**RAIRO Operations Research**
RAIRO Oper. Res. **36** (2002) 1-19
DOI: 10.1051/ro:2002002

# THE MAXIMUM CAPACITY SHORTEST PATH PROBLEM: GENERATION OF EFFICIENT SOLUTION SETS

T. Brian Boffey[1], R.C. Williams[2], B. Pelegrín[3] and P. Fernandez[3]

**Abstract**. Individual items of flow in a telecommunications or a transportation network may need to be separated by a minimum distance or time, called a "headway". If link dependent, such restrictions in general have the effect that the minimum time path for a "convoy" of items to travel from a given origin to a given destination will depend on the size of the convoy. The Quickest Path problem seeks a path to minimise this convoy travel time. A closely related bicriterion problem is the Maximum Capacity Shortest Path problem. For this latter problem, an effective implementation is devised for an algorithm to determine desired sets of efficient solutions which in turn facilitates the search for a "best" compromise solution. Numerical experience with the algorithm is reported.

**Keywords:** Quickest path, shortest path, path capacity, efficient solution.

**Mathematics Subject Classification.** 90B10, 90B18.

## Introduction

Shortest path methods play a central role in many problems associated with telecommunications and transportation networks. Sometimes, however, throughput is restricted by "headway" requirements on the minimum separation in space

Received August, 1997.

[1] University of Liverpool, Liverpool L69 7ZL, U.K.
[2] University of Abertay, Dundee, U.K.
[3] University of Murcia, Murcia, Spain.

© EDP Sciences 2002

or time between successive items of flow. Thus, the (minimum) time for a convoy of items to travel from a given origin $O$ to a given destination $D$ has a contribution corresponding to inter-item headways. Consequently, if headway restrictions are link dependent, the path(s) leading to minimal convoy travel time will, in general, depend on convoy size. The determination of minimal convoy travel time paths gives rise to the so-called Quickest Path problem [6, 22]. The Maximum Capacity Shortest Path problem [1], a closely related bicriterion problem, is defined in the next section.

The Quickest Path problem is relevant for situations in which convoy travel time is critical, or at least very important. For transportation networks it is usually the case that vehicles are travelling as "individuals", though the armed forces do make use of convoys for moving equipment and personnel quickly. For telecommunications networks the situation is different. There, individual items are packets, with "messages" (the convoys) often comprising very many packets. Frequently, these messages are constrained to follow a single route determined by the system and a time headway between packets is determined by the "speed" of each relevant link. For time critical applications the identification of a minimal convoy travel time path assumes importance. Perhaps more interesting is the potential application to combined routing /congestion control in computer networks when datagram routing is employed. A shortest path corresponds to one with shortest delay (under "normal conditions"). A maximum capacity path would correspond to one that could transport relatively many packets per unit time (possibly also taking account of buffer occupancy en route). Since network conditions will be varying, calculations should be short and it may be appropriate to calculate few paths, perhaps merely a maximum capacity path and a minimum length path. This could help to control congestion when a long "burst" of traffic is encountered since routing could then be switched to a higher capacity path. The relevance of this is that it has been found to be relatively common for network traffic to exhibit self-similarity (*e.g.* [24, 27]) which implies that there will be "bursts of all lengths". This scheme is tentative as there are other complications, but it is worthy of investigation.

Algorithms for the Quickest Path problem with good asymptotic behaviour have been devised (*e.g.* [29]). Here we present an algorithm with the same complexity but describe an improved implementation. The strategy, which is based on making effective use of information gained early in the solution process to reduce the amount of effort required subsequently, has been found to lead to substantial computational savings for the Bicriterion Shortest Path (BSP) problem and it was of interest to investigate whether similar savings could be achieved for the present problem. The results in Section 3 demonstrate that considerable computational savings can indeed be made, though they are less impressive than for BSP [3].

To be precise, the (time) *headway*, $h_\alpha$, of a link $\alpha$ is the minimum permissible time difference between $t_i$ the (start of) arrival of the $i$-th item and $t_{i+1}$ the (start of) arrival of the immediately succeeding item. This corresponds to a variety of different real situations as illustrated in Figure 1. In Figure 1a, items are of negligible dimension and time of arrival is well defined – the headway is imposed to

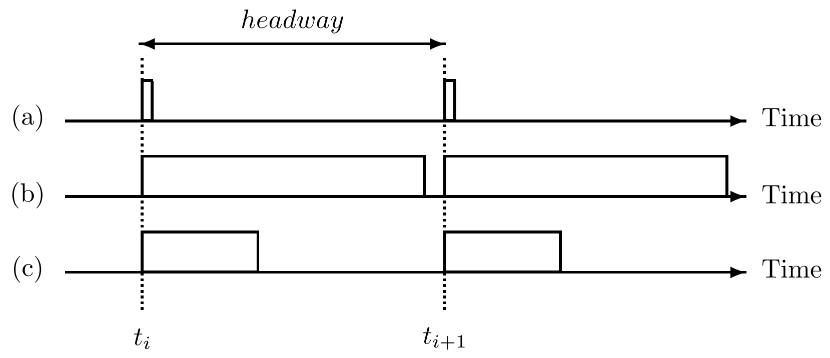

FIGURE 1. The time $t_{i+1} - t_i$ is the headway (minimal time between successive item arrivals). The rectangles on each line indicate the "extent" of an item (that is, the time it takes an item to arrive) and the remaining headway time represents the minimal enforced gap between items.

keep items well separated. On the other hand, the time difference may be entirely due to items being extensive (Fig. 1b) – this is a sustainable view of the situation pertaining to digital signals over a telecommunications link. Finally, the situation may be somewhere between these two extremes (Fig. 1c) as would be the case for trains on a metro network.

Now consider a *convoy* of $s$ identical items. For the whole convoy to pass along a link $\alpha$, the time $t_\alpha$ taken will be at least

$$t_\alpha = l_\alpha + \sigma h_\alpha \tag{1}$$

where $l_\alpha$ is the time taken for one unit to traverse $\alpha$. The value of $\sigma$ will vary from $s-1$ when arrival is instantaneous (Fig. 1a) up to $s$ when the headway is due solely to items being extensive. In any case, the optimal path will depend on $\sigma$ and we shall be interested in determining optimal paths as a function of $\sigma$ for a range of values (which might even be all $\sigma$ in $0 < \sigma < \infty$). Consequently, the precise way in which $\sigma$ relates to $s$ will not be discussed further.

To the authors' knowledge, Moore [25] was first to consider routing of convoys through networks as quickly as possible, and two algorithms were proposed but these, however, were not tested other than on a very small illustrative example. Recently there has been much interest in the problem in relation to telecommunications networks. Chen and Chin [6], Hung and Chen [18], Rosen *et al.* [29] and Kagaris [22] have considered the problem under the name "Quickest Path" (QP) Problem. Related problems have been studied in [4, 5, 7, 8, 11, 17–21, 23]. Boffey [1] posed the Maximal Capacity Shortest Path (McSP) problem which is very similar. (We use the abbreviation McSP with lowercase "c" to distinguish the problem from the Maximal Covering Shortest Path problem [10].)

