Technical Notes

Reducing Space Requirements for Shortest Path Problems

J. IAN MUNRO and RAUL J. RAMIREZ
University of Waterloo, Waterloo, Ontario, Canada
(Received March 1980; accepted November 1981)

The problem of determining the shortest path through a level network using as little space as possible is considered. Let k denote the number of levels and assume each level contains m nodes. A space efficient technique is presented by which the shortest route from a source to a sink may be found in a complete level graph using $\theta(m + k)$ storage locations and a factor of only $\theta(\log k)$ more basic operations than space inefficient methods. If an edge from node p of level i to node q of level $i + 1$ exists only if $p \geq q$, then the space saving technique may also be employed. In this case the run time of the algorithm is at most twice that of conventional approaches.

MANY PROBLEMS solved by dynamic programming techniques can be (and often are) formulated in terms of finding the shortest path in an acyclic k-partite network. Although such problems are easily solved in what amounts to little more than a carefully ordered scan of the graph (Bellman [1957], Dantzig [1960], Floyd [1962], Minty [1957]), the process often proves costly because of space requirements. The graph itself may have to be stored. Even if this difficulty can be overcome, the fact that the entire path (not just its length) is to be determined appears to add storage requirements of the order of the number of nodes in the graph.

Consider networks, such as those of Figure 1, a and b, which consist of a source s, a sink t, and k groups (or levels) of m nodes each. Weighted edges may be presented from s to the nodes of level 1, from nodes of level i to those of level $i + 1$, and from those of level k to the sink, t. Following Paul et al. (1976) we refer to such graphs as *level graphs* or *level networks*. They can be useful in modeling the possible changes in state of a system with time, by letting the levels denote time and nodes within a level the possible states at that time. Particularly interesting cases of level graphs are those which are *complete*, in that every node at level i is connected to every node at level $i + 1$ (Figure 1a), and those which are *dropping* in that there is an edge from node p of level i to node q of level

Subject classification: 112 dynamic programming applications, 321 information systems management.

Operations Research
Vol. 30, No. 5, September–October 1982
0030-364X/82/3005-1009 $01.25

$i + 1$ if and only if $p > q$ (Figure 1b). Our interest is in finding the shortest path through complete level graphs and dropping level graphs. Edge weights are taken to be arbitrary, but it will be assumed that they are either computable or for some other reason the space necessary to store the graph itself can be ignored.

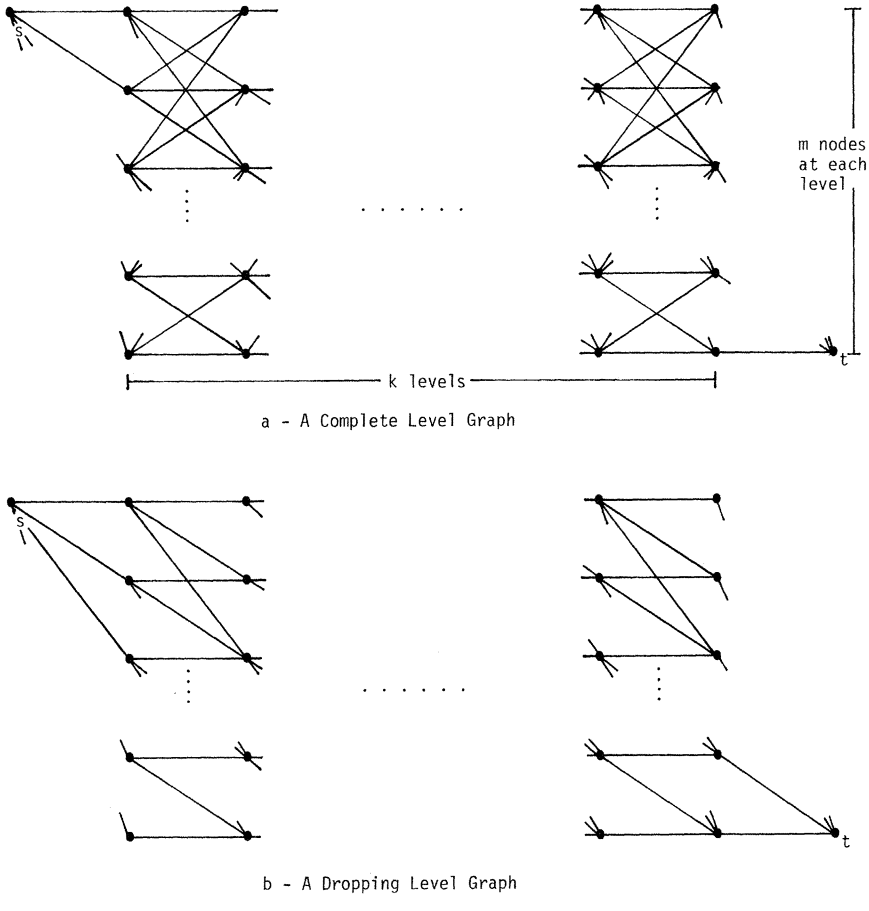

Figure 1. Level graphs.

