Information Processing Letters 16 (1983) 61-65
North-Holland Publishing Company

26 February 1983

STABLE MARRIAGES BY COROUTINES

Lloyd ALLISON

Department of Computer Science, University of Western Australia, Nedlands 6009, Western Australia

Communicated by H. Whitfield
Received 9 August 1982
Revised 12 November 1982

The stable marriage problem is an appealing version of many pairing problems. A solution by coroutines is given, based on the recursive algorithm of McVitie and Wilson (1971). There are few published algorithms where coroutines are really useful but they solve this problem very naturally.

Keywords: Coroutine, stable marriage problem, algorithm

1. Introduction

There are relatively few published algorithms using coroutines notwithstanding a great deal of literature about them. A coroutine solution to the stable marriage problem is given to help redress the imbalance and because it is an interesting problem which coroutines solve naturally. The algorithm is based on that of McVitie and Wilson [5] which was specified in ALGOL 60.

The stable marriage problem was posed by Gale and Shapley [1].

Stable Marriage Problem ([1]). Given a set of n men, a set of n women and a list of preferences of marriage partner for each man and each woman, find a stable marriage for the whole group.

The marriage is *unstable* if there are a man and a woman who are not married to each other but who *both* prefer each other to their actual partners. The problem was given as a light hearted variation of the college admissions problem. Gale and Shapley showed that at least one and possibly many solutions always exist and gave an algorithm to solve the problem.

McVitie and Wilson [5] gave three ALGOL 60 programs – the Gale and Shapley algorithm, a faster algorithm of their own and an algorithm to find all stable marriages. A coroutine solution based on the second of these is offered here as an interesting use of this mechanism.

McVitie and Wilson also extended their work to unequal sets [6] but that will not be covered here. Wirth [8] has used the stable marriage problem to illustrate recursion, giving a slower but simpler backtracking algorithm.

A number of languages offer coroutines including SL5 [2] and Modula-2 [7]. One example of the use of coroutines is for generators which produce a sequence of objects to control iteration or to provide a sequence of names. Another use is in the implementation of other control mechanisms, notably backtracking or nondeterminism. The book by Marlin [4], as well as a large bibliography, includes the telegram problem [3] solved by coroutines.

2. The algorithm

The McVitie and Wilson algorithm [5] to find *a* stable marriage is very appealing. Each man proposes to his preferences in order until he is accepted. If he is later jilted he continues with his next choice. Each woman accepts her more preferred suitor; she accepts the first proposal but later may jilt her financė if she gets a better offer. The sequence of events starts with the first man proposing to his first choice and being accepted. Each subsequent man proposes either to an unengaged woman who accepts him or to an already engaged woman. In the latter case she may reject him, or, if she prefers him to her current fiancé, change allegiances. In either case the rejected man proposes to his next choice.

It is not obvious that this leads to a stable marriage, but it does, and an informal argument is sketched here. If there is only one man and one woman the process certainly works. Now, suppose $i - 1$ of the men are married to some $i - 1$ of the women and consider the ith man. If he proposes to an unengaged woman she accepts. Since none of the first $i - 1$ men proposed to her the new marriage is stable. If the ith man proposes to an engaged woman she either accepts him because she is getting a better choice or rejects him. In the first case she is less likely than ever to render the marriage unstable. In either case the rejected man can only promote women to better choices and the same argument applies. Since there is at least one unengaged woman, the chain of refusals and jiltings must terminate. In fact this leads to the male optimal solution in which each man is at least as well off as in any other solution. Reversing the roles leads to the female optimal solution.

As an example suppose there are three men 1, 2 and 3 and three women A, B and C with preferences

1	A	B	C
2	A	C	B
3	B	C	A

A	2	1	3
B	2	3	1
C	1	2	3

```
procedure man (me: manindex);
  var     j, choice: womanindex;
begin     for j := 2 to n do {try each choice}
            choice := mchoice[me, j];
            {propose!} resume w[choice] with choice, me
          od
end man;

procedure woman (me: womanindex; suitor: manindex);
  var     jilt: manindex;
begin     wengagement[me] := suitor {accept first};
          return;
          do {ask questions later}
            if wrank[me, suitor] < wrank[me, wengagement[me]] then
              jilt := wengagement[me];
              wengagement[me] := suitor;
              resume m[jilt]
            else
              resume m[suitor]
            fi
          od
end woman;

begin     for i := 1 to n do
            w[i] := create woman with i;
            m[i] := create man with i
          od;
          set up choices and rankings;
          for i := 1 to n do
            resume m[i]
          od;
          process the stable marriage in wengagement [ ]
end.
```