Various of the above cited papers contain Quickest Path algorithms and quoted worst case complexity results. These results predict asymptotic behaviour but it is also relevant to consider algorithm implementation for practical networks that may contain no more than a few hundred nodes. This is particularly so if calculations are required frequently to take account of changes in network conditions (*e.g.* link failure or congestion in telecommunications networks). This paper considers algorithm implementation in some detail and presents numerical experience.

The plan of the paper is as follows. The next section looks briefly at the relationship between QP and McSP. Then in Section 2 an algorithm to calculate all efficient solutions with path length not exceeding some specified maximum is considered in some detail together with its efficient implementation. Numerical experience with the implementation is reported in Section 3 and, finally, we present our conclusions.

## 1. Efficient sets for problems McSP and BQP

Concepts relating to multicriteria problems will briefly be described, then some theoretical results which underpin the subsequent algorithmic development are established.

### 1.1. Multicriteria concepts

Consider the bicriterion minimisation problem

P: minimise $Z_1(x)$
minimise $Z_2(x)$
subject to $x \in \Omega$.

In general, there will be no single solution which simultaneously minimises both objectives and the concept of "optimality" is replaced by that of "efficiency". More precisely, let $x, y \in \Omega$ be two feasible solutions and suppose

$$Z_1(x) < Z_1(y) \text{ and } Z_2(x) \leq Z_2(y) \quad \textbf{or} \quad Z_1(x) \leq Z_1(y) \text{ and } Z_2(x) < Z_2(y).$$

When (at least) one of these two conditions holds then $(Z_1(x), Z_2(x))$ is said to *dominate* $(Z_1(y), Z_2(y))$. A feasible solution $x$ is *efficient* if $(Z_1(x), Z_2(x))$ is dominated by no other $(Z_1(u), Z_2(u))$, where $u$ is a feasible solution, and the set of all efficient solutions of a problem P will be denoted by $E(\mathrm{P})$. Each efficient solution $x$ defines an equivalence class $C(x) = \{u \mid (Z_1(u), Z_2(u)) = (Z_1(x), Z_2(x))\}$. We denote by $RE(\mathrm{P})$ a subset of $E(\mathrm{P})$ which contains precisely one *representative* element from each equivalence class. An efficient solution $y$ is *extreme* if $(Z_1(y), Z_2(y))$ is an extreme point of the set $\{(Z_1(u), Z_2(u)) \mid u \in \Omega\}$ in objective function space and there exists no convex combination $(1-\theta)(Z_1(a), Z_2(a)) + \theta(Z_1(b), Z_2(b))$ which dominates $(Z_1(y), Z_2(y))$ where $0 < \theta < 1$ and $a, b \in \Omega$. $EE(\mathrm{P})$ will denote the set of all extreme efficient solutions of problem P and $REE(\mathrm{P})$ will denote $EE(\mathrm{P}) \cap RE(\mathrm{P})$. It should be noted that $RE(\mathrm{P})$ and $REE(\mathrm{P})$ need not be unique.

A technique sometimes used to approximate $E(\mathrm{P})$ (or an $RE(\mathrm{P})$) is the Constraint Method. This involves solving the problem with one objective, say $Z_2$, the other being restricted. Thus

$$\begin{aligned}\mathrm{P}(k)\colon\ & \text{minimise } Z_2(x)\\ & \text{subject to } \ Z_1(x) \le k,\ \ x \in \Omega\end{aligned}$$

is solved for an “appropriate” set $\mathcal{K}$ of values of $k$ (see Sect. 2.5). It is shown in the next subsection that the method is well suited to the McSP and Bicriterion Quickest Path (BQP defined below) problems.

For more information on multicriteria methods and vector optimization the reader is referred to [9, 30].

## 1.2. McSP and the BQP problems

If the minimum time headway between items on link $\alpha$ is $h_\alpha$ then $c_\alpha = 1/h_\alpha$ is just the *capacity* of $\alpha$. Denoting by $\Lambda$ the set of simple (loopless) paths from *origin* $O$ to *destination* $D$, then for any $\pi \in \Lambda$, the capacity $c(\pi)$ of $\pi$ is defined by $c(\pi) = \min_{\alpha\in\pi} c_\alpha$. Also if $l_\alpha$ is the length of $\alpha$ then the length $l(\pi)$ of $\pi \in \Lambda$ is $l(\pi) = \Sigma_{\alpha\in\pi} l_\alpha$. The Maximal Capacity Shortest Path problem may now be stated formally as

$$\begin{aligned}\mathrm{McSP}\colon\ & \text{minimise } -c(\pi)\\ & \text{minimise } l(\pi)\\ & \text{subject to } \ \pi \in \Lambda.\end{aligned}$$

Referring back to the convoy problem mentioned in the introduction, the minimum convoy travel time will be controlled by the least capacity link encountered. Thus travel time $T(\pi)$ on path $\pi \in \Lambda$ will be $T(\pi) = l(\pi) + \sigma/c(\pi)$ where $\sigma$ is as described earlier. The conventional Quickest Path problem is then [6]

$$\begin{aligned}\mathrm{QP}(\sigma)\colon\ & \text{minimise } \{l(\pi) + \sigma/c(\pi)\}\\ & \text{subject to } \ \sigma \in \Lambda.\end{aligned}$$

Solving $\mathrm{QP}(\sigma)$ for $0 < \sigma < \infty$, amounts to an application of the Weighting Method (see Cohon [9]) to the *Bicriterion Quickest Path* problem:

$$\begin{aligned}\mathrm{BQP}\colon\ & \text{minimise } 1/c(\pi)\\ & \text{minimise } l(\pi)\\ & \text{subject to } \ \pi \in \Lambda.\end{aligned}$$

**Proposition 1.1.** *Provided all $c_\alpha > 0$,*
*(i) E(BQP) = E(McSP);*
*(ii) EE(McSP) $\subseteq$ EE(BQP).*

*Proof.* (i) follows immediately by observing that the transformation $x \to -1/x$ is order preserving for $-\infty < x < 0$ and substituting $x = -c(\pi)$.

(ii) Suppose $G = E(\mathrm{BQP}) - EE(\mathrm{BQP})$ is non-empty and let $\pi \in G$ be a non-extreme efficient solution of BQP. It is easily seen that there must exist $\rho$,

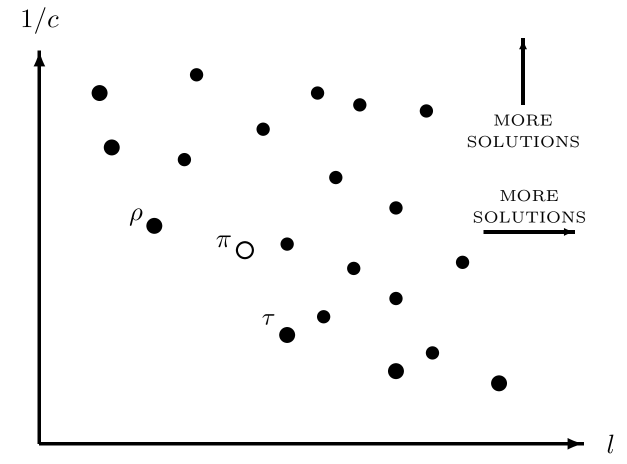

FIGURE 2. If $\pi$ is an efficient but not extreme solution of BQP, there exist $\rho, \tau \in E(\text{BQP})$ such that $\pi$ "is between $\rho$ and $\tau$" (see proof of proposition).

