
Journal of Heuristics, 5, 29–46 (1999)

Stable Marriage and Genetic Algorithms: A Fertile Union

BRIAN ALDERSHOF aldershof@worldnet.att.net
Graham Capital Management, Stamford, CT 06902

OLIVIA M. CARDUCCI* carducci@lafayette.edu
Department of Mathematics, Lafayette College, Easton, PA 18042-1781

Abstract. We describe a pair of genetic algorithms for solving two stable matching problems. Both stable matching problems we will consider involve a set of applicants for positions and a set of employers. Each applicant and each employer prepares a rank order list of a subset of the actors in the other set. The goal is to find an assignment of applicants to employers in which if applicant a is not assigned to employer b then either a prefers his assignment to b or b prefers its assignment to a. In other words, no applicant/employer pair can both improve their situations by dropping their current assignments in favor of each other. Our goal will be to enumerate the stable matchings.

One of the problems we will consider is the well-known stable marriage problem, in which neither applicant nor employer preference lists are linked. In the other problem, we will allow pairs of applicants who form a couple to submit joint rank order lists of ordered pairs of employers.

Keywords: Stable marriage, stable matching, genetic algorithm

1. Introduction

In this paper we describe our application of genetic algorithms to two distinct stable matching problems, the "stable marriage" and "couples" problems. The stable marriage problem is well-known and algorithms for finding stable matches are relatively easy to implement. The couples problem is significantly more difficult although variants of proposal algorithms used in the stable marriage case are often used to produce stable matches. The genetic algorithms we propose are very general; they can be used, with minor modifications, to solve many different stable matching problems. The genetic operators we introduce as well as the linear-inequality based fitness function can be generalized in rather straightforward ways to a wide range of matching problems including roommate problems, variable group sizes including singles, couples, and larger groups, preference list ties, and others. Our genetic approach to matching problems seems to be new and more general than any existing matching techniques.

Sections 2–4 provide background information on the stable marriage problem, genetic algorithms and the polyhedral characterization of the stable marriage problem. Sections 5–9 provide a description of our genetic algorithm. Section 10 gives the results of applying our genetic algorithm to several example problems.

* Current address: Department of Mathematical Sciences, Muhlenberg College, 2400 Chew St., Allentown, PA 18104. carducci@muhlenberg.edu

2. Stable Matching Problems

The best known stable matching problem is the stable marriage problem. The stable marriage problem is the problem of pairing n women and n men in such a way that no participant has an incentive to switch partners. The men and women each state preferences over the individuals of the opposite sex. The goal is to find a matching with the property that if a and A are matched to each other, but a prefers B to A, then B prefers its mate to a. In other words, B would not be willing to dump its mate for a. If B prefers a to its mate, then (a, B) form an *unstable pair*. Unstable pairs will play a significant role in our genetic algorithms.

Gale and Shapley (1962) provide an algorithm for finding a stable marriage given any set of preferences. In Gale and Shapley's algorithm each man proposes to his most preferred mate that he has not already proposed to. Each woman considers all proposals she has received and rejects all but her most preferred who she keeps on hold. Each man who has been rejected proposes to the next woman on his list. This process continues until either no men are rejected or all men who have been rejected have proposed to every woman on their lists. All women are paired with the man they currently have on hold. A woman who has not received a proposal or a man who has had all of his proposals rejected remains single. Gale and Shapley's article sparked a serious interest among mathematicians in the stable marriage problem and its variants (Gusfield and Irving, 1989, Knuth, 1976, Roth and Sotomayor, 1990).

Unbeknownst to Gale and Shapley, an algorithm equivalent to their algorithm had been in use by the Association of American Medical Colleges to match graduating medical students to hospital residency positions for 10 years (Roth, 1984). Today, the National Resident Matching Program (NRMP) involves roughly 20,000 positions (Roth, 1990). In Gale and Shapley's algorithm, the men propose to the women, so the matching which results is man-optimal in the sense that every man is matched with his highest rated achievable woman (Gale and Shapley, 1962). If the women propose to the men, then the resulting matching would be women-optimal in the same sense. The NRMP finds the hospital-optimal matching.

Originally the NRMP's problem was equivalent to the marriage problem, but overtime real-world considerations have forced them to modify their service. The most significant modification was allowing couples to express their preferences over ordered pairs of positions. We will call this problem the couples problem. This natural attempt to meet the needs of the growing number of couples graduating from medical schools has significantly complicated the NRMP's task. Roth (1984) has shown that some instances of the couples problem have no solutions. (There are no stable matchings.) Moreover, Ronn (1990) has shown that determining whether or not an instance of the couples problem has a stable matching is *NP*-complete.

The NRMP uses a modified version of Gale and Shapley's algorithm to find a matching that guarantees that each student and hospital that is matched is matched with someone from their preference list. (In most applications, preference lists do not include all possibilities.) They then check whether the matching they have found is stable. According to Elliott Peranson of National Matching Services, the NRMP algorithm has found a stable matching every year since they began checking in the late 1970's.

To summarize the results described so far, the stable marriage problem always has a stable matching and we can find a stable matching easily. An instance of the couples problem may or may not have a stable matching, but it appears that the instances faced by the NRMP do have a stable matching and that the NRMP regularly finds one.

At this point, one might ask whether there are other stable matchings and if so, what are the consequences of choosing one stable matching over another. In the stable marriage case, Benjamin, Converse, and Krieger (1995) provide a construction of preference lists that results in $O(2^{2n}\sqrt{2n})$ stable matchings for n hospitals and n students thereby giving an instance of a stable matching problem with an exponential number of stable matchings. However, in the stable marriage problem, all stable matchings leave the same students and hospitals unmatched (Roth, 1984). Thus when the NRMP was essentially solving a marriage problem, choosing a particular stable matching did not entail choosing which students were assigned residency positions and which were not. Unfortunately there are instances of the couples problem in which different stable matchings match different numbers of students (Aldershof and Carducci, 1996). In this case, choosing a particular stable matching may mean choosing which students are assigned.

