```
                begin
                    if h<min then begin up:=low; min:=h end
                    col[k]:=col[up]; col[up]:=j; up:=up+1
                end
            end;
            for h:=low to up-1 do
            begin j:=col[h]; if y[j]=0 then goto augment end
        end; {up=low}
        j1:=col[low]; low:=low+1; i:=y[j1];                    {# # scan a row}
        u1:=c[i,j1]-v[j1]-min;
        for k:=up to n do
        begin
            j:=col[k]; h:=c[i,j]-v[j]-u1;
            if h<d[j] then
            begin
                d[j]:=h; pred[j]:=i;
                if h=min then
                    if y[j]=0 then goto augment
                        else begin col[k]:=col[up]; col[up]:=j; up:=up+1 end
            end
        end
    until false; {repeat ends with goto augment}
augment:
    for k:=1 to last do                                {# # updating of column prices}
        begin j1:=col[k]; v[j1]:=v[j1]+d[j1]-min end;
    repeat                                                 {# # augmentation}
        i:=pred[j]; y[j]:=i; k:=j; j:=x[i]; x[i]:=k
    until i=i1
end; {of augmentation}
h:=0;                          {# # # # DETERMINE ROW PRICES AND OPTIMAL VALUE}
for i:=1 to n do begin j:=x[i]; u[i]:=c[i,j]-v[j]; h:=h+u[i]+v[j] end;
lapjv:=h
end
```

Fig. 6. Pascal function for the linear assignment algorithm LAPJV

8. Computational Results

We compare our algorithm LAPJV with several other methods, both on dense and on sparse problems. The computational results are for Fortran codes run on a CDC Cyber 750 (with OPT = 2), but the algorithms were developed in Pascal on personal computers (Apricot PC and F1, Olivetti M24). Running times on these computers are typically between 2 and 4 seconds for full dense problems of size 100, using Borland's Turbo Pascal compiler (version 3.0).

The classical Hungarian code of Silver [27] is an intermediary for a comparison with the shortest path based algorithm of Hung and Rom [18]. They give the ratio of their computing times to those of the Silver code translated into Fortran. Dividing the same ratios for LAPJV by those published by Hung and Rom shows that on problems of full density our algorithm is about twice as fast (Table 2).

Table 2. *Computation times of LAPJV divided by those of Hung and Rom*

problem size	range of cost coefficients 1–100	1–1000
50	.34	.72
100	.31	.52

We left primal simplex methods out of consideration, as the literature shows that these are outperformed by several other methods. The algorithm of Hung and Rom is "about twice as fast" as that of Barr, Glover and Klingman [2], which is one of the best primal simplex methods. Glover confirmed this in a private communication. Carpaneto and Toth [6] compare the same primal simplex method with their Hungarian code for sparse problems SPASS and with a Tomizawa based method, adapted from a code in Burkard and Derigs [4]. Both algorithms are faster by some margin. Finally, the Hungarian method as implemented by McGinnis [25] is "roughly comparable in solution speed", but in an addendum he improves his method to a much faster one.

The computation times of LAPJV in Table 3 are averages for ten full density problems, compared to those of the algorithms:

- ASSCT: $O(n^4)$ Hungarian method coded by Carpaneto and Toth [5], making extensive use of pointer techniques to locate zero valued reduced costs,
- LSAP: Dorhout's improved version of Tomizawa's algorithm [10, 11] translated into Fortran and published by Burkard and Derigs [4],
- ASSIGN: the algorithm of Bertsekas [3], as made available by the author and adapted for full density costs matrices.

Table 3. *Computation times for assignment problems of full density (in ms)*

cost range		LAP algorithm ASSCT	LSAP	ASSIGN	LAPJV
1–100	50	51	32	22	15
	100	149	168	114	64
	150	283	535	256	179
	200	420	1363	520	323
1–1000	50	121	30	24	17
	100	637	165	123	77
	150	1447	456	364	225
	200	2217	850	665	406
1–10000	50	145	31	28	25
	100	1085	168	134	103
	150	3562	453	410	259
	200	6989	919	779	456