$\tau \in EE(\text{BQP})$ such that $l(\rho) < l(\pi) < l(\tau)$ and $1/c(\rho) > 1/c(\pi) > 1/c(\tau)$ (see Fig. 2). If $\theta,\ 0 < \theta < 1$, is the unique number satisfying

$$l(\pi) = (1-\theta)l(\rho) + \theta l(\tau), \tag{2}$$

then $\rho$ and $\tau$ can be chosen so that $1/c(\pi) \geq (1-\theta)(1/c(\rho)) + \theta(1/c(\tau))$ since otherwise $\pi$ would be extreme for BQP. Also, $c(\rho), c(\tau) > 0$ ($c_\alpha > 0$) and $x \to 1/x$ is a convex function for $0 < x < \infty$, hence it follows that $(1-\theta)(1/c(\rho)) + \theta(1/c(\tau)) > 1/[(1-\theta)c(\rho) + \theta c(\tau)]$. These last two relations combine to give $c(\pi) < (1-\theta)c(\rho) + \theta c(\tau)$ and hence

$$-c(\pi) > (1-\theta)(-c(\rho)) + \theta(-c(\tau)). \tag{3}$$

Together, equations (2) and (3) imply that $\pi \in E(\text{McSP}) - EE(\text{McSP})$ and hence that

$$E(\text{BQP}) - EE(\text{BQP}) \subseteq E(\text{McSP}) - EE(\text{McSP}).$$

The desired result follows immediately. □

**Example 1.2.** Suppose $\rho, \pi$ and $\tau$ are the only efficient solutions and that (see Fig. 3)

$$l(\rho) = 2,\ l(\pi) = 5,\ l(\tau) = 8, \text{ and } c(\rho) = 0.2,\ c(\pi) = 0.4,\ c(\tau) = 0.8.$$

(It is trivial to construct a network realising these values.) Then $l(\pi) = 0.5l(\rho) + 0.5l(\tau)$ that is, $\theta = 0.5$. Clearly, $\pi \in EE(\text{BQP})$. However, $\pi \notin EE(\text{McSP})$ since $0.5(-c(\rho)) + 0.5(-c(\tau)) = -0.5 < -0.4 = -c(\pi)$.

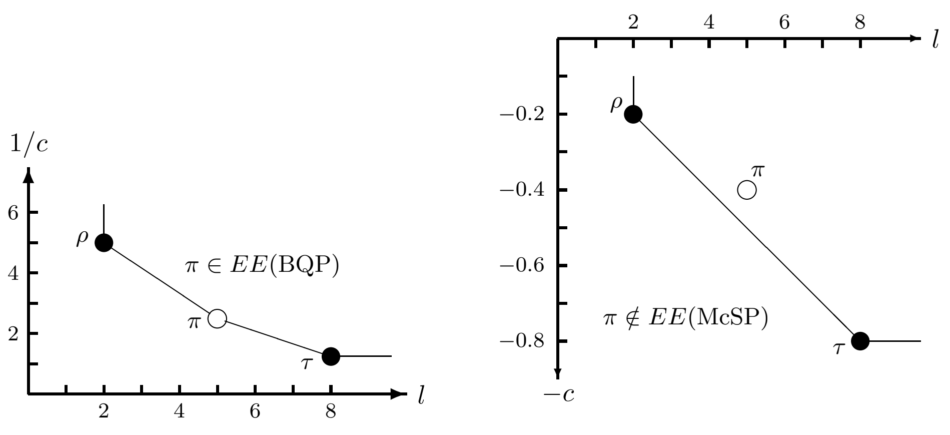

FIGURE 3. Illustration that $EE$(McSP) may be a strict subset of $EE$(BQP).

The implication of Proposition 1.1 part (i) is that, as far as computing the set of efficient solutions is concerned, an algorithm to solve BQP provides an algorithm to solve McSP and *vice versa*. On the other hand, efficient solutions are more likely to be extreme for BQP than for McSP. Viewed differently, an algorithm to find extreme efficient solutions of BQP can be used to find both extreme and "mildly non-extreme" efficient solutions of McSP.

## 2. Algorithms to compute efficient sets

Chen and Chin [6] and others have proposed an algorithm to find a quickest $O - D$ path for a single value of $\sigma$ (see Eq. (1)) based on the observation that the number of distinct path capacity values cannot exceed the number of links in the network since the capacity of a path is equal to the capacity of one of its links. This means that we can determine $E$(McSP) by using the Constraint Method with (all) optimal solutions of

$$\text{McSP}(k): \quad \begin{array}{l} \text{minimise } l(\pi) \\ \text{subject to } -c(\pi) \leq -k,\ \pi \in \Lambda \end{array}$$

being found for all $k \in \mathcal{K} = \{c_\alpha\}$. The requirement that $-c(\pi) \leq -k$ means that $\pi$ must not contain any link a with $c_\alpha < k$. This may be ensured without losing any feasible solutions of McSP($k$) by omitting constraint $-c(\pi) \leq -k$, but working with a modified network obtained by removing all links $\alpha$ with $c_\alpha < k$. With this modification we are left with a conventional single origin single destination shortest path problem for which very efficient algorithms exist.

At this point we need to make a distinction between computing $E$(McSP) and an $RE$(McSP). The former would require finding all shortest paths for each $k \in \mathcal{K}$ (and there may be many), whereas the latter merely requires a single shortest path for each $k$. To simplify our discussion we shall henceforth be concerned with finding an $RE$(McSP) (or an $REE$(McSP)) only – it will be clear that to compute $E$(McSP) could be effected by replacing a shortest path algorithm by a $k$-shortest path algorithm.

A prototype algorithm to compute an $RE$(McSP) now becomes apparent.

## PROTOTYPE ALGORITHM

STEP 1. Form the set $\mathcal{K} = \{k_1, k_2, \ldots, k_r\}$ of distinct capacity values ordered so that $k_1 < k_2 < \ldots < k_r$.

**Repeat** STEP 2 for $i = 1, 2, \ldots, r$.

STEP 2. Find a shortest path $\pi$ (if one exists) from $O$ to $D$ through the network $N_i$, the original network with links of capacity $c_\alpha < k_i$ removed.

STEP 3. Remove the non-efficient solutions from the set $\{\pi\}$.

The calculation is clearly dominated by the applications of STEP 2. If use is made of Dijkstra's algorithm [14], and the priority list of temporary labels is implemented as a Fibonacci heap, then each shortest path can be found in $O(e + n \log n)$ time [15]. This gives a time complexity of $O(re + rn \log n)$, or $O(e^2 + ne \log n)$ since $r$ may be $O(e)$, for our prototype algorithm which is the same as the time complexity quoted by Rosen *et al.* [29] to find a *single* quickest $O - D$ path.

While we have been able to quote a theoretical complexity there is much that remains as regards a practical implementation of the prototype algorithm. Of course, if a single network calculation is required and there is only a very small number of different link capacity values then a straightforward implementation will solve the problem very quickly. From now we shall assume that there are several, or many, different values.

