# On Finding a Stable Roommate, Job, or Spouse: A Case Study Crossing the Boundaries of Computer Science Courses

Owen Astrachan
Department of Computer Science
Duke University
Durham, NC 27706
ola@cs.duke.edu

## Abstract

The use of real-world problems as the basis for assignments in Computer Science courses is attractive for many reasons. At the same time it is difficult to find such a problem that offers the same richness that is found, for example, in sorting or searching. In this paper a problem is presented that has many real-world instances and which is pedagogically attractive at all levels of Computer Science from the level of a non-major's course to that of an advanced algorithms course.

## 1 Introduction

Sorting and searching have proven to be effective problems for use in Computer Science courses for many reasons. They are easily coded by students in a CS1 course (but see [Pat88] for how easy it is to get binary search wrong) and are easily analyzed by students in introductory courses while providing material for advanced courses as well. There is a wealth of material on both problems ranging from the coverage given in most textbooks to the encyclopedic reference of [Knu73]. We tend to view sorting and searching as easy to motivate for our students in addition to being such a rich source of material. What teacher of Computer Science has not either literally or figuratively given students a phone book and asked for both the number of Mary Smith and for the owner of the number 555-1234?

Yet many students see these problems as contrived. In addition, many colleges and universities are beginning to offer courses to non-majors that have a programming component (e.g., [DH91], [Bie91]).

For students in such a course it is harder still to motivate programmed solutions to sorting and searching problems. With these factors in mind and primed by [Pat91] to find a real-world problem, we have uncovered a problem whose genesis is in the real world but whose abstractions are rich with the potential for some deep analysis. This is a problem whose solution is readily understood by beginning programmers but which offers the opportunity for students in advanced courses to develop significant improvements. In short, it is a problem worthy of study at many levels.

After describing the problem, we give some indications of its successful use in a variety of courses. Space constraints prevent us from developing the problem as fully as is done in our courses. Our intent here is to give a flavor for some of the different uses of the problem and to provide an incentive for further investigation of the problem.

## 2 The Problem

Medical students in their fourth year of medical school undergo a process known as *The Match*. Since this process will effectively decide where each student spends from three to seven years in a residency program, it is approached with great trepidation. This trepidation is heightened by the fact that the methodology in Figure 1 accurately describes the process.

What causes the trepidation is that the pairings output by the computer program *constitute a legal obligation on the part of a medical student*. Each student is given one hospital and the student must serve a residency at this hospital. Knowledge of the algorithm used to generate the pairings is obviously a key factor in determining how a resident should generate a list of acceptable or desirable hospitals.

1. Each student ranks those hospitals at which a residency is desired (usually about 15 such hospitals).
2. Each hospital ranks those students to whom a residency might be offered (a hospital with $n$ residencies typically ranks more than $n$ students).
3. These ranking lists serve as input to a computer program. The output of the program is a pairing of hospitals to students.

Figure 1: Methodology for *The Match*

## 2.1 Similar Problems

This Hospitals/Residents problem has a long history whose telling ([Rot84], [GI89]) is interesting in and of itself. It has many counterparts from Computer Science and Game Theory which serve to further anchor it in the real-world. The problem was first brought to the attention of the academic world in Gale and Shapley's paper [GS62] in the form of the problem of college admissions (although the Hospitals/Residents solution pre-dates this by roughly ten years.) In Computer Science this is the problem of finding a perfect matching in a bipartite graph [GJ79] and is known as the *stable marriage* problem (for reasons described later.) Two books describe it in depth from different points of view [GI89], [Knu76]; the former covers more recent results and is written more with implementation techniques in mind than the latter which uses the problem as the basis for studying several techniques of algorithm analysis.

## 2.2 Stable Matchings

The algorithm actually used in the Hospital/Residents problem [Nat91], [Rot84] has the property that it generates a *stable* matching. The matching is stable because no two residents can both improve their selections by switching hospitals (a symmetric property holds for the hospitals.) If the matching was not stable there would be little incentive for medical students to go through the match.

