row i, thus unassigning the row k previously assigned to column j. Clearly, row i can now be reduced by its minimum reduced costs, but if the second minimum is higher, reduction transfer is possible by increasing the elements in column j. If so, we next consider row k, as for this row the minimum reduced costs may now occur in a column different from j. If this is the case, the alternating path is extended as before. If not, and furthermore the minimum is still unique, the path may even reverse direction, and be extended by rows and columns visited before. The process is continued until either an additional assignment is found, or no reduction transfer takes place.

The two previous reduction procedures both have computational complexity $O(n^2)$. It can be shown that for augmenting reduction at most $O(n^2 \cdot R)$ steps are taken, with R the range of the cost coefficients. With each step involving $O(n)$ operations, the complexity is at most $O(n^3 \cdot R)$. A different argument is as follows. We determine $O(n)$ alternating paths. Each extension of a path takes $O(n)$ operations. So, an $O(n^3)$ computational complexity is obtained by simply allowing no more than n path extensions. In practice the procedure never even approaches this number of extensions.

Computational experiments show best results when the procedure of augmenting row reduction is performed twice.

The advantages of this initialization strategy over standard column and row reduction amply compensate the additional computation time. On full density problems with $n=100$ average total time has been decreased by 10% (on cost range $1-100$) to 18% (on $1-1000$). This initialization phase takes 60% to 70% of total run time, whereas simple column and row reduction would take about 20%. We expect the effect of these initialization routines to be larger on augmentation approaches less efficient than ours. Table 1 illustrates the advantages. It gives the average reduction sum (the value of the current dual solution) and the average number of assignments in the partial primal solution. These figures indicate how much effort must still be put in the augmentation phase of the algorithm. The increased average numbers of zero reduced costs coefficients suggest that this method is also useful for assignment algorithms based on maximal flow.

Table 1. *Column and row reduction compared to the initialization in LAPJV (averages for 25 problems of each type with $n=100$)*

	cost range	column and row reduction	initialization in LAPJV
reduction sum	1–100	87.2	96.6
in % of optimum	1–1000	87.2	98.0
number of	1–100	75	90
assignments	1–1000	75	95
number of zero reduced	1–100	188	205
cost coefficients	1–1000	142	162

5. The Augmentation Phase

Augmentation starts by finding an alternating path. This is a sequence of, alternately, row and column indices, with the first an unassigned row, the last a column, and the intermediate columns and rows assigned in successive pairs. If the final column is unassigned, augmentation of a partial solution can take place along such a path by assigning all rows in the path to their succeeding column, which results in one more assignment.

Augmentation in shortest path based algorithms (step 3, in Section 3) can best be described without direct reference to the underlying minimum cost flow problem. It requires only a simple modification of the shortest path method of Dijkstra [12]. In Fig. 3 we give two procedures. Dijkstra's algorithm SHPATH1 determines a shortest path tree rooted in node kk and traceable in the pred-array. The set $A[i]$ contains all nodes j for which arc $\langle i,j\rangle$ exists. Procedure SHPATH-AUGMENT is its modification for the assignment problem, which determines the shortest augmenting path for one additional assignment in row kk.

```
procedure SHPATH1(kk);
begin
    TOSCAN:={1 ... n}-{kk}; for j:=1 ... n do d[j]:=∞;
    i:=kk; d[kk]:=0; μ:=0;
    repeat
        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 end;
        μ:=∞;
        for all j∈TOSCAN do if d[j]<μ then begin μ:=d[j]; i:=j end;
        TOSCAN:=TOSCAN-{i}
    until TOSCAN={}
end

procedure SHPATH-AUGMENT(kk);
begin
    TOSCAN:={1 ... n}; for j:=1 ... n do d[j]:=∞;
    i:=kk; d[kk]:=0; μ:=0;
    repeat
        for all j∈A[i]∩TOSCAN do
            if μ+cred[i,j]<d[j] then begin d[j]:=μ+cred[i,j]; pred[j]:=i end;
        μ:=∞;
        for all j∈TOSCAN do if d[j]<μ then begin μ:=d[j]; μj:=j end;
        i:=y[μj]; TOSCAN:=TOSCAN-{μj}
    until y[μj]=0;
    <augment along the path from column μj to row kk>
end
```

Fig. 3. A shortest path algorithm according to Dijkstra and a modified version for augmentation in assignment algorithms

This procedure only describes the method. For the actual implementation an adapted version of Dijkstra's shortest path algorithm as in Fig. 4 is to be preferred. The shortest path algorithms in Figs. 3 and 4 differ in the use of a set SCAN, containing all rows that can be scanned for the current minimum d-value (the