### 2.1. Implementing the shortest path calculations

Let $\pi_i$ denote a shortest path from $O$ to $D$ through the network $N_i$, $i = 1, 2, \ldots$ Initially, a (full) shortest path calculation is performed to calculate $\pi_1$ in $N_1 = N$. However, since $N_2$ results from $N_1$ by removing some of the links, it may be expected that $\pi_2$ will show significant similarities to $\pi_1$. Similarly, $\pi_3$ is likely to be similar to $\pi_2$, but less similar to $\pi_1$ than $\pi_2$ is to $\pi_1$, and so on. That is, the initial calculation to find $\pi_1$ provides knowledge that is potentially useful for guiding successive shortest path calculations. One means of incorporating such knowledge is to use a "consistent function" to guide selection of the next node for scanning.

**Definition 2.1.** A function $h : V \to R$, from the node set of network $N$ to the set of real numbers, is *consistent* if, for every link $xy$, $h(x) \leq a_{xy} + h(y)$ where $a_{xy}$ is the length of $xy$.

In essence, a consistent function provides, for each node $x$, a lower bound from that node to the destination $D$ [2]. If a suitable consistent function is available then Dijkstra's method may be extended to the following algorithm [16,26].

**Algorithm 2.2.** *Dijkstra(h)*

```
STEP 1. (Setup)
    Set dist($x$) $= \infty$, $\forall x \neq O$; dist($O$) $= 0$; $u = O$; $CAND = \{u\}$.
**While** $CAND \neq \emptyset$ perform STEP 2.
STEP 2. (Iteration)
    Remove $u$ from $CAND$ ($i.e.$ $CAND \leftarrow CAND \setminus \{u\}$).
    **if** $u = D$
    **then** terminate **else** for each link $uv$
            **if** $\mathrm{dist}(u) + a_{uv} < \mathrm{dist}(v)$
            **then** $\mathrm{dist}(v) \leftarrow \mathrm{dist}(u) + a_{uv}$; $CAND \leftarrow CAND \cup \{u\}$.
    Select $u \in CAND$ such that $\mathrm{dist}(u) + h(u) = \min_{x \in CAND}(\mathrm{dist}(x) + h(x))$.
```

This specialises to the usual Dijkstra algorithm if $h$ is a constant ($h(x) = h(D)$ for all nodes $x$). (Note that, for brevity, we have omitted all pointer operations for reconstructing a shortest path at the termination of the algorithm.)

Dijkstra ($h$) is only useful if an easily computed non-constant consistent function is available. Moreover, the benefits increase as the values $h(x) - h(D)$ more accurately "estimate" the actual distances from $x$ to $D$. Then, since removal of links cannot decrease distances, $h$, defined by $h(x) = d_i(x, D)$ all nodes $x$, provides a consistent function for McSP with respect to shortest distance calculations for networks $N_{i+1}, N_{i+2}, \ldots$ where $d_i(x, D)$ denotes the shortest distance from node $x$ to node $D$ through network $N_i$. Advantage of this is taken by initially setting $i = 1$ and specifying an integer $m > 1$, then replacing the iteration step of the prototype algorithm by

```
**While** $i \leq r$ perform STEP 2.
STEP 2. (Iteration)
    **if** $i = pm + 1$ for some integer $p$
    **then** determine the set $\{d_i(x, D)\}$ and set $h(x) = d_i(x, D)$ all nodes $x$.
            The shortest $O - D$ path $\pi$ is obtained as a by-product.
    **else** use Dijkstra($h$) to obtain $\pi$.
    **if** $d_i(O, D) = \infty$ ($i.e.$ $O$ not connected to $D$ in $N_i$)
    **then** set $i \leftarrow r + 1$ (early termination)
    **else** set $i \leftarrow i + 1$.
```

Every $m$ iterations the consistent function is updated by performing a (full) shortest path calculation in the network with link directions reversed. This will be termed the "$p$-th *restart*" for $p = 0, 1, \ldots$ If, as is likely to be the case, the network is symmetric then this may be achieved merely by finding the shortest distances from $D$ to every other node. For $m = 1$ the algorithm essentially reverts to the prototype algorithm with conventional shortest path algorithm. For $m > r - 1$, only one consistent function (corresponding to $i = 1$) is used.

## 2.2. Using path capacity information

At the start of the $p$-th restart the shortest path $\pi_p(x)$ is found in $N_{pm+1}$ from each node $x$ to node $D$. It is also straightforward to calculate the capacity $g_p(x)$ of $\pi_p(x)$ at the $p$-th restart, as well as the length of $\pi_p(x)$.

**Proposition 2.3.** *Suppose that Dijkstra(h) is being applied to $N_i$ where $pm+1 < i \leq (p+1)m$. If $u$ is the node selected and $g_p(u) \geq k_i$, then $\pi$, the shortest path in $N_i$ from $O$ to $u$ followed by $\pi_p(u)$ is a shortest path from $O$ to $D$ in $N_i$.*

*Proof.* Since since $i > pm+1$ the shortest path in $N_i$ from $u$ to $D$ must be at least as long as $\pi_p(u)$. However, since $g_p(u) \geq k_i$ every arc of $\pi_p(u)$ must lie in $N_i$ and so $\pi_p(u)$ is indeed the required shortest path from $u$ to $D$. □

Once a node $u$ has been selected for which $g_p(u) \geq k_i$, the search is over and $\pi_p(u)$ may be used to complete the shortest path from $O$ to $D$ in $N_i$. Rough calculations suggested, however, that the effort of calculating the $g_\pi(x)$ would probably not be offset by the reduction in number of nodes added to set *CAND*. Consequently, this modification was not tested.

## 2.3. Extraction of efficient solutions

Step 3 of the prototype algorithm can be streamlined by not calculating unwanted paths (or repetitions) as the algorithm proceeds. A diagram will be used to describe the general way in which this may be achieved.

An ordered list of paths $\mathcal{L} = \{\pi_1, \pi_2, \dots\}$ containing the current candidate efficient solutions is maintained. Also, paths $\pi_i$ are assigned pointers parent($\pi_i$) which are used to extract the extreme efficient solutions at termination. (Details of the pointer calculation will not be given but see Fig. 4.)

Imagine now that lengths of shortest paths for each of the potential capacity values are as shown in Figure 4. At the first iteration ($i = 1$) the shortest path $\pi$ from $O$ to $D$ may be either $\pi_1 = \tau_a$ or $\pi_1 = \tau_A$. Suppose $\pi_1 = \tau_A$ and set $\mathcal{L} = \{\tau_A\}$. Our first (extreme) efficient solution has been obtained and $i$ is increased to 5 (nothing new being obtained by performing step 2 for $i = 2, 3, 4$). On the other hand, if we set $\pi_1 = \tau_a$, then $\mathcal{L} = \{\tau_a\}$ and $i$ is increased to 4. At the next iteration ($i = 4$) $\pi_4 = \tau_A$ is obtained. Since $l(\tau_A) = l(\tau_a)$ but $c(\tau_A) > c(\tau_a)$, $\tau_a$ has been found not to be efficient and so $\tau_a$ is removed from $\mathcal{L}$ and $\tau_A$ added giving $\mathcal{L} = \{\tau_A\}$; $i$ is increased to 5.