The genetic algorithms we describe here are designed to enumerate stable matchings. Since our genetic algorithms are heuristic algorithms, we cannot guarantee that they find all stable matchings in all examples. However a result of Roth and Vande Vate implies that, at least in the marriage case, our genetic algorithm will eventually find at least one stable matching (Roth and Vande Vate, 1990).

The problem of enumerating all stable matchings is $\#P$-complete (Irving and Leather, 1986). (Recall, $\#P$-complete problems are "at least as hard" as NP-complete problems.) Nonpolynomial time solutions to the stable marriage enumeration problem are given in (Irving and Leather, 1986, Gusfield, 1987) where they show how to construct a new stable matching from an existing one. There are no such approaches that we are aware of for the couples problem.

3. Genetic Algorithms

Genetic algorithms are stochastic optimization procedures that mimic natural genetic selection. A genetic algorithm begins with a collection of solution strings, a *population*, of possible solutions for a particular problem. These strings, analogous to chromosomes in natural genetic selection, are randomly combined through a process called *mating* and perturbed through a process called *mutation*. The function to be optimized is called a *fitness function*. Just as in natural genetic selection, solution strings that do not have high fitness values are eventually eliminated from the population through a process called *deletion*. Solution strings with high fitness values are replicated, propagate, and, if the algorithm works well, have progeny with even higher fitness values.

Genetic algorithms were introduced by Holland, whose monograph Adaptation in Natural and Artificial Systems (Holland, 1975) is the seminal reference work. For a genetic algorithm to be successful, possible solutions of the problem must be coded so that small segments of the solution string contain useful information about the fitness of the string. The power of genetic algorithms lies in propagating small segments of these representa-

tions that lead to regions of high fitness. Holland named these segments *schemata*. For example, if hospitals are represented with integers and their position in the string represents the student assigned to them then a schema is given by ##2###4##. This schema represents the subset of all orderings that have student 3 assigned to hospital 2 and student 7 assigned to hospital 4. The number signs (#) are "don't care" symbols but we require that each student be assigned exactly once. These schemata are called ordering schemata (*o*-schemata) (Goldberg and Lingle, 1987). The fundamental theorem of genetic algorithms, called the "schema theorem," says that if schema are replicated with probability proportional to their average fitness then above-average schemata will increase exponentially in the population and below-average schemata will be extinguished exponentially. If we think of a schema as describing some region of a solution space, then a genetic algorithm works by taking an exponentially increasing number of samples in regions of a solution space with high average fitness values.

Genetic algorithms work by exploiting "implicit parallelism." Each string in the population is comprised of many schemata. Through mating and mutation, each string represents a parallel processor of multiple schemata. It can be shown that the number of schemata processed by n strings in a successful genetic algorithm is approximately $O(n^3)$ (Goldberg, 1989). This processing leverage is the heart of genetic algorithms. If enough information about the solution space is encoded in schemata to find the maximum, then the information content of n function evaluations is proportional to n^3.

Genetic algorithms have been successfully used to find solutions to the traveling salesman problem (TSP) and other *NP*-hard problems (e.g., Grefenstette, et al., 1985). Our genetic algorithm shares many features with genetic solutions to the TSP. A genetic solution to the TSP involves combining partial routes into a full route. A genetic approach to stable matching combines partial assignments into a complete matching. As will be discussed below, there are several complications inherent in stable matching that are not present in genetic solutions to the TSP. In particular, the objective ("fitness") function in the TSP is simply the route length; a fitness function for the stable matching problem is not nearly as obvious. The mating operator described below is new, although it is based on a mating operator that was invented for the TSP, called "cyclic crossover." Interestingly, cyclic crossover is not a very good operator for the TSP, but the stable matching problem is sufficiently different that our adaptation works quite well. A schema in the TSP is a partial route, but genetic solutions to the TSP always involve mating and mutating complete routes. This seems unavoidable because there is no natural way of leaving a city off the route and penalizing the fitness function appropriately. In the stable matching problem, we can leave any number of firms and workers unassigned and our fitness function is naturally penalized. By deliberately combining these partial assignments, we dramatically increase our probability of finding stable matches and decrease our computation time.

4. Modeling Stability

To implement a genetic algorithm we need an alphabet to encode the partial solutions (chromosomes), a fitness function, a mating scheme, and a mutation scheme. Our choice for the chromosomes and the fitness function can be best understood by expressing the

stable matching problem as the problem of finding a 0-1 matrix that satisfies a set of linear inequalities. Let H be the set of hospital positions to be matched and P the set of students to be matched. If a hospital, say Philadelphia General—General Surgery, has more than one identical position available, H contains $h_1, h_2, \ldots, h_i$ corresponding to the i general surgery positions at Philadelphia General. For $h \in H$ and $p \in P$, let

$$x_{hp} = \begin{cases} 1, & \text{if } p \text{ is assigned to } h \\ 0, & \text{otherwise} \end{cases}$$

Let $j >_h p$ indicate that hospital h prefers person j to person p. In other words, j is higher than p on h's preference list. The notation $i >_p h$ is defined analogously. If h is on p's preference list and p is on h's preference list, then the pair (h, p) is *acceptable*. Let A be the set of acceptable pairs. Let $X = \{x_{hp}\}$, $h \in H$, $p \in P$ be a matrix whose elements are all integer and satisfy inequalities (1)–(5) below. Then X represents a stable marriage (Rothblum, 1992, Vande Vate, 1989).

$$\sum_{p \in P} x_{hp} \leq 1 \quad \text{for all } h \in H \tag{1}$$

$$\sum_{h \in H} x_{hp} \leq 1 \quad \text{for all } p \in P \tag{2}$$

$$x_{hp} \geq 0 \quad \text{for all } h \in H \text{ and } p \in P \tag{3}$$

$$x_{hp} = 0, \quad \text{for all } (h, p) \in (H \times P) \backslash A \tag{4}$$

$$\sum_{j >_h p} x_{hj} + \sum_{i >_p h} x_{ip} + x_{hp} \geq 1, \quad \text{for all } (h, p) \in A \tag{5}$$