The stable marriage problem is an abstraction of the Hospitals/Residents problem that is easier to reason about. In the stable marriage problem $n$ men and $n$ women each rank order all members of the opposite sex. Based on these rankings a pairing of women to men is derived that is stable, i.e., no man and woman would both be happier with each other than the spouse received in the stable match. As an example, consider the rankings of four men and four women as shown in Figure 2.

| man | rankings | | | | woman | rankings | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 4 | 1 | 3 | 1 | 2 | 1 | 4 | 3 |
| 2 | 3 | 1 | 4 | 2 | 2 | 4 | 3 | 1 | 2 |
| 3 | 2 | 3 | 1 | 4 | 3 | 1 | 4 | 3 | 2 |
| 4 | 4 | 1 | 3 | 2 | 4 | 2 | 1 | 4 | 3 |

Figure 2: An instance of the marriage problem

An unstable match is given by the pairs (1,1), (3,2), (2,3) (4,4) where a pair consists of a woman's number followed by a man's number. This match is unstable since woman 4 prefers man 1 to her given partner (4) and man 1 prefers woman 4 to his given partner (1). A stable match is given by the pairs (1,4), (2,3), (3,2), (4,1).

If instead of $n$ men and $n$ women we have $2n$ people who desire to find roommates among each other than we have the *stable roommates* problem.

Although each of these three problems is similar and they all share some characteristics, each of them differs from the others in interesting ways.

## 2.3 The Hospitals/Residents Algorithm

Since this problem is to be studied by students in a class for non-majors with little programming expertise, and since it is important that all students in introductory courses have a laboratory experience [Tuc91], we have developed a solution that is easy to understand because it hides unnecessary details in code to which students do not need access. We agree with the methods espoused in [Pat91] that low-level detail should be put into callable, but separately compilable modules. We use a version of Pascal that supports separately compilable units; Modula-2 and Ada also provide such constructs.

The complete program consists of approximately 200 lines of code; we include the main loop of the program in Figure 3 (lifted verbatim from the runnable code) to illustrate both the actual algorithm used [Rot84] and that our neophyte students can understand the algorithm from the code. (Notes about the style of this code are included in an appendix.)

## 2.4 Other sources of information

An informal survey of several data structures and algorithms texts shows that only a few cover the stable marriage problem [Kor86], [Sed90], [Wir76]. Sedgewick briefly mentions the relationship to the Hospitals/Residents problem, but none of the books either gives or discusses solutions to this version of the problem. McVitie [MW71] discusses the stable marriage problem and gives code for generating all stable matches

```
procedure MakeMatch(var hospitalList : HospitalListType;
                    var residentList : ResidentListType);
var
    currentHospital, resident : integer;
begin
    while PositionsExist(hospitalList) do begin               {still unfilled spots?}
        currentHospital := OpenHospitalIndex(hospitalList);   {find needy hosp.}
        resident := NextPotentialMatch(currentHospital);
        if IsAlreadyMatched(resident) then begin
            if Prefers(resident,currentHospital) then begin   {re-assign resident}
                UnAssign(resident);
                TentativelyAssign(resident,currentHospital);
             end
        end
        else begin
            TentativelyAssign(resident,currentHospital);      {assign first time}
        end;
    end;
end;
```

Figure 3: Hospitals/Residents main loop

in addition to solving the stable marriage problem. The brief paper by Quinn [Qui85] discusses parallel algorithms for solving the stable marriage problem.

## 3 The non-major's course

In this section we discuss those aspects of the problem that have proven effective in our course for non-majors. Most of this material is also covered in our first two courses for majors when we discuss this problem. The problem is particularly attractive for non-majors as we can point out two references ([Rot84], [GS62]) from journals in areas in which many of our students do major.

There are many interesting properties of stable matchings and the Hospital/Residents algorithm as it is given in Figure 3. A partial list of those properties that have been effective in generating interest is given in Figure 4. Proofs of these properties or "theorems" can be found in [GI89].

### 3.1 Using the problem

Programming is an integral part of our non-major's course, but we cover only a subset of Pascal and we expect our students to be more facile at reading and modifying existing code than in developing their own programs. The first laboratory exercise using the Hospital/Residents program entails running the program on several different inputs. Students are given questions to ascertain that they can interpret the output and that they can generate input files in the proper format.