By whichever of the two routes, we have arrived at the same situation. At the next iteration ($i = 5$) $\pi_5 = \tau_B$ is obtained, $\mathcal{L}$ is updated to $\{\tau_A, \tau_B\}$ and $i$ increased to 7. After three more iterations $\mathcal{L} = \{\tau_A, \tau_B, \tau_C, \tau_D, \tau_E\}$ and $i = 11$. Now, no further $O - D$ paths remain and the algorithm terminates. It may be noted that Kagaris *et al.* [22] have suggested such a strategy.

We note, but do not give details, that extreme solutions are determined as the algorithm proceeds by means of pointers parent($\pi_i$) (see Fig. 4). (Essentially the pointers are determined so that following them results in a path that "bends to

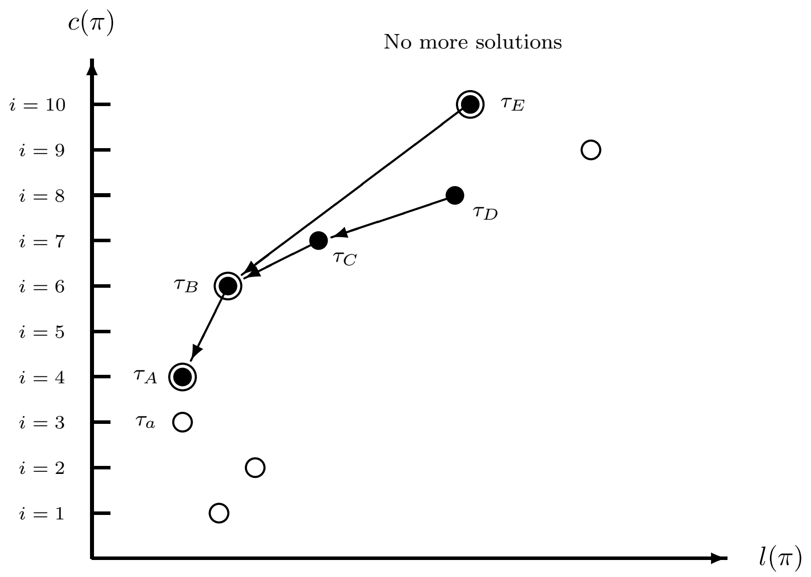

FIGURE 4. For each of the potential path capacity values, only the "leftmost" (shortest) path, if any, is shown. Efficient solutions $\tau_A, \ldots, \tau_E$ are indicated by filled circles; $\tau_A, \tau_B$ and $\tau_E$ are also *extreme* efficient solutions. The arrows indicate parent pointers. (The values of $c(\pi)$ are shown equally spaced but this need not be the case.)

the left".) The extreme efficient solutions for the hypothetical example of Figure 4 are: $\tau_E, \tau_B = \text{parent}(\tau_E)$ and $\tau_A = \text{parent}(\text{parent}(\tau_E))$.

**Algorithm 2.4.** *($\mathcal{EFFSOL}$)*
*(To calculate RE(McSP))*

```
STEP 1 (Setup)
  Form the ordered set $\mathcal{K} = \{k_1, k_2, \ldots, k_r\}$.
  (Or subset of this set if minimal capacity specified – see Sect. 2.5)
  Set $i = 1, q = 1, \mathcal{L} = \emptyset, \delta = 0$. Specify $m$.
While $i \leq r$ perform STEP 2.
STEP 2. (Iteration)
  (2a) if $q = pm + 1$
         then determine $\{d_i(x, D)\}$ and $\pi$; set $h(x) = d_i(x, D)$ all nodes $x$.
         else use Dijkstra($h$) to obtain $\pi$.
  (2b) if $l(\pi) > \delta$ then $\delta \leftarrow l(\pi)$ and add $\pi$ to $\mathcal{L}$.
         if $\delta = \infty$
         then $i \leftarrow r + 1$ (to terminate)
         else $i \leftarrow j + 1$ where $c(\pi) = k_j$; set $q \leftarrow q + 1$.
```

Note that since $i$ increases irregularly we have chosen to restart every $m$ shortest path calculations actually performed (*i.e.* whenever $q = pm + 1$).

## 2.4. Data structures

At each iteration of a label-setting algorithm, such as Dijkstra($h$), it is required to select a candidate node (member of *CAND*) with minimal label (in this case $\text{dist}(u) + h(u)$). That is, *CAND* is maintained as a priority queue which may be implemented as a binary heap, a list of buckets [12], etc. We chose to represent *CAND* by a singly linked circular list of buckets with nodes being inserted into buckets so that the labels are in increasing size. It is easy to see that labels of nodes in *CAND* cannot differ by more than twice the length of a longest arc which enables determination of a suitable number of buckets with which to represent *CAND* – more than 100 buckets were never necessary. Also to take advantage of the fact that shortest paths are found through a sequence of networks on the same node set, fast label initialization was used based on the device introduced in [28].

Networks were kept in forward star form and were already in this format at data input. To form networks $N_i$ explicitly would have required considerable overheads and consequently we formed them implicitly by incorporating a test on arc capacity $c_{uv}$ when scanning node $u$ in step 2 of Dijkstra($h$). Finally we note that all restart shortest path calculations were performed using a label-correcting algorithm [13].

## 2.5. Finding a best compromise solution

It is likely that, in practice, the decision maker would be interested in choosing the final solution from a very restricted range of solutions. We envisage, therefore, that the algorithm might be used along the following lines:

(1) first the decision maker (DM) specifies some upper limit $\ell_{\max}$ on path length and a lower limit $c_{\min}$ on capacity. If the DM is uncertain regarding these parameters then $\ell_{\max}$ can be set to $\infty$ (in practice a very large number) and / or $c_{\min}$ set to $k_1$;
(2) next the algorithm described is run to find all efficient solutions with lengths less than or equal to $\ell_{\max}$ in the modified network with links of capacity less than $c_{\min}$ removed;
(3) the DM then specifies a "target capacity" $\gamma$. The system replies with an efficient solution whose capacity (in $\mathcal{K}$) is as near as possible to $\gamma$ together with efficient solutions with next higher and next lower capacities (where these exist);
(4) if the DM is satisfied with one of the solutions presented then stop, otherwise return to (3).

It is clear that the above can easily be varied by, for example, displaying more than three efficient solutions at a time; giving emphasis to extreme efficient solutions; setting target length instead of target capacity. Other variants can be imagined.

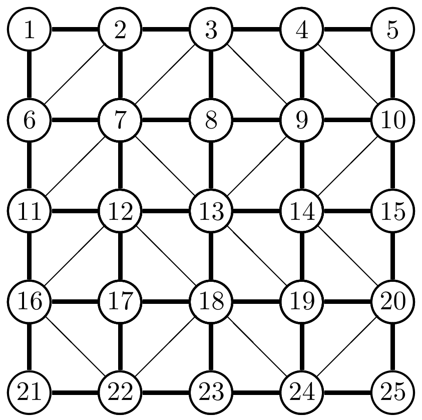

FIGURE 5. A $5 \times 5$ grid network shown by the heavy lines. When the fine diagonal lines are added we have an example of a $5 \times 5$ crossgrid. (Note each line actually represents a pair of oppositely directed arcs.)