(The notation $j >_h p$ is an abbreviation for $\{j \in P : j >_h p\}$. Other notations are analogous.) It is well-known that inequalities (1)–(3) describe a matching. The inequalities (4) ensure that if a student and hospital are matched, then they are an acceptable pair. The inequalities (5) ensure that the solution represents a stable matching. We will generate our chromosomes so that they represent acceptable matchings, that is so they satisfy inequalities (1)–(4). Each remaining constraint that is violated represents an unstable pair; the pair that was used to generate the violated constraint is unstable. Our fitness function is the number of constraints that are satisfied. There are several advantages to this choice of fitness function. First, matchings with high fitness that are not stable have a small number of unstable pairs. This is not critical for the stable marriage problem since we know that a stable matching exists, but is extremely important in variations, like the couples problem, where there may be no stable matching. Finding "nearly stable" matchings may be all that is possible. Second, we know the value of the fitness function at optimality. For a stable matching the value of the fitness function is the number of acceptable pairs. Third, in evaluating the fitness function we identify unstable pairs. We use this information in the adaptive mutations described below.

A similar approach can be used for the couples problem. Let H be the set of hospitals to be matched and $P = S \cup M \cup W$ the set of people to be matched. Note, S, M, and W are disjoint; S represents the set of individual applicants, M and W are individual members of couples with each couple consisting of a person from M and a person from

W. Let $C = \{(m, w) \in M \times W : (m, w)$ is a couple$\}$ be the set of couples among the people to be matched. For $s \in S$ and $h \in H$ if h is on s's preference list and s is on h's preference list, then (h, s) is an acceptable assignment. Similarly, for $(h, k) \in H \times H$ and $c = (m, w) \in C$ if (h, k) is on c's preference list, m is on h's preference list, and w is on k's preference list, then $((h, k), (m, w))$ is an acceptable assignment. Let A_1 be the set of acceptable assignments not involving a couple and let A_2 be the set of acceptable assignments involving a couple. Let $X = \{x_{hp}\}, h \in H, p \in P$ be a matrix whose elements are all integer and satisfy inequalities (6)–(12) below. Then X represents a stable matching (Carducci, 1997, Rothblum, 1992, Vande Vate, 1989).

$$\sum_{p \in P} x_{hp} \leq 1 \quad \text{for all } h \in H \tag{6}$$

$$\sum_{h \in H} x_{hp} \leq 1 \quad \text{for all } p \in P \tag{7}$$

$$x_{hp} \geq 0 \quad \text{for all } h \in H \text{ and } p \in P \tag{8}$$

$$x_{hs} = 0, \quad \text{for all } (h, s) \in (H \times S) \backslash A_1 \tag{9}$$

$$\min\{x_{km}, x_{lw}\} = 0, \quad \text{for all } (k, l, m, w) \in (H \times H \times C) \backslash A_2 \tag{10}$$

$$\sum_{j >_h s} x_{hj} + \sum_{i >_s h} x_{is} + x_{hs} \geq 1, \quad \text{for all } (h, s) \in A_1 \tag{11}$$

$$\sum_{p >_{h_1} m} x_{h_1 p} + \sum_{p >_{h_2} w} x_{h_2 p} + \sum_{(kl) >_c (h_1 h_2)} \min\{x_{km}, x_{lw}\} + \min\{x_{h_1 m}, x_{h_2 w}\} \geq 1,$$
$$\text{for all } (k, l, m, w) \in A_2 \tag{12}$$

(The notation $j >_h s$ is an abbreviation for $\{j \in S : j >_h s\}$. Other notations are analogous.) It is well-known that inequalities (6)–(8) describe a matching. Inequalities (9) and (10) ensure that only acceptable assignments are made. Inequalities (11) and (12) ensure that the matching is stable. Again, we will generate our chromosomes so that they satisfy inequalities (6)–(10). Each remaining inequality that is violated represents an unstable pair and our fitness function is the number of inequalities that are satisfied.

5. Alphabet

The alphabet we use is similar to the one used in genetic solutions to the TSP. In the marriage problem we represent an assignment as a list of hospitals with no hospitals repeated. The kth hospital on the list is assigned student k. Our initial population consists of only acceptable matchings and our mating and mutation operators are designed so that all solution strings we generate represent acceptable matchings.

6. Generating the Initial Population

To avoid complications, we assume that all preferences listed are acceptable pairs. For example, if hospital 5 ranks student 6 then student 6 also ranks hospital 5. This is a simple matter of deleting unattainable hospitals from students' lists and vice versa.

We generate our initial population for the marriage problem by beginning with a random hospital and assigning it a student from its list at random. We then select another hospital at random and assign it a random student from its list. We proceed through all the hospitals this way, except that if all the students on a hospital's list are assigned, we simply leave the hospital unmatched. Of course, this procedure is linear and requires very little computer time even for problems of the size faced by the NRMP.

For the couples problem, we choose students to assign in a random order. We then select hospitals from their lists in a random order, including "unassigned" in the random ordering. We assign the student the first available hospital or unassigned from this random ordering. Of course, the complexity here is linear and thus requires very little computation time.

7. Mating

7.1. Mating Operator

The traditional crossover operator is not useful in solving the stable matching problem. A traditional crossover operator would combine two strings, e.g., ABCDEFG and BCDFGAE, by choosing a cleaving point at random and switching the chromosomes after the cleaving point. Usually, this will not give a legal assignment. For example, it is easy to see that for these two strings any cleaving point gives children with at least one hospital repeated. The problem is to find a mating operator that combines two legal assignments to provide two different and distinct legal assignments.

A similar problem has been studied extensively in finding mating operators for a genetic solution to the TSP. A solution string for the TSP is a list of cities constituting a tour in which no city is visited more than once. A mating operator must combine these lists to produce one or two different lists each giving legal tours. In the traditional TSP, all orderings in which no city is repeated are legal assignments. In the stable matching problem, however, we are concerned not with order but with position. A legal matching must not only assign at most one student to at most one hospital, but also requires that each assignment be acceptable to both the student and the hospital. Most of the mating operators that have been used in the TSP do not fulfill this requirement.

A mating operator that does fulfill the requirement is called *cyclic crossover* (Oliver, Smith, and Holland, 1986). Cyclic crossover produces two children for each set of parents (Parent 1 and Parent 2). In order to guarantee that the children represent acceptable assignments, each hospital will appear in either the position it held in Parent 1 or the position it held in Parent 2. To mate two chromosomes a random starting point on Parent 1 is selected. The starting gene h_1 is copied into Child 1 in the same position as in Parent 1. The gene in the same position in Parent 2 is called the *opposing gene*, h_2. Hospital h_2 cannot appear in Child 1 in the position it held in Parent 2 (h_1 is there), so h_2 is copied into Child 1 in the same position it is found in Parent 1. This yields a new opposing gene h_3 which is in the same position in Parent 2 as h_2 is in Parent 1. We copy h_3 into Child 1 in the position h_3 holds in Parent 1. We continue copying opposing genes into Child 1 until we complete a cycle by finding that an opposing gene has already been copied into Child 1. The remaining genes are copied into Child 2 in the same positions in which they occur in Parent 1. The

remaining positions in both children are filled using the chromosomes in the same positions as they occur in Parent 2. In our implementation, we avoid immediately reproducing the parents by requiring that strings differ at the starting point. An example is given below:

Starting point	Step 1:	Step 2:
Parent 1: ABCDEFG	Child 1: ABCD*F*	Child 1: ABCDGFE
Parent 2: BCDFGAE	Child 2: ****E*G	Child 2: BCDFEAG

In the example above, B is the starting gene and C is the first opposing gene. Thus position two in Child 1 is filled with B and since C appears in position three in Parent 1, position three in Child 1 is filled with C. The next opposing gene is D (position three, Parent 1); D is placed in position four in Child 1 (D's position in Parent 1). The next opposing gene, F, is placed in position six in Child 1; then A is placed in position one in Child 1. The next opposing gene, B, already appears in Child 1 and the cycle is complete. We copy the genes not involved in the cycle (E and G) into Child 2 in the positions they held in Parent 1. The remaining positions are filled with the genes from the corresponding positions in Parent 2.

Cyclic crossover guarantees that legal parents produce legal offspring. A difficulty with this mating operator is that the number of offspring of a set of parents is limited by the number of cycles created by the parents. Hence, mating schemes which entail multiple copies of parents mating produce many duplicate offspring. Even without allowing multiple copies of parents, we found that cyclic crossover generates many duplicates. Prior to the start of the deletion procedure, we eliminate all duplicate chromosomes.

Cyclic crossover works unmodified only with complete matchings. To accomodate unmatched students and hospitals, we introduce an inversion and restart operator. Many of the problems caused by unassignment can be dealt with by simply switching the two parents and restarting the mating process with the last gene encountered. Examples of our inversion/restart cyclic crossover mating operator are given below for unassigned students and unassigned hospitals.

7.1.1. Unassigned Students

If we start with an unassigned student in Parent 1, a cycle ends with either an unassigned student in Parent 2 or a hospital assigned in Parent 2 that is unassigned in Parent 1. For example: (Student 2 is unassigned in Parent 1.)

Starting point	Step 1:	Step 2:
Parent 1: AUCDEFG	Child 1: AUCD*F*	Child 1: AUCDGFE
Parent 2: BCDFGAE	Child 2: ****E*G	Child 2: BCDFEAG

An unassigned student in Parent 2 is somewhat more difficult. If the mating operator encounters an unassigned student in Parent 2 but started with a hospital that is assigned in

both parents it is unclear how to complete the cycle to ensure that a hospital is not assigned twice in an offspring. We solve this problem by using the inversion and restart operator. For example, the following cycle is interrupted when the third opposing gene is unassigned:

```
     Starting point
           |
Parent 1: ABCDEFG
Parent 2: BCDUGAE
```

Step 1: Inversion and Restart

```
     Starting point          Step 2:          Step 3:
           |
Parent 1: BCDUGAE    Child 1: BCDU*A*   Child 1: BCDUEAG
Parent 2: ABCDEFG    Child 2: ****G*E   Child 2: ABCDGFE
```

7.1.2. Unassigned Hospitals

An unassigned hospital in Parent 1 is similar to an unassigned student in Parent 2. We use the inversion and restart operator except that after restart we end the cycle as if we had started with an unassigned student. For example: (Hospital F is unassigned in Parent 1.)

```
     Starting point
           |
Parent 1: ABCDEUG
Parent 2: BCDFGAE
```

Step 1: Inversion and Restart

```
     Starting point          Step 2:          Step 3:
           |
Parent 1: BCDFGAE    Child 1: BCDF*A*   Child 1: BCDFEAG
Parent 2: ABCDEUG    Child 2: ****G*E   Child 2: ABCDGUE
```

7.1.3. Couples

Even if there were not unassigned students or hospitals, cyclic crossover could easily separate a couple. Each time we assign a member of a couple to an offspring, which may happen multiple times in a cycle, we must make sure that this student's partner is also assigned to this offspring. Sometimes this happens simply through completing the cycle. Other times we need to explicitly assign the partner to the same offspring and start a new

cycle. Of course, this new cycle may encounter other couples and require more new cycles and so on. We used recursion to perform this bookkeeping, but other strategies would also work.