Students are then asked to reason about the problem in several ways. The order in which hospitals are considered is changed in two ways: (1) changing the physical order of the hospitals in the input file and (2) changing the body of the function `OpenHospitalIndex`. Students are asked to make generalizations and, as a result, the first theorem in Figure 4 is discovered.

Finally, students note that it is not always the case that each resident is offered a position (or that each hospital fills its positions.) This is in contrast to the stable marriage problem for which a stable match can always be found. Students are asked to reason about the differences in the problems and offer possible explanations.

This problem also lends itself to the study of many ethical questions. In light of the the third theorem of Figure 4 it is particularly interesting that the literature given to medical students [Nat91] assures them that "You will be matched with your highest ranked hospital that offers you a position." Resolving these two statements leads to many interesting discussions.

It is also interesting to note that it is possible for a hospital to receive a better group of interns by misrepresenting its true choices. This situation is asymmetric to the resident's position and does not reflect the analogous situation in the stable marriage problem in which lying does not yield an improvement for the sex whose role corresponds to the hospitals' role above.

1. The order in which each hospital's resident list is considered does not affect the final match (exactly which hospital index is returned by the function `OpenHospitalIndex` as shown in Figure 3 is unimportant as long as the hospital has unsubscribed residency positions and has not exhausted its list of acceptable candidates.)
2. If some hospital does not fill all its positions (remains undersubscribed) then this hospital will receive exactly the same residents no matter what algorithm is used as long as a stable match is generated.
3. For any resident, the hospital assigned by the algorithm in Figure 3 is the lowest ranked hospital that could be part of some stable match.

Figure 4: Theorems of the Hospital/Residents Problem

Finally, we return to this problem when we discuss the concept of tractable and intractable problems in this class. The stable marriage problem remains solvable in polynomial time even if ties are permitted in rankings. This is in sharp contrast to the stable roommates problem in which allowing ties in rankings changes the complexity of the problem from polynomial to NP-complete.

## 4 The first course for majors

In our first course for majors (CS1) we expect a significantly larger programming effort from the students. For example, we do not cover records in our non-major's course so that most of the code that manipulates arrays of records is relegated to a unit and not available for perusal. In our CS1 course we would adopt the same approach initially, but allow the students access to the unit after we have covered the appropriate material.

This allows us to consider an easy modification of the program which changes the complexity of the implementation from $O(n^3)$ to $O(n^2)$ (where we assume for simplicity that there are both $n$ hospitals and $n$ residents.)

The current implementation of the boolean function `Prefers` determines whether a resident prefers a hospital to the resident's currently assigned hospital by searching the resident's list of hospitals for both and then comparing the result. We ask our students to change this linear implementation to one that works in constant time by constructing a preference matrix $P_r$ with the property that $P_r[i,j]$ is the ranking that resident $i$ gives to hospital $j$ (the procedure `TentativelyAssign` must be changed to use the matrix as well, and another matrix $P_h$ which gives the ranking that hospital $i$ gives resident $j$ constructed.) The construction of these matrices is easily done during the initialization of the other data structures in the program. This problem allows the students to reason about the classic time/space tradeoff since $O(n^2)$ storage is needed for the matrix.

The current program also uses a predefined list unit that we ask our students to examine and modify. We give them a unit that implements a list as a record with two fields: an array and a count of the number of elements in the list. We ask our students to change the implementation to one using linked lists. The use of separate units for this is essential and our students learn from experience that modularity makes programming simpler and more adaptable to other tasks.

Finally, reasoning about correctness is an important part of our first course. We ask our students to reason about why the while loop in procedure `MakeMatch` must terminate (the key is the function `NextPotentialMatch` which has the "fortunate" side-effect of moving down a hospital's list of residents.) Since we have, at this point, spent a great deal of time discussing why functions should not have side-effects, we ask our students to change the solution so that it conforms to the guidelines we have developed in class.