## 3. NUMERICAL RESULTS

### 3.1. THE DATA

In order to test the algorithm we used *grid* and *crossgrid* networks, which permit controlled variation of parameters – two real networks were also used. An $a \times b$ grid network consists of $a$ rows each containing $b$ nodes. Every node is linked by an arc to each of its neighbours in the same row or same column (Fig. 5). The arc lengths are generated by sampling an integer from a uniform discrete distribution over the range (100, 1000) then dividing by 10. Nodes are numbered from "left to right" in each row starting with the "top row and working down". An $a \times b$ grid can be turned into a crossgrid by adding "diagonal" arcs (see Fig. 5) the orientation being chosen "NW-SE" or "NE-SW" randomly with probability 0.5. To represent the likely longer lengths of diagonal arcs, their lengths are generated as earlier and then multiplied by 1.4 ($\approx \sqrt{2}$) before dividing by 10.

For convenience we consider square grids ($a = b$) only. This is not seen as a significant restriction since:

(a) various $O - D$ pairs differently disposed relative to the grid, are considered;
(b) the results are not expected to differ markedly unless $a/b$ (or $b/a$) is very small.

When $a$ is odd and at least 3, the specific $O - D$ pairs considered are: A (opposite corners); B (corner – mid opposite side); C (opposite mid sides); D (about centre, two rows and two columns apart). Referring to the network of Figure 5, $O - D$ pairs of types A, B, C and D are (1, 25), (1, 23), (11, 15) and (7, 19) respectively (see also Tab. 2). It should be noted that there is a fundamental difference between the **close** $O-D$ pairs (type D) and the **boundary** $O-D$ pairs (types A, B and C) in that for close pairs the origin and destination nodes are always at most 4 arcs apart.

The capacities are sampled from a uniform distribution over the range $(0.5, r+0.5)$ and then rounded to the nearest integer. That is, there are potentially up to $r$ different capacities available. An $a\times a$ grid with $r$ different arc capacities permitted will be denoted by G$a(r)$; the crossgrid with the corresponding parameters will be denoted by C$a(r)$.

The two "real" networks (Net140 with 140 nodes and 406 arcs, and Net254 with 254 nodes and 760 arcs) relate to portions of the road network of Lancashire in Northern England. These represent rural areas and it is seen that both networks are sparse (in fact sparser than grid networks of comparable sizes). In both cases, arc lengths are based on real distance but capacities were not available and so were generated as for G$a(r)$ with $r = 15$.

## 3.2. Numerical experiments

The effectiveness of the proposed algorithm $\mathcal{EFFSOL}$, will clearly depend on the number of efficient (and extreme efficient) solutions there are, which in turn will depend on the number of possible arc capacities, $r$, and the disposition of the $O-D$ pairs with respect to the network. Section 3.2.1 looks at this dependence and the following subsection tests the use of Dijkstra($h$) without restarts ($m=\infty$) against a straightforward version of $\mathcal{EFFSOL}$ which uses label-correcting throughout.

Having obtained these results we turn to the main purpose of the paper which is testing the performance of $\mathcal{EFFSOL}$ in terms of network size and restart frequency.

### 3.2.1. *Dependence of efficient set size on r*

If there are few possible arc lengths (*i.e.* $r$ is small) then $|RE|$ and $|REE|$ will be small and algorithm $\mathcal{EFFSOL}$ will require few iterations. Indeed, for $r = 1$, only *one* shortest path calculation is required. Consequently, the first experiment investigates the values of $|RE|$ and $|REE|$ for three values of $r$ which may be regarded as being "small" ($r=5$), "intermediate" ($r=15$) and "large" ($r=50$). This is done for the four $O-D$ types A, ..., D.

Results reported in Table 1 are for "medium-sized" (625 node) grid and crossgrid networks (but see Tab. 3 for some results for networks of other sizes). Each data set contains 5 problems and the numbers reported are averages.

As expected, $|RE|$ and $|REE|$ increase with $r$ but it may be noted that the *rate of* increase is less than that of $r$. It follows that, for small $r$, determining sets of efficient solutions is easy. On the other hand, $r = 50$ is likely to be higher than would be required in practice (at least for telecommunications applications). Correspondingly, we restrict $r$ to the single value $r = 15$ from now on unless otherwise stated.

TABLE 1. Dependence of efficient set size on $r$.

| Data set | A(48)* | | B(36)* | | C(24)* | | D(4)* | |
|---|---|---|---|---|---|---|---|---|
| | $\lvert RE\rvert$ | $\lvert REE\rvert$ | $\lvert RE\rvert$ | $\lvert REE\rvert$ | $\lvert RE\rvert$ | $\lvert REE\rvert$ | $\lvert RE\rvert$ | $\lvert REE\rvert$ |
| G25(5) | 2.2 | 2.2 | 2.6 | 2.6 | 2.6 | 2.6 | 2.6 | 2.6 |
| G25(15) | 6.2 | 4.0 | 6.8 | 4.0 | 6.6 | 4.2 | 3.2 | 2.4 |
| G25(50) | 15.6 | 5.8 | 13.2 | 6.0 | 8.6 | 4.2 | 2.0 | 2.0 |
| C25(5) | 2.2 | 2.0 | 3.0 | 2.8 | 3.4 | 3.4 | 3.2 | 2.8 |
| C25(15) | 7.4 | 4.8 | 8.4 | 5.4 | 9.0 | 6.0 | 4.4 | 3.4 |
| C25(50) | 18.6 | 7.0 | 18.4 | 6.4 | 16.6 | 5.6 | 5.6 | 3.2 |

* For each $O-D$ pair, the number in parentheses is the least number of arcs in any path from $O$ to $D$ for grid networks. The corresponding numbers for crossgrid networks would not be greater.

TABLE 2. Results for a simple implementation of $\mathcal{EFFSOL}$.

| Data set–OD pair | $\nu_c$ | $t_c$ | $\nu_h$ | $t_h$ | $\delta\nu\%$ | $\delta t\%$ |
|---|---|---|---|---|---|---|
| G25(15)–A(1,625) | 4389 | 2.12 | 2312 | 1.88 | 47.3 | 11.7 |
| G25(15)–B(1,613) | 4857 | 2.32 | 1868 | 1.47 | 61.5 | 36.6 |
| G25(15)–C(301,325) | 4497 | 2.18 | 1829 | 1.42 | 59.3 | 34.9 |
| G25(15)–D(287,339) | 1881 | 0.92 | 755 | 0.43 | 59.9 | 53.7 |
| C25(15)–A(1,625) | 5625 | 3.58 | 2730 | 2.90 | 51.5 | 18.9 |
| C25(15)–B(1,613) | 6119 | 3.85 | 2449 | 2.60 | 60.0 | 32.3 |
| C25(15)–C(301,325) | 6374 | 4.02 | 2261 | 2.37 | 64.5 | 41.1 |
| C25(15)–D(287,339) | 2629 | 1.81 | 874 | 0.69 | 69.1 | 61.8 |

It is also seen that for the close (type D) $O-D$ pairs there are, as expected, very few efficient solutions for all three values of $r$. To see why this is so, note that for grid networks, $O$ and $D$ are only four arcs apart and so most of the network is largely irrevelevant with only a few paths from $O$ to $D$ being relevant. For other $O-D$ pairs there is not a clear pattern as far as the effect of closeness is concerned.

## 3.3. Testing a simple implementation of $\mathcal{EFFSOL}$

We now compare a simple Dijkstra($h$) and a label-correcting version of algorithm $\mathcal{EFFSOL}$. Results are shown in Table 2: $\nu_c$ and $\nu_h$ are the average total numbers of nodes scanned using the label-correcting and Dijkstra($h$) algorithms respectively, and $t_c$ and $t_h$ are the corresponding times (here and elsewhere in seconds, the programs being written in Ada and run on a Sun 3). $\delta\nu$ and $\delta t$ are the percentage reductions in numbers of nodes scanned and times taken respectively, obtained by using Dijkstra($h$) as compared to label-correcting throughout. It was found that performance is fairly insensitive to size of bucket chosen and for all results in Tables 2 and 3 the bucket size was taken to be 3.0.

TABLE 3. Variation of computational effort with network size.

| Data set | $\|RE\|$ | $\|REE\|$ | $\nu_c$ | $t_c$ | $\nu_h$ | $t_h$ | $\delta t\%$ | $\lambda$ |
|---|---|---|---|---|---|---|---|---|
| G15(15) | 6.0 | 4.2 | 1506 | 0.74 | 672 | 0.54 | 27.8 | 2.38 |
| G25(15) | 6.8 | 4.0 | 4857 | 2.32 | 1868 | 1.50 | 35.5 | 2.39 |
| G35(15) | 6.2 | 4.4 | 9204 | 4.48 | 3413 | 2.62 | 41.5 | 2.14 |
| C15(15) | 7.0 | 3.6 | 1832 | 1.12 | 730 | 0.77 | 31.2 | 4.89 |
| C25(15) | 8.4 | 5.4 | 6119 | 3.84 | 2449 | 2.58 | 32.8 | 4.91 |
| C35(15) | 8.0 | 5.4 | 11669 | 7.52 | 4355 | 4.58 | 39.1 | 4.67 |
| Net140 | 4.8 | 3.2 | 772 | 0.31 | 314 | 0.23 | 25.8 | 1.6 |
| Net254 | 4.2 | 2.4 | 932 | 0.44 | 439 | 0.28 | 36.7 | 1.1 |

It is seen that the percentage reduction in nodes scanned, $\delta\nu$, is considerable being 47% or more (on average) in all cases. However, the % reduction in computing time is always less since scanning generally takes longer with Dijkstra($h$). Nevertheless, the average reduction in time is mostly over 30% and never less than 11%, and so even without restarts Dijkstra($h$) leads to less computation.

The percentage improvement is greatest for the close $O-D$ pairs and this might be expected to be even more pronounced for larger networks. The reason for this is that the use of Perko's device [28] leads to much less initialisation cost since labels are initialised "globally" by a single statement. As regards the boundary $O-D$ pairs, there seems to be a tendency for the improvement to be less the further apart $O$ and $D$ are. From now on we consider grid and crossgrid networks with $O-D$ pairs of type B only – this provides a relatively stern test for our algorithm since in this case the origin and destination are far apart (though not quite as far apart as for type A).

### 3.3.1. *Variation with network size*

The next experiment was directed at finding how the computational effort varies with network size. In all cases, $r$ was taken to be 15 and a bucket size of 3.0 employed.

Table 3 gives some results for grid and crossgrid networks of three sizes ($15 \times 15 = 225$ nodes, $25 \times 25 = 625$ nodes and $35 \times 35 = 1225$ nodes) and with $O-D$ pairs of type B. Results are also given for the two real networks Net140 and Net254, in this case averages being taken over five different sets of $O-D$ pairs. The definitions of $\nu_c$, $\nu_h$, $t_c$, $t_h$ and $\delta t$ are as earlier and $\lambda = 1000 t_h/n$ where $n$ is the number of nodes in the network. On the basis of the $\lambda$-values we conclude that, for networks of the same type, computing time grows roughly linearly with network size. However, the rate of increase is larger for crossgrids than the sparser grid networks. Also, $\lambda$ is lowest for Net140 and Net254 which are even sparser. That is, the indication is that improvement is greater for nearer $O-D$ pairs and for more highly connected networks.

TABLE 4. Effect of restart frequency.

| $m$ | G25(15) | | C25(15) | | G35(50) | | C35(50) | |
|---|---|---|---|---|---|---|---|---|
| | $t_h(m)$ | $S(m)$ | $t_h(m)$ | $S(m)$ | $t_h(m)$ | $S(m)$ | $t_h(m)$ | $S(m)$ |
| 1 | 2.400 | 100 | 3.824 | 100 | 11.948 | 100 | 16.200 | 100 |
| 2 | 1.656 | 69.0 | 2.564 | 67.1 | 7.344 | 61.5 | 10.072 | 62.2 |
| 3 | 1.496 | 62.3 | 2.356 | 61.6 | 6.148 | 51.5 | 8.560 | 52.8 |
| 4 | 1.472 | 61.3 | 2.312 | 60.5 | 5.858 | 49.0 | 7.688 | 47.3 |
| 5 | 1.520 | 63.3 | 2.284 | 59.7 | 5.456 | 45.7 | 7.622 | 47.0 |
| 6 | 1.480 | 61.7 | 2.464 | 64.4 | 5.356 | 44.8 | 7.856 | 48.5 |
| 7 | 1.460 | 60.8 | 2.612 | 68.3 | 5.472 | 45.8 | 7.840 | 48.4 |
| 8 | 1.460 | 60.8 | 2.568 | 67.2 | 5.740 | 48.0 | 7.732 | 47.7 |
| $\infty$ | 1.472 | 61.3 | 2.588 | 67.7 | 7.232 | 60.5 | 10.294 | 63.5 |

### 3.3.2. *Effect of restart frequency*

Recall that, so far, $\mathcal{EFFSOL}$ has been employed with label-correcting being used: either all the time by taking $m = 1$, or at the first iteration only by taking $m = \infty$ (in practice any integer greater than $r - 1$ will do). It is natural to ask whether some intermediate value of $m$ might give better results and the last experiment investigates this briefly. Again the $O - D$ pairs were of type B but the bucket size was changed to 3.5 as now fewer nodes tend to be labeled. Table 4 shows results for the sets of medium sized networks G25(15) and C25(15). To illustrate the effects of $m$ the computation time $S(m)$ for restart period $m$ is expressed as the percentage $S(m) = 100t_h(m)/t_h(1)$ where $t_h(j)$ is the computation time when $m = j$.

For G25(15) no value of $m$ was significantly better than for $m = \infty$. This, though disappointing, is perhaps not too surprising since there are few efficient solutions ($|RE| = 6.8, |REE| = 4.0$ on average). For the crossgrid networks C25(15) there is some improvement becoming evident for $1 \leq m \leq 6$ even though there are not many more efficient solutions ($|RE| = 8.4, |REE| = 5.4$ on average) than for G25(15).