7.2. Mating Strategy

We have experimented with several different mating strategies. The natural genetic selection analogy suggests that selective mating can further evolution. Hence, we have implemented stochastic, polygamous mating strategies in which chromosomes have a probability of replicating proportional to some monotonic function of their fitness. In particular, if our initial population contains N (N even) chromosomes, in each generation we conduct $N/2$ rounds of mating. In each round, a chromosome with fitness f has a probability of mating proportional to $\exp[-\alpha \frac{BestFit-f}{BestFit-LeastFit}]$ where *BestFit* is the fitness of the most fit chromosome in that generation and *LeastFit* is the least fit chromosome in that generation. Hence, α is the premium on fitness for mating. Note that if $\alpha = 0$ this mating strategy is simply sampling with replacement and α large implies that more fit chromosomes mate more frequently than less fit chromosomes. Our empirical results suggest that α near 1 provides the highest probability of finding stable matches.

8. Mutations

In keeping with the natural genetic selection analog, mutations should be random perturbations in solution strings. There do not seem to be any compelling reasons, however, not to enhance at least some part of the population by perturbing strings in a direction that is likely to improve their fitness. Suh and Van Gucht have used a local improvement operator to improve the performance of genetic algorithms on the traveling salesman problem (Suh and Van Gucht, 1989).

In the stable matching problem, a violated inequality implies that the hospital-student pair that generated the inequality is an unstable pair. A mutation designed to fix an unstable pair seems likely to result in improved fitness or a useful schema. We call mutations aimed at eliminating unstable pairs *adaptive mutations*.

For the marriage problem, we used both adaptive and random mutations in our algorithm. If we ignore the possibility of unmatched students and hospitals, mutations are fairly simple. For a random mutation, a student is selected at random. Suppose this is student k and he is currently assigned hospital s. The student is assigned a different hospital selected at random from his list, say hospital t. But hospital t may now be assigned to two students, and hospital s is unassigned. We find the student to whom hospital t was assigned and see if he finds hospital s acceptable. If he does, then he is assigned hospital s and the mutation terminates. If not, he is assigned a different hospital selected at random from his list. The process repeats until hospital s is assigned to a student. For an adaptive mutation, we begin by selecting a random unstable pair. Thus we have selected a student k and a hospital t who prefer each other to their current assignments. We assign k to t. As in the random mutation,

t may now be assigned two students and the hospital k was assigned to is unassigned. We resolve this as we did for the random mutations.

The process is slightly complicated by allowing unmatched assignments. First, when a hospital is selected from a student's list, we treat unmatched as if it were another hospital. If at any time in the mutation a student is assigned unmatched, the mutation terminates. Further, if a hospital which was previously unassigned is assigned to a student, the mutation terminates. There are two advantages to using unassignment in mutations. First, the computation time for a mutation is decreased significantly because mutations end whenever unassigned is selected. Second, our empirical results show dramatic improvements in our probability of finding stable matches by introducing unmatcheds in any generation. This seems to be due to the increased ease of developing new schema.

For the couples problem, we used variations of these two mutation operators and added a third. Since we allow the possibility that some students and hospitals will be unmatched in any assignment, there are sometimes available hospitals that acceptable students would prefer to their assignments. Our new mutation simply assigns these hospitals to these students. In the early generations, this has a dramatic effect on the fitness of a few chromosomes but becomes less important in later generations.

Our random and adaptive mutations were performed somewhat differently than in the marriage problem. Because of the couples and the unassigneds, the mutation strategies given in our discussion of the marriage problem were not very practical. For either mutation, we selected a random number of students from a matching. For the examples given below, we selected between 2 and 16 students from the group of 50. We then made all the hospitals these students were assigned available. We then simply reassigned all the students, of course allowing them to become unmatched. The difference between the random mutations and the adaptive mutations was that the adaptive mutations began by reassigning so as to fix an unstable pair. The random mutations just reassigned students and hospitals in a random order.

9. Deletion

The natural genetic selection analog suggests that deletion, at least in the short term, should be stochastic. In nature, fit chromosomes do not eliminate unfit chromosomes immediately but only through long-term enhanced survival. Many different deletion procedures have been proposed and used successfully to mimic this process including tournament selection, roulette wheel selection, probabilistic competition, and others (Goldberg, 1989, pages 121–122). These procedures maintain some diversity in the population by guarding against premature extinction of useful schemata.

After each generation we reduce the population back to its original size. To do this we repeatedly randomly select pairs of chromosomes, with replacement, from the population and have them compete. The loser is removed from the population. If we select two chromosomes with fitnesses f_1 and f_2 then the probability of the chromosome with fitness f_1 remaining in the population is

$$\phi\left(\frac{f_1 - f_2}{\sqrt{2}sh}\right)$$

where ϕ is the standard Normal cumulative probability distribution function, s is the standard deviation of fitnesses in the population and h is our "heat" function. Hence, if h is small there is a premium on fitness during deletion. Our empirical results suggest that h near 0.125 is optimal.

10. Examples

10.1. Example 1

Example 1: 50/50 marriage

We created an example marriage problem containing 50 hospitals and 50 students. Each student chose between 8 and 14 hospitals at random. Each student then ranked its preference list randomly. Each hospital then ranked the students that had chosen them randomly. This resulted in 863 acceptable pairs, and thus 863 inequalities to be satisfied for a stable match.

We experimented with a wide range of population sizes (ranging from 20 to 450) and mutation rates (from 0% to 100%). We did not always get stable matchings. Since the algorithm is stochastic, results varied from trial to trial even when we retained the same parameters. We observed three stable matchings, the hospital-optimal, the student-optimal, and one in between. To verify that we had found all possible stable matchings, we applied the rotation algorithm described in Irving and Leather (1986). In fact the genetic algorithm did find all possible stable matchings in this instance.

Figure 1 shows the results of one run of the genetic algorithm with a population size of 300, an adaptive mutation rate of 20%, and a random mutation rate of 50%. The first stable match occurs at generation 72 and all 3 stable matches appear at generation 81. The average population fitness seems to asymptote at a fitness of 860.