## 5 The second course for majors

Many interesting data structures and algorithms for both the stable marriage problem and the Hospitals/Residents problem are covered in detail in [GI89]. We use some of these in our second course when we develop elementary graph algorithms and data structures. We also discuss lower bounds for the first time and investigate the $\Omega(n^2)$ lower bound for the stable matching problem (this proves to be a lower bound even if we ignore the initialization of the data structures which is clearly an $\Omega(n^2)$ operation.)

As an example of the types of interesting problems we examine in this course let us consider the total number of stable matches (i.e., not just the one generated by the algorithm shown in Figure 3) for a given problem instance. It can be shown that for any $n \geq 0$, where $n$ is a power of two, that there is a problem instance of size $n$ with at least $2^{n-1}$ stable matchings. Given this exponential bound it is quite interesting that a polynomial algorithm — $O(n^5)$ — exists for constructing a compact representation of all the stable matchings. While this is at first glance disturbing, we remind our students that as an example of recursion we have investigated an algorithm for generating the power set of a given set. This algorithm is, in fact, a compact (and polynomially constructible) representation of an exponential number of things.

This representation allows us to develop polynomial algorithms for the stable marriage problem that generate *egalitarian matches*. Recall the third theorem of Figure 4 as it relates to marriages. As noted above, if the algorithm outlined in Figure 3 is used a stable match is generated that is optimal for hospitals. We discuss this optimality as it relates to the marriage problem and reason about "male-optimality" versus "female-optimality". Finally, we note that an egalitarian match treats both sexes equally. It is interesting to note that although a sex-biased optimal matching can be constructed in $O(n^2)$ time, the egalitarian algorithm is $O(n^4)$.

## 6 More advanced courses

There are many modifications of the stable marriage problem amenable to analysis and study in a more advanced algorithms course. The egalitarian match discussed in the context of our second course for majors is generated using network flow algorithms (again we usually hide these details in that course.) The egalitarian matching algorithm can be generalized to an optimal matching algorithm. The input to the optimal matching algorithm is a sequence of preference lists in which each person provides a real number ranking for the elements on the preference list rather than just a rank ordering. This allows a finer granularity in preference lists and permits different optimizations (e.g., we can either maximize or minimize the matches according to how the real number weights are assigned.) This optimization problem admits an $O(n^4 \log n)$ solution.

As mentioned previously, we do discuss tractable and intractable problems in our earlier courses, but only on a superficial level. In later algorithms or theory of computation courses we first discuss in depth the concepts of NP-completeness, reductions, and how this problem relates to them. For example, as noted above allowing ties in the ranking of the stable roommates problem makes that problem NP-complete. This is shown by a reduction from 3-SAT [GI89], [GJ79]. In addition, although the stable marriage problem admits a polyomial solution, and an efficient representation of all stable matchings can be constructed in polynomial time, the problem of determining the number of stable matches is #P-complete [GJ79]. This means that determining the number of stable matches is hard (and may be hard even if P=NP) in contrast to the decision problem of determining if there is a stable match (the answer is always yes!), or the decision problem of determining if a given pair is part of a stable match (which admits an $O(n^2)$ solution.)

Finally we note that the three dimensional matching problem, which corresponds to a stable marriage in which there are three different sexes (perhaps on some other planet?), is NP-complete. This serves to reinforce the idea that higher dimensionality often affects the complexity of a problem (consider 2-SAT and 3-SAT for example.)

## 7 Summary

We hope that we have offered an incentive for studying the stable marriage problem in its many forms in courses that cover a broad spectrum of Computer Science. Our students have found it interesting and we have been quite pleased with how versatile it is, generating interesting problems that can be discussed by non-majors in their first course and by graduate students in an algorithms course.

We close with a somewhat amusing anecdote relating Santayana's warning of what happens to those who choose not to study the history of the Hospitals/Residents problem.