Any of the straightforward algorithms (Dantzig, Floyd, Minty) finding shortest paths requires inspection of each edge once, and so $\theta(m^2k)$ basic operations for either complete or dropping level graphs. We use $\theta(f)$ to denote an expression bounded above and below by constants times f. Such a run time is clearly near optimal since any algorithm must look at the entire graph. What is disturbing is that these methods will use $\theta(mk)$ units of storage, even when the graph itself need not be stored. The

storage costs of conventional methods are due to the fact that, as the shortest paths from s to nodes at a new level are discovered, either the entire paths are explicitly retained or the penultimate node on the shortest path from s to each node which has been inspected must be retained. This is, of course, done so that the path from s to t is either explicitly stored or is easily reconstructed when the method reaches node t. The storage costs are essentially the same in either case, and at least $\theta(mk)$ no matter how carefully the algorithms are coded. Since many computing facilities base charges on the product of the time and space used, this can have a major impact on the cost of the computation.

The following divide and conquer approach is similar in flavor to the methods of Paul et al. and requires only $\theta(m + k)$ storage with the relatively minor penalty of a factor of $\log k$ in run time for complete level graphs and a factor of 2 (or less) for dropping level graphs. Using the time-space product as a measure of cost, this leads to a saving of a factor of at least $\theta(mk/((m + k)\log k))$ for complete level graphs and $\theta(mk/(m + k))$ for the dropping case.

As we have noted, the basic shortest path algorithms, when applied to networks of the type under discussion, can be viewed as moving from s to t by finding (and retaining) the shortest path from s to each node at level i. From this, paths to nodes at level $i + 1$ are determined, at which time the paths from s to level i nodes are either forgotten or become implicit in their extensions. The space requirement is, then, the product of the path length (k) and the number of nodes at each level (m). If, however, the length of the path were required, and not the path itself, only $\theta(m)$ storage would be necessary. The crux of our method is that not only can this path length be found using $\theta(m)$ space, but also that the midpoint (or any constant number of points) on the path may also be found without affecting the order of magnitude of the space requirement. This is achieved by simply retaining the points at level $k/2$ which were passed through on the shortest paths to nodes under consideration at subsequent levels (i.e., levels greater than $k/2$).

The path finding method then proceeds recursively as follows: To find the shortest path from s to t

(i) Find the length of the shortest path from s to t and the midpoint, mid, of this path. On completion the value of the length may be discarded.
(ii) Recursively, find the shortest path from s to mid, and from mid to t.

If it were acceptable to produce the points on the shortest path in an arbitrary order, this method could be implemented using only $\theta(m + \log k)$ space. The $\log k$ term comes from the overhead required for the depth

of recursion. The reasonable constraint that the path be retained in memory alters this to $\theta(m + k)$. It is not difficult to argue that this is within a constant factor of the space requirements of any algorithm for solving the problem, no matter how long its run time may be.

This leaves the question of whether or not the run time of this algorithm is acceptable. Since the straightforward solutions look at each edge once, gkm^2 (for some positive constant g) is a reasonable estimate of the number of basic operations used to find the length and the midpoint of the shortest path in a complete level network. Let $C(k, m)$ denote the run time (number of basic operations) required by the technique outlined above for the complete case, and $D(k, m)$ that for a dropping level

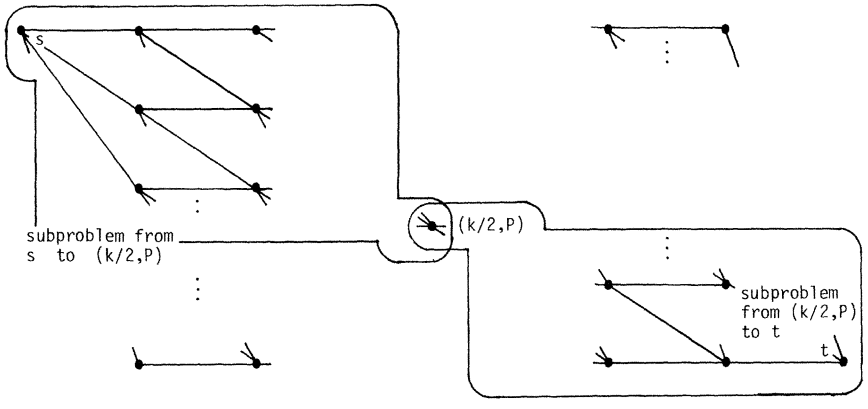

Figure 2. A space efficient algorithm for dropping level graphs. On determining the midpoint $(k/2, P)$ of the shortest path from s to t, recursive calls are made to solve the problem from s to $(k/2, P)$ and $(k/2, P)$ to t.

network. Then, ignoring the trivial issue of the parity of k,

$$C(k, m) = gkm^2 + 2C(k/2, m) \text{ and } C(1, m) = \theta(m).$$

Hence $C(k, m) < gm^2 (\log k) + \theta(m)$ (all logarithms are taken to base 2).

The dropping case is somewhat better behaved. Once the midpoint is discovered, all nodes below and to the left of it can be neglected in solving the s to midpoint subproblem, as can those above and to the right for the other subproblem. This is illustrated in Figure 2. Letting $(k/2, p)$ denote the position of the midpoint we have the recursion

$$D(k, m) < gkm^2 + D(k/2, p) + D(k/2, m - