variable μ). The set may be updated while scanning, and (relatively expensive) searching for a new value of the minimum does not take place until SCAN is empty. Especially for sparse networks this leads to substantially lower computation times.

```
procedure SHPATH2(kk);
begin
   TOSCAN:={1 ... n}-{kk}; for j:=1 ... n do d[j]:=∞;
   d[kk]:=0; SCAN:={kk}; μ:=0;
   repeat
      select any i∈SCAN; SCAN:=SCAN-{i};
      for all j∈A[i]∩TOSCAN do if μ+c[i,j]<d[j] then
      begin
         d[j]:=μ+c[i,j]; pred[j]:=i;
         if d[j]=μ then begin SCAN:=SCAN+{j}; TOSCAN:=TOSCAN-{j} end
      end;
      if SCAN={} then
      begin
         μ:=min {d[j] | j∈TOSCAN; d[j]>μ};
         for j∈TOSCAN do if d[j]=μ then SCAN:=SCAN+{j};
         TOSCAN:=TOSCAN-SCAN
      end
   until SCAN=
end.
```

Fig. 4. Modified version of Dijkstra's shortest path method, basis for improved augmentation

Augmentation in our LAP algorithm is given by the procedure AUGMENT in Fig. 5, which is based on SHPATH2. Some minor improvements have been made. Instead of a list SCAN containing rows, a list of columns facilitates updating of column prices. Furthermore, we do not keep and update row prices. These can be determined easily when needed due to complementary slackness. Updating of column prices takes place as will be discussed in Section 6.

The column sets READY, SCAN and TODO are mutually disjoint, and READY $\cup$ SCAN $\cup$ TODO $=\{1 \dots n\}$. So in the code one array COL of length n is sufficient, with the elements of READY kept in front, just before the elements of SCAN. This set is scanned first-in-first-out. So, its elements transfer automatically to READY. The remaining places in the array are used for TODO.

Just like the initialization, the augmentation phase has computational complexity $O(n^3)$. So this also holds for the entire algorithm. As for the memory requirements: the full density version uses a costs matrix and eight arrays of n elements.

Derigs and Metz [8] investigated implementations of Dijkstra's shortest path algorithm in assignment methods. The fastest implementation turned out to be very similar to ours. They discovered that in their algorithms it is advantageous to determine complete shortest path trees, so that more than one augmenting path per iteration may be found. Our algorithm is faster when in each iteration only one augmenting path is determined, which is probably due to the extensive initialization procedures.

```
procedure AUGMENT;
begin
   for all unassigned i* do
   begin
      for j:=1 ... n do begin d[j]:=c[i*,j]-v[j]; pred[j]:=i* end;
      READY:={}; SCAN:={}; TODO:={1 ... n};
      repeat
         if SCAN={} then
         begin
            μ:=min {d[j] | j∈TODO}; SCAN:={j | d[j]=μ}; TODO:=TODO-SCAN
            for all j∈SCAN do if y[j]=0 then go to augment
         end;
         select any j*∈SCAN; i:=y[j*]; SCAN:=SCAN-{j*}; READY:=READY+{j*}
         for all j∈TODO do if μ+cred[i,j]<d[j] then
         begin
            d[j]:=μ+cred[i,j]; pred[j]:=i;
            if d[j]=μ then
               if y[j]=0 then go to augment else
               begin SCAN:=SCAN+{j}; TODO:=TODO-{j} end
         end
      until false; (* repeat always ends with go to augment *)
augment:
      (* price updating *)
      for all k∈READY do v[k]:=v[k]+d[k]-μ;
      (* augmentation *)
      repeat
         i:=pred[j]; y[j]:=i; k:=j; j:=x[i]; x[i]:=k
      until i=i*
   end
end.
```

Fig. 5. The procedure AUGMENT for augmentation in the algorithm LAPJV

We experimented with augmentation based on the new shortest path methods from Glover et al. [15, 16]. However, we did not find a faster procedure than one based on SHPATH2. Carraresi and Sodini [7] report very good results with an algorithm based on these shortest path methods. It must be noted that this LAP algorithm performs well only on (very) sparse problems. Karp [21] also improved shortest path based algorithms, but his modifications are more theoretically interesting than practically. By using priority queues, he reduced expected running time for the LAP to $O(n^2 \cdot \log n)$.

6. Adjustment of the Dual Solution

After augmentation of a partial assignment the values of the dual variables must be updated to restore complementary slackness, that is,

$$c[i,k]-u[i]-v[k]=0, \quad \text{if } x[i]=k \quad (i=1 \dots n), \tag{1}$$

$$c[i,j]-u[i]-v[j]\geq 0 \qquad (i,j=1 \dots n). \tag{2}$$

Substituting the values $u[i]$ from (1) in (2) leads to:

$$c[i,k]-v[k]\leq c[i,j]-v[j] \qquad (j=1 \dots n).$$