10.2. Example 2

For a somewhat different kind of problem, we used the Benjamin et al. (1995) Latin marriage construction to form preference lists for 8 students and 8 hospitals that have 222 stable matches. More specifically, we used the preference lists given by the DS_4 latin square. The latin square is given in Figure 2. For the given latin square, entry a_{ij} is hospital i's ranking of student j and $7 - a_{ij}$ is student j's ranking of hospital i. Following Benjamin's notation, 0 indicates the best rank and 7 the worst. To find all the stable matchings, we used populations that were significantly larger than the number of stable matchings. A graph of the number of stable matches by generation using a population of size 500 and a mutation rate of 50% is given in Figure 3.

We find it reassuring that the genetic algorithm recovers all of the stable matchings in this problem. Notice that the number of stable matches found grows rapidly in the first few generations reaching, 200 after only 10 generations. The growth rate then becomes much slower as 28 more generations are required to find the remaining stable matches. This is because the necessary schemata for these stable matches are not present in the population and must be created.

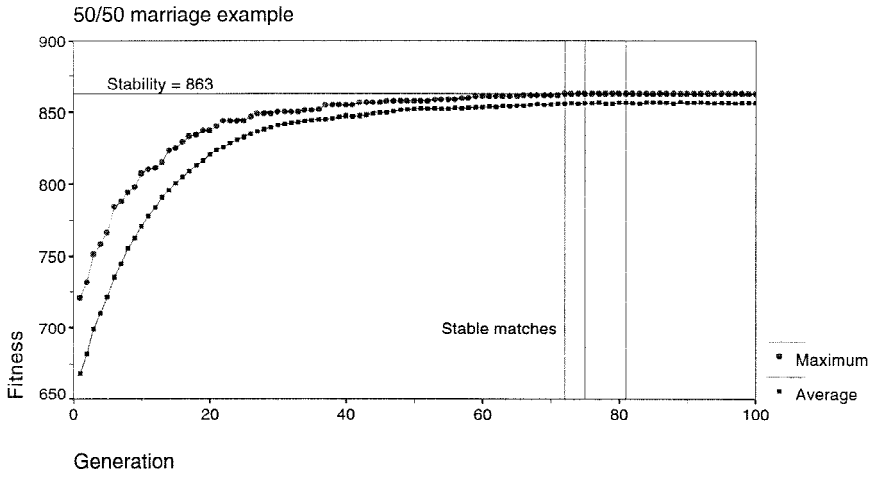

Figure 1. Evolution of fitness.

$$\begin{vmatrix} 0 & 2 & 4 & 6 & 1 & 3 & 5 & 7 \\ 2 & 4 & 6 & 0 & 7 & 1 & 3 & 5 \\ 4 & 6 & 0 & 2 & 5 & 7 & 1 & 3 \\ 6 & 0 & 2 & 4 & 3 & 5 & 7 & 1 \\ 1 & 3 & 5 & 7 & 0 & 2 & 4 & 6 \\ 7 & 1 & 3 & 5 & 2 & 4 & 6 & 0 \\ 5 & 7 & 1 & 3 & 4 & 6 & 0 & 2 \\ 3 & 5 & 7 & 1 & 6 & 0 & 2 & 4 \end{vmatrix}$$

Figure 2. Latin square preference matrix DS_4.

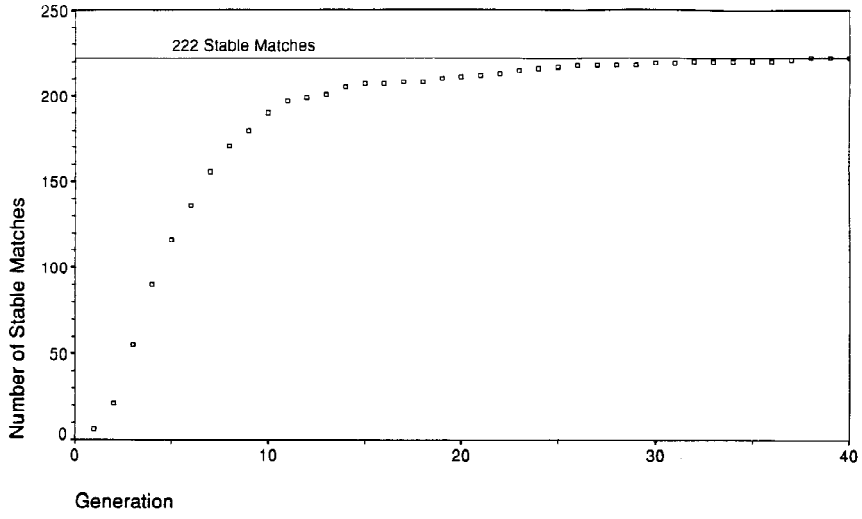

Figure 3. Number of stable matches for latin marriages by generation. *Genepool size = 500, Mutation rate = 50%.*

10.3. Example 3

We developed and tuned our couples algorithm using a modified form of example 1 in which we made the first 10 pairs of students couples. After preprocessing, all the single students had between 11 and 20 hospitals on their lists. In general, the couples had fewer hospitals but between 14 and 22 choices because hospitals were repeated with other choices for their spouse. The hospitals had between 7 and 19 choices on their lists. We have run the genetic algorithm multiple times on these preference lists with many different parameters and populations. We have identified four stable matchings. Without generating all possible legal matchings and checking them (a computationally infeasible task) there is no way to be certain that there are no others. However, if there are other stable matchings they are quite different from the ones we found and very difficult to get to genetically. Since we do not know how to make preference lists with GA-hard stable matchings, we do not think it likely that we could have made one randomly when we generated these.

