), as otherwise \( (s, a) \) would block \( \mu \) at \( [\sim] \); let \( s' = \mu(a) \). If \( s' = s_1 \), then since \( s_1 \) prefers his match under \( \mu' \) to his match under \( \mu \), \( s' \in R \). Otherwise, since (with respect to his preferences declared in \( [\sim] \)) \( s \) strictly prefers \( \mu'(s) \) to \( \mu(s) \), stability of \( \mu \) with respect to \( [\sim] \) implies that \( s' \sim_a s \). Since we defined \( s = \mu'(a) \), thus \( s' \sim_a \mu'(a) \). Then, stability of \( \mu' \) with respect to \( [\sim'] \) implies that \( \mu'(s') \sim_{s'} a \). Since we defined \( a = \mu(s') \), thus \( \mu'(s') \sim_{s'} \mu(s') \) and therefore \( s' \in R \). As a result, we can write \( T = \{a : \mu'(a) \in R\} = \{a : \mu(a) \in R\} \).
10.7 Constrained mechanism design

Part 2: Since every student \( s \in R \) prefers \( \mu'(s) \) to \( \mu(s) \), stability of \( \mu \) with respect to \( \succ \) implies that \( \forall a \in T, \mu(a) \succ a \mu'(a) \). Therefore, during the execution of the student-application algorithm on \( \succ \), each student \( s \in R \) will apply to \( \mu'(s) \) and will get rejected by \( \mu'(s) \) at some iteration. In other words, each \( a \in T \) rejects \( \mu'(a) \in R \) at some iteration. Let \( s_1 \) be (weakly) the last student in \( R \) who applies to an advisor during the execution of the student-application algorithm. This application is sent to \( \mu(s_1) \in T \); let \( \mu(s_1) = a_1 \). By construction, \( a_1 \) must have rejected \( \mu'(a_1) \) at some strictly earlier iteration of the algorithm. Thus, when \( s_1 \) applies to \( a_1 \), \( a_1 \) must reject an application from some \( s_2 \not\in R \) such that \( s_2 \succ a_1 \mu'(a_1) \) (fact 1). Note that \( s_2 \not= s_1 \), since \( s_2 \not\in R \) and \( s_1 \in R \). Since \( s_2 \) applies to \( a_1 \) before he finally gets matched to \( \mu(s_2) \), we have that \( a_1 \succ s_2, \mu(s_2) \). Furthermore, since \( s_2 \not\in R \), we also have that \( \mu(s_2) \succ s_2, \mu'(s_2) \). Therefore \( a_1 \succ s_2, \mu'(s_2) \) (fact 2). Thus, from (fact 1) and (fact 2), \( (s_2, a_1) \) blocks \( \mu' \) at \( \succ \) and \( \mu' \) is not stable with respect to \( \succ \), yielding our contradiction. 

Of course, it is similarly possible to achieve a direct mechanism under which truth telling is a dominant strategy for advisors by using the advisor-application version of the deferred acceptance algorithm.

10.7 Constrained mechanism design

So far we have assumed that the mechanism designer is free to design any mechanism, but this assumption is violated in many applications—the ones discussed in this section, and many others. In particular, often one starts with given strategy spaces for each of the agents, with limited or no ability to change those. Examples abound:

- A city official who wishes to improve the traffic flow in the city cannot redesign cars or build new roads;
- A UN mediator who wishes to incent two countries fighting over a scarce resource to cease hostilities cannot change their military capabilities or the amount or value of the resource;
- A computer network operator who wishes to route traffic a certain way cannot change the network topology or the underlying routing algorithm.

Many other examples exist, and in fact such constraints can be thought of as the norm rather than the exception. How can such would-be mechanism designers intervene to influence the course of events?

In Chapter 2 we already encountered this problem. Specifically, in Section 2.4 we saw how imposing social laws—that is, restricting the options available to agents—can be beneficial to all agents. Social laws played an important coordinating role (as in “drive on the right side of the road”) and, furthermore, in some