Fig. 1. Stable marriage by coroutines.

```
MODULE stable marriage;
(* L.A. Computer Science, University of Western Australia *)
(* here IMPORT standard routines FROM SYSTEM, TTIO & WriteStrings *)
CONST maxn = 10;
VAR workspace: ARRAY [0..200* maxn] OF WORD;
   w, m:ARRAY[1..maxn] OF PROCESS; main:PROCESS;
   wengagement: ARRAY [1..maxn] OF INTEGER;
   wrank, mrank, wchoice, mchoice:ARRAY [1..maxn] OF
                                              ARRAY [1..maxn] OF INTEGER;
   n, i, theman, thewoman, pref, suitor:INTEGER;
   nextch:CHAR;
   isnum:BOOLEAN;

PROCEDURE man;
   VAR me, wo, j:INTEGER;
BEGIN
   me:=i (* when created *);
   TRANSFER(m[me], main);
   FOR j:=1 TO n DO (* propose to preferences in order *)
     wo:=mchoice[me, j]; suitor:=me;
     TRANSFER(m[me], w[wo])
   END;
   WriteString('error: should never get here'): WriteLn
END man;

PROCEDURE woman;
   VAR me, jilt:INTEGER;
BEGIN
   me: =i (* when created *);
   TRANSFER(w[me], main);
   (* accept first offer *)
   wengagement[me]:=suitor;
   TRANSFER(w[me], main);
   LOOP (* all the while accepting better offers *)
     IF wrank[me, suitor] < wrank[me, wengagement[me]] THEN (* fickle *)
       jilt:=wengagement[me];
       wengagement[me]:=suitor;
       TRANSFER(w[me], m[jilt])
     ELSE
       TRANSFER(w[me], m[suitor])
     END (* if *)
   END (* loop *)
END woman;

BEGIN (* main *)
   ReadInt(n, nextch, isnum);
   FOR i:=1 TO n DO
     NEWPROCESS(man, ADDRESS(ADR(workspace[200*i])), 200, m[i]);
     TRANSFER(main, m[i]);
     NEWPROCESS(woman, ADDRESS(ADR(workspace[200*i+100])), 200, w[i]);
     TRANSFER(main, w[i]);
   END;
   (* here read preferences and invert into wrank and mrank *)
   FOR the man:=1 TO n DO (* marry the man *)
     TRANSFER(main, m[theman]
   END;
   (* here output the solution held in wengagement [ ] *)
END stablemarriage.
```

Fig. 2. Stable marriage in Modula-2.

Then

1 proposes to A who accepts,
2 proposes to A, who accepts and jilts 1 and 1 is then accepted by B,
3 proposes to B who accepts him jilting 1 and 1 is then accepted by C.

The stable marriage is then

$$(1, C) \qquad (2, A) \qquad (3, B).$$

The algorithm is given in SL5-like notation in Fig. 1. The three components available in SL5 for coroutine use are

create –	creation of an instance of a coroutine,
with –	binding actual parameters to the coroutine instance,
resume –	transferring control to the coroutine instance.

There is an instance of a coroutine for each man and each woman; these are created at the beginning of the main program. A 'man' coroutine is a loop over his preferences and in order to propose he uses *resume.* This passes control to the appropriate instance of a 'woman' coroutine. Initially a woman accepts the first suitor's proposal, but subsequently she compares each suitor with her current fiancé and rejects the less favoured, using *resume* to pass control to the rejected man coroutine. The main program loops, executing *resume* on each man coroutine in turn until all are married.

The McVitie and Wilson ALGOL version [5] differs as follows. Procedure instances can only be created by (recursive) calls but information local to the instance is lost with return or loss of control and must be kept globally – in particular the choice counter for each man. The first call or proposal to a woman cannot be distinguished easily (as own variables cannot be initialised) so a dummy man is introduced.

In the ALGOL solution the controlling information resides in the main program and the procedure calls are a sophisticated way of altering it. In the coroutine solution each instance of man or woman controls its own information independently of the rest. In the description of their algorithm McVitie and Wilson actually refer to a woman keeping a man 'in suspense' which corresponds to a coroutine being inactive.

3. An implementation

Modula-2 [7] offers coroutines and is quite widespread. The algorithm previously decribed was coded in Modula-2, executed, and is presented in Fig. 2 for completeness.

The significant features of Modula-2 for these purposes are

PROCESS –	coroutine,
NEWPROCESS –	given a procedure, the base of an area, its length and a process variable, produces a process or coroutine instance,
TRANSFER –	is equivalent to *resume* except that a variable to hold the 'return' address as well as a destination must be given.

Further, a process or coroutine cannot have parameters so that the easiest way for one of an array of identical coroutines to identify itself is for it to access some global counter immediately after being created. The suitor must be global too. (These features of Modula-2 are low level but are ideally suited to its intended use of the systems programming.)

4. Conclusion

A stable marriage algorithm has been programmed with coroutines. It is argued that these naturally model the algorithm and its informal description. Compared to the recursive algorithm [5] the final sequence of proposals, acceptances and rejections is the same except that coroutine *resume* replaces recursive procedure call, and local variables replace array elements with a slight gain in efficiency. The solution is put forward as a nontrivial example of the use of coroutines.

References

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

[2] D.R. Hanson and R.E. Griswold, The SL5 procedure mechanism, Comm. ACM 21 (5) (1978) 392–400.

[3] P. Henderson and R. Snowdon, An experiment in structured programming, BIT 12 (1972) 38–53.

[4] C.D. Marlin, Coroutines, in: Lecture Notes in Comput. Sci. 95 (Springer, Berlin, 1980).

[5] D.G. McVitie and L.B. Wilson, The stable marriage problem, Comm. ACM 14 (7) (1971) 486–490; Algorithm 411 (1971) 491–492.

[6] D.G. McVitie and L.B. Wilson, Stable marriage assignment for unequal sets, BIT 10 (1970) 295–309.

[7] N. Wirth, Modula-2, Rept., E.T.H. Zurich, 1980.

[8] N. Wirth, Algorithms + Data Structures = Programs (Prentice-Hall, Englewood Cliffs, NJ, 1976).

Added by Editor

[9] N. Wirth, Programming in Modula-2 (Springer, Berlin, 1982).