We have found these stable matchings using populations containing between 150 and 500 matchings. There seems to be some positive correlation between population size and probability of finding stable matchings but it is not as strong as might be supposed. Further, we usually find the first stable match after 60 generations or so. We run the algorithm for 400 generations. It seems, however, that if we have not found all stable matches by 200 generations then they are not found. With populations of size 500, we find stable matches nearly every time and all four stable matches about 75% of the time.

Students' Lists																				
1	3	8	4	8	2	2	3	*U*	2	10	[10]^{1,3}	7	[7]^{2,4}	4	8	10	7			
2	1	10	3	7	3	*U*	*U*	1	1	8	[7]^{1,3}	8	[10]^{2,4}	1	*U*	*U*	*U*			
3	[12]^{3,4}		11	12	8	9	*U*	*U*	8	7	11	12	7	8	[U]^{1,2}					
4	[14]^{3,4}		14	11	9	6	13	11	6	6	13	13	9	*U*	[6]^{1,2}					
5	3	7	1	3	7	3	*U*	*U*	[1]	*U*	6	1	9	1	6	7				
6	2	9	5	5	8	4	4	2	[U]	5	8	2	8	4	9	*U*				
7	1	2	1	5	5	*U*	2	13	15	15	15	[13]								
8	2	5	5	2	*U*	2	*U*	14	13	14	*U*	[U]								
9	4	5	5	8	*U*	[4]	*U*	4	8	9	8	9	*U*							
10	2	1	2	9	2	[U]	1	1	10	10	*U*	*U*	10							
11	10		[17]		1		7		12		8	2	4	6	11					
12	7		[11]		10		1		14		4	3	5	17						
13	[5]		16		14		11		1		2	9	15	4	7					
14	3		17		8		19		[9]		13	18	1	5						
15	[6]^{3,4}		11		2		19		[14]^{1,2}		1	13	17	7	20	3	9	18	8	12
16	[8]		1		18		12		19		15	17	13	4	9	6	16			
17	[15]		6		5		7		13		14	19	3	18	4	20				
18	8		[3]		9		7		12		5	16	13	4	1	19	18	2	15	6
19	3		2		[19]		7		4		11	14	5	9	13	8	10	6	17	12
20	[2]		18		15		20		3		8	6	16	11	4	17	9	19		

Table 1.

10.4. Example 4

It is rather difficult to present the fifty/fifty example given above, so we also tested our algorithm on several examples containing twenty students and twenty hospitals. One set of preference lists are given in Table 1. These lists apparently have four stable matchings. The stable matchings are given in Table 2. Pairings that form the stable matchings are also boxed in Table 1. If a pairing does not occur in all stable matchings it is given a superscript to indicate the matching in which it occurs.

There are several interesting features to this example. First, it is clear that singles are much more likely to get a choice high on their lists than couples are. Further, couples are far more likely to have one spouse unmatched than a single is to be unmatched. This is quite typical in the examples we generated simply because it is more difficult to accomodate a couple than a single. This represents quite a hardship for couples. They must interview at more hospitals, make strategic decisions with incomplete information, make trade-offs between the goals of either partner, and probably still end up with a choice far down their list.

Two of the stable matchings had three students and hospitals unmatched and two stable matchings had four unmatched. For most students and hospitals the stable matchings were the same, however five students (couple 1 consisting of students 1 and 2, couple 2 consisting of students 3 and 4, and student 15) and five hospitals(6, 7, 10, 12, and 14) received different

Hospitals' Lists												
1	13	12	[5]	10	7	15	16	14	11	2	18	
2	13	18	11	[20]	6	8	7	10	19	15	1	
3	12	[18]	2	20	17	15	14	5	1	19		
4	17	[9]	11	20	6	16	12	13	18	1	19	
5	12	17	8	[13]	6	9	7	18	14	19		
6	18	[4]^{1,2}	17	19	11	16	20	5	[15]^{3,4}			
7	11	18	[1]^{2,4}	[2]^{1,3}	5	17	3	13	12	19	15	
8	9	[16]	14	18	11	3	20	1	15	19	6	2
9	13	16	18	[14]	3	6	10	15	9	5	20	19	4
10	10	[1]^{1,3}	12	[2]^{2,4}	19	11						
11	20	[12]	15	11	3	19	13	4				
12	15	11	[3]^{3,4}	16	18	19	[U]^{1,2}					
13	[7]	16	4	15	8	17	19	18	14			
14	[15]^{1,2}	12	19	[4]^{3,4}	17	8	13					
15	13	16	20	[17]	7	18						
16	18	16	13	20	[U]							
17	19	15	[11]	16	12	20	14					
18	14	17	18	20	16	15	[U]					
19	[19]	18	14	20	17	16	15					
20	20	15	17	[U]								

Table 1. Continued.

Student	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Stable 1	10	7	U	6	1	U	13	U	4	U	17	11	5	9	14	8	15	3	19	2
Stable 2	7	10	U	6	1	U	13	U	4	U	17	11	5	9	14	8	15	3	19	2
Stable 3	10	7	12	14	1	U	13	U	4	U	17	11	5	9	6	8	15	3	19	2
Stable 4	7	10	12	14	1	U	13	U	4	U	17	11	5	9	6	8	15	3	19	2

Table 2.

assignments in different stable matchings. The students agree that stable #3 is best, but there is no matching that all the hospitals agree is best. Thus this example has a student-optimal match, but no hospital-optimal match. These differences again underline the need for an algorithm that generates multiple stable matchings and some fixed criteria for deciding among them. We like stable #3 the best because all five students and two of the five hospitals receiving different assignments receive their best achievable assignment in stable #3.

11. Directions for Future Work

We have also programmed and experimented with steady-state GA's. A steady-state GA involves simultaneous matings, mutations, and deletions. A steady-state GA is certainly faster (particularly with twenty computers on a network working simultaneously on the problem) than the generational GA's discussed here, although we have not made direct comparisons (our code has evolved in other ways as well).