Since the scope for improvement is greater when more capacities are permitted and may be expected to be greater for larger networks, two further sets of results were obtained for data sets G35(50) and C35(50). The right hand columns of Table 4 show that there is appreciable improvement for the range $4 \leq m \leq 8$. Again results are better for crossgrids (with improvement for C35(50) and $m = 5$ being 26% better than without restarts).

## 4. CONCLUSIONS

The Quickest Path problem has been related to the problem McSP of finding a maximal capacity shortest path. For a given data set, the two problems are equivalent in that they have the same set of efficient solution paths. However, for McSP, fewer of these may be extreme solutions.

A method, based on the strategy of Chen and Chin [6], was described for obtaining an $RE$ (or an $REE$) set for McSP. Then, the paper discussed ways in which this algorithm can be implemented effectively by making use of knowledge gained earlier in the calculations (as embodied in the consistent function $h$). This was followed by a short discussion of ways in which a decision maker might use the algorithm to obtain a best compromise solution. Finally, numerical experience of applying the algorithm was presented for grid, crossgrid and two real networks.

We now comment on the data structures used. It was found that the use of a bucketing system can lead to better performance than using a heap. Such gain would be offset if the algorithm needed much tuning as regards the value of the best bucket size. (While the results could undoubtably be improved by tailoring bucket size to data set, this would counter the aim of developing a robust algorithm.) However, it was found that performance was fairly insensitive to bucket size and so the use of buckets seems fully justified.

As regards the overall performance, the greatest gain is in only performing $|RE| + 1$ shortest path calculations rather than $r$, the number of capacities available. (The number is $|RE| + 1$ since there will generally be a final calculation in which it is found that $O$ and $D$ are not connected. Exceptionally, additional calculations will be needed – see the discussion relating to solutions $\tau_a$ and $\tau_A$ of Fig. 4.) It appears that using restarts can also be advantageous for more highly connected networks with many arc capacity values. Such networks, however, seem to be less likely to occur in practice.

The results of Tables 2, 3 and 4 demonstrate the improvements obtained by using our algorithm. These were substantial, though not as good as those obtained for the Bicriterion Shortest Path problem for which the improvement was impressive [3].

In conclusion, we note that the techniques discussed here are potentially useful for implementations of algorithms for related problems such as the $k$-quickest path or the all pairs quickest path problems.

## References

[1] T.B. Boffey, Multiobjective routing problems. *TOP* **3** (1995) 167-220.

[2] T.B. Boffey, *Distributed Computing: associated combinatorial problems.* McGraw-Hill (1992).

[3] T.B. Boffey, Efficient solution generation for the Bicriterion Routing problem. *Belg. J. Oper. Res. Statist. Comput. Sci.* **39** (2000) 3-20.

[4] G.-H. Chen and Y.-C. Hung, On the quickest path problem. *Inform. Process. Lett.* **46** (1993) 125-128.

[5] G.-H. Chen and Y.-C. Hung, Algorithms for the constrained quickest path problem and the enumeration of quickest paths. *Comput. Oper. Res.* **21** (1994) 113-118.

[6] Y.L. Chen and Y.H. Chin, The quickest path problem. *Comput. Oper. Res.* **17** (1990) 179-188.

[7] Y.L. Chen, An algorithm for finding the $k$ quickest paths in a network. *Comput. Oper. Res.* **20** (1993) 59-65.

[8] Y.L. Chen, Finding the $k$ quickest simple paths in a network. *Inform. Process. Lett.* **50** (1994) 89-92.

[9] J.L. Cohon, *Multiobjective Programming and Planning.* Academic Press (1978).

[10] J.R. Current, C.S. ReVelle and J.L. Cohon, The maximum covering/shortest path problem: A multiobjective network design and routing problem. *EJOR* **21** (1985) 189-199.

[11] J.S. Dai, S.N. Wang and X.Y. Yang, The multichannel quickest path problem. *Int. J. Systems Sci.* **25** (1994) 2047-2056.

[12] E.V. Denardo and B.L. Fox, Shortest-route methods: 1. Reaching, pruning, and buckets. *Oper. Res.* **27** (1979) 161-186.

[13] R. Dial, F. Glover, D. Karney and D. Klingman, A computational analysis of alternative algorithms and labelling techniques for finding shortest path trees. *Networks* **9** (1974) 215-248.

[14] E.W. Dijkstra, A note on two problems in connection with graphs. *Numer. Maths* **1** (1959) 269-271.

[15] M.L. Fredman and R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM* **34** (1987) 596-615.

[16] P. Hart, N. Nilsson and B. Raphael, A formal basis for the heuristic determination of minimal cost paths. *IEEE Trans Syst. Man. Cybernet.* **4** (1968) 100-107.

[17] Y.-C. Hung, Distributed algorithms for the constrained routing problem in computer networks. *Computer Communications* **21** (1998) 1476-1485.

[18] Y.-C. Hung and G.-H. Chen, *On the quickest path problem.* Springer, *Lecture Notes in Comput. Sci.* **46** (1991).

[19] Y.-C. Hung and G.-H. Chen, Distributed algorithms for the quickest path problem. *Parallel Comput.* **18** (1992) 823-834.

[20] Y.-C. Hung and G.-H. Chen, Algorithms for the constrained quickest path problem and the enumeration of quickest paths. *Comput. Oper. Res.* **21** (1994) 113-118.

[21] Y.-C. Hung and G.-H. Chen, *The quickest path problem in distributed computing systems.* Springer, *Lecture Notes in Comput. Sci.* **579** (1992).

[22] D. Kagaris, G.E. Pantziou, S. Tragoudis and C.D. Zaroliagis, *On the computation of fast data transmission in networks with capacities and delays.* Springer, New York, *Lecture Notes in Comput. Sci.* **955** (1995) 291-302.

[23] D. Lee and E. Papadopolou, The all-pairs quickest path problem. *Inform. Process. Lett.* **45** (1993) 261-267.

[24] W.E. Leland, M.S. Taqqu, W. Willinger and D.V. Wilson, On the self-similar nature of Ethernet traffic. *IEEE/ACM Trans. Networking* **2** (1994) 1-15.

[25] M.H. Moore, On the fastest route for convoy-type traffic in flowrate-constrained networks. *Transportation Sci.* **10** (1976) 113-124.

[26] G.L. Nemhauser, A generalized permanent label setting algorithm for the shortest path between specified nodes. *J. Math. Anal. Appl.* **38** (1972) 328-334.

[27] V. Paxson and S. Floyd, Wide-area traffic: The failure of Poisson modelling. *Proc. ACM Sigcomm '94* (1995) 149-160.

[28] A. Perko, Implementation of algorithms for $k$ shortest loopless paths. *Networks* **16** (1987) 149-160.

[29] J.B. Rosen, S.-Z. Sun and G.-L. Xue, Algorithms for the quickest path problem and the enumeration of quickest paths. *Comput. Oper. Res.* **18** (1991) 579-584.

[30] R.E. Steuer, *Multiple Criteria Optimization: Theory, Computation and Applications*. Wiley (1986).

to access this journal online:
www.edpsciences.org