It appears that federal judges might be persuaded to be part of a clerkship-match in order to forego the kind of experiences reported in [Mar] and partially reproduced below:

> In their eagerness to capture the best clerks, the judges have steadily pushed up the hiring process; instead of looking for students in their third year of law school as custom once required, judges surreptitiously began recruiting second-year students in fall and offered some jobs as early as February, disrupting studies and making decisions on the basis of fewer grades and flimsier evidence.
>
> "It was positively surreal, the most ludicrous thing I've ever been through," said one Stanford student who recently endured the process.
>
> ...years ago, for instance, Judge Winter offered a Yale student a clerkship at 11:35 and gave her until noon to accept. At 11:55, as she was trying to reach a California judge to whom she had also applied, he withdrew his offer.

## Appendix

Space considerations prevent the inclusion of the code we use for this problem. We will be happy to send it electronically to any who request it. Alternatively, we will return any Macintosh or IBM-compatible (5.25 or 3.5 inch) disk with the complete code if a return mailer is provided.

The code given in Figure 3 is the version of the code presented in our class for non-majors. We have found that these students are better served by keeping parameters to a minimum. However, we do not want to depend completely on global variables. To these students, the only potential confusion is that the parameter `residentList` is not used in the body of procedure `MakeMatch` (these students have access to `MakeMatch` and `OpenHospitalIndex` and to no other subprograms.)

The format of the input file used by this program is

```
# of residents
 hospital list for resident 1
...
hospital list for resident n
# of hospitals
# of slots in hospital 1 resident list for hospital 1
...
# of slots in hospital n resident list for hospital n
```

## References

[Bie91] Alan W. Biermann. An overview course in academic computer science: A new approach for teaching nonmajors. In *The Papers of the Twenty-first SIGCSE Technical Symposium on Computer Science Education*, pages 236–239. ACM Press, February 1991. SIGCSE Bulletin V. 23 N. 1.

[DH91] Richard W. Decker and Stuart H. Hirshfield. A survey course in computer science using hypercard. In *The Papers of the Twenty-first SIGCSE Technical Symposium on Computer Science Education*, pages 229–235. ACM Press, February 1991. SIGCSE Bulletin V. 23 N. 1.

[GI89] Dan Gusfield and Robert W. Irving. *The Stable Marriage Problem: Structures and Algorithms.* MIT Press, 1989.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1979.

[GS62] D. Gale and L.S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.

[Knu73] D.E. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, 1973. Sorting and Searching.

[Knu76] D.E. Knuth. *Mariages Stables.* Les Presses de l'Université Montréal, 1976.

[Kor86] James F. Korsh. *Data Structures, Algorithms, and Program Style.* PWS Publishers, 1986.

[Mar] David Margolick. Annual race for clerks becomes a mad dash, with judicial decorum left in the dust. New York Times, March 17, 1989.

[MW71] D.G. McVitie and L.B. Wilson. The stable marriage problem. *Communciations of the ACM*, 14(7):486–492, 1971.

[Nat91] National Resident Matching Program, Evanston, Illinois. *Handbook for students Participating Through U.S. Medical Schools*, April 1991.

[Pat88] Richard E. Pattis. Textbook errors in binary searching. In *The Papers of the Nineteenth Technical Symposium on Computer Science Eduction*, pages 190–194. ACM Press, February 1988. SIGCSE Bulletin V. 20 N. 1.

[Pat91] Richard E. Pattis. A Philosophy and Example of CS-1 Programming Projects. In *The Papers of the Twenty-first SIGCSE Technical Symposium on Computer Science Education*, pages 34–39. ACM Press, February 1991. SIGCSE Bulletin V. 23 N. 1.

[Qui85] M.J. Quinn. A note on two parallel algorithms to solve the stable marriage problem. *BIT*, 25:473–476, 1985.

[Rot84] Alvin E. Roth. The Evolution of the Labor Market for Medical Interns and Residents: A Case Study in Game Theory. *Journal of Political Economy*, 92(6):991–1016, 1984.

[Sed90] Robert Sedgewick. *Algorithms in C.* Addison-Wesley, 1990.

[Tuc91] Allen B. Tucker, editor. *Computing Curricula 1991 Report of the ACM/IEEE-CS Joint Curriculum Task Force.* ACM Press, 1991.

[Wir76] Niklaus Wirth. *Algorithms + Data Structures = Programs.* Prentice-Hall, 1976.