It is difficult to gauge the success of our algorithm for the couples problem because we do not have a technique for enumerating stable matchings for the couples problem, short of generating all stable matchings and testing them. We are interested in knowing how to construct examples of couples problems which have known numbers of stable matchings to serve as test problems.

We plan to apply the genetic algorithm approach to the roommate problem with ties. Modifying this genetic algorithm to solve the roommate problem with ties will require a new way of constructing genes which will also mean modifying the other aspects of the algorithm. We should be able to base our fitness function on a set of linear inequalities, however.

12. Conclusion

In this paper we provide a genetic algorithm approach to the stable marriage problem and to the couples problem. An interesting facet of these algorithms is the application of the polyhedral characterization of stable matching problems to our fitness function. The genetic algorithm is expected to generate many (most?) stable matchings. The implicit parallelism of the genetic algorithm makes this possible. We feel that genetic algorithms will prove useful in solving related matching problems.

The examples above are obviously much smaller than most real-world problems. Further, most real-world problems involve preference lists that are very short relative to the number of available choices (e.g., a medical student might rank only a dozen hospital positions from 20,000 possibilities). We have simulated some real-world problems involving as many as 2000 students and hospital positions. To solve these problems we first pre-process the preference lists using an algorithm described in (Aldershof, Carducci, Lorenc, 1998). The pre-processing simplifies the preference lists and usually results in many forced pairings. After the pre-processing, we continue with the genetic algorithm as described above. A genetic enumeration of matchings in a problem of size 2000 can be accomplished in a couple of hours. There is no reason we could not use the same approach to solve problems of the size of the NRMP matching except for some memory limitations of the programming language we used.

In 1996, the NRMP commissioned a study comparing the algorithm they used at the time (an algorithm believed to favor the hospital-optimal match) and a similar algorithm that would favor the student-optimal match. They found that in the majority of the matches studied both algorithms yielded the same stable match. When differences occurred, they involved fewer than 5% of the participants. (This appears to be common in matches in which participants rank only a small fraction of the participants available to them (Lorenc, 1997).) Thus, the NRMP does not feel that is is important to enumerate stable matches in their case. Interestingly, they did decide to change the algorithm believed to favor the student-optimal match (Roth and Peranson, 1997).

13. Acknowledgment

The authors would like to thank Asela Gunawardana for introducing them to genetic algorithms.

References

Aldershof, B., and O. M. Carducci. (1996). "Stable Matchings with Couples," *Discrete Applied Mathematics* 68, 203–207.

Aldershof, B., O. M. Carducci, and D. C. Lorenc. (1998). "Refined Inequalities for Stable Marriage," submitted for publication.

Benjamin, A. T., C. Converse, and H. A. Krieger. (1995). "How Do I Marry Thee? Let Me Count the Ways," *Discrete Applied Mathematics* 59, 285–292.

Carducci, O. M. (1997). "Linear Programming and the Couples Matching Problem," in preparation.

Gale, D., and L. S. Shapley. (1962). "College Admission and the Stability of Marriage," *American Mathematical Monthly* 69, 9–15.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.

Goldberg, D. E., and R. Lingle. (1987). "Alleles, Loci, and the Traveling Salesman Problem." In *Proceedings of an International Conference on Genetic Algorithms and Their Applications.*

Grefenstette, J., R. Gopal, B. Rosmaita, and D. Van Gucht. (1985). "Genetic Algorithms for the Traveling Salesman Problem." In *Proceedings of an International Conference on Genetic Algorithms and Their Applications.*

Gusfield, D. (1987). "Three Fast Algorithms for Four Problems in Stable Marriage," *SIAM Journal on Computing* 16, 111–128.

Gusfield, D., and R. W. Irving. (1989). *The Stable Marriage Problem: Structure and Algorithms*. MIT Press.

Holland, J. H. (1975). *Adaptation in Natural and Artifical Systems*. University of Michigan Press.

Irving, R. W., and P. Leather. (1986). "The Complexity of Counting Stable Marriages," *SIAM Journal of Computing* 15, 655–667.

Knuth, D. E. (1976). *Marriage Stables*. Les Presses de l'Universite de Montreal.

Lorenc, D. C. (1997). "Entry Level Employment Markets and Order Preserving Strategies," Undergraduate Thesis. Lafayette College, Easton, PA.

Oliver, I. M., D. J. Smith, and J. R. C. Holland. (1986). "A Study of Permutation Operators on the Traveling Salesman Problem." In J. J. Grefenstette (ed.), *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*. New Jersey: Lawrence Erlbaum Associates, Publishers.

Ronn, E. (1990). "*NP*-Complete Stable Matching Problems," *Journal of Algorithms* 11, 285–304.

Roth, A. E. (1984). "The Evolution of the Labor Market for Medical Interns and Residents: A Case Study in Game Theory," *Journal of Political Economy* 92, 991–1016.

Roth, A. E. (1990). "New Physicians: A Natural Experiment in Market Organization," *Science* 250, 1524–1528.

Roth, A. E., and M. A. Oliveira Sotomayor. (1990). *Two Sided Matching: A Study in Game-Theoretic Modeling and Analysis*. Cambridge University Press.

Roth, A. E., and E. Peranson. (1997). "The Effects of the Change in the NRMP Matching Algorithm," *Journal of the American Medical Association* 278, 729–732.

Roth, A. E., and J. H. Vande Vate. (1990). "Random Paths to Stability in Two-Sided Matching," *Econometrica* 58, 1475–1480.

Rothblum, U. G. (1992). "Characterization of Stable Matchings as Extreme Points of a Polytope," *Mathematical Programming* 54, 57–67.

Suh, J. Y., and D. Van Gucht. (1987). "Incorporating Heuristic Information into Genetic Search." In J. J. Grefenstette (ed.), *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms.* New Jersey: Lawrence Erlbaum Associates, Publishers.

Vande Vate, J. H. (1989). "Linear Programming Brings Marital Bliss," *Operations Research Letters* 8, 147–153.