4.2. Algorithm

Split works on an auxiliary graph $H=(X,A,Z)$. X contains $n+1$ nodes indexed from 0 to n. A contains one arc (i,j), $i<j$, if a trip visiting customers S_{i+1} to S_j is feasible in terms of load (condition (4)) and cost (condition 5)). The weight z_{ij} of (i,j) is equal to the trip cost.

$$\forall (i,j) \in A\colon \sum_{k=i+1}^{j} q_{S_k} \leqslant W, \tag{4}$$

$$\forall (i,j) \in A\colon z_{ij} = c_{0,S_{i+1}} + \sum_{k=i+1}^{j} (d_{S_k} + c_{S_k,S_{k+1}}) + d_{S_j} + c_{S_j,0} \leqslant L. \tag{5}$$

An optimal DVRP solution for S corresponds to a min-cost path μ from 0 to n in H. This evaluation is reasonably fast because H is circuitless, $|A|=O(n^2)$, and the node numbering provides a natural topological ordering: in that case, μ can be computed in $O(n^2)$ using Bellman's algorithm [1]. The algorithm is faster when the minimal demand $q_{\min}$ is large enough: since a trip cannot visit more than $b=\lfloor W/q_{\min}\rfloor$ customers, the complexity becomes $O(nb)$.

The top of Fig. 1 shows a sequence $S=(a,b,c,d,e)$ with $W=10$, $L=\infty$ and demands in brackets. H in the middle contains for instance one arc ab with weight 55 for the trip $(0,a,b,0)$. μ has three arcs and its cost is 255 (bold lines). The lower part gives the resulting VRP solution with three trips.

The algorithm of Fig. 2 is a version in $O(n)$ space that does not generate H explicitly. It computes two labels for each node $j=1,2,\ldots,n$ of X: V_j, the cost of the shortest path from node 0 to node j in H, and P_j, the predecessor of j on this path. The *repeat* loop enumerates all feasible sub-sequences $S_i \ldots S_j$ and directly updates V_j and P_j. The required fitness $F(S)$ is given at the end by V_n. For a given i, note that the incrementation of j stops when L is exceeded: no feasible trip is discarded since the triangle inequality holds.

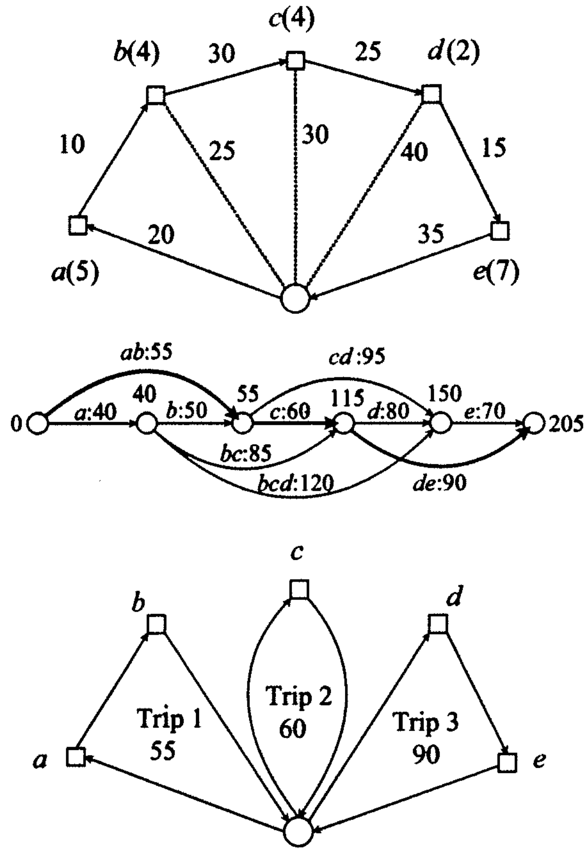

Fig. 1. Example of chromosome evaluation.

The vector of labels P is kept with the chromosome to extract the DVRP solution at the end of the GA, using the algorithm of Fig. 3. It builds up to n trips (worst case with one vehicle per demand). Each trip is a list of clients, possibly empty. The procedure *enqueue* adds a node at the end of a trip. The number of non-empty trips (or vehicles) actually used is given by t.

```
V_0 := 0
for i := 1 to n do V_i := +∞ endfor
for i := 1 to n do
   load := 0; cost := 0; j := i
   repeat
      load := load + q_{S_j}
      if i = j then
         cost := c_{0,S_j} + d_{S_j} + c_{S_j,0}
      else
         cost := cost − c_{S_{j−1},0} + c_{S_{j−1},S_j} + d_{S_j} + c_{S_j,0}
      endif
      if (load ≤ W) and (cost ≤ L) then
         //here substring S_i ... S_j corresponds to arc (i − 1, j) in H
         if V_{i−1} + cost < V_j then
            V_j := V_{i−1} + cost
            P_j := i − 1
         endif
         j := j + 1
      endif
   until (j > n) or (load > W) or (cost > L)
enfor.
```

Fig. 2. Algorithm for the splitting procedure *split*.

```
for i := 1 to n do trip(i) := ∅ endfor
t := 0
j := n
repeat
   t := t + 1
   i := P_j
   for k := i + 1 to j do enqueue(trip(t), S_k) endfor
   j := i
until i = 0.
```

Fig. 3. Algorithm to extract the VRP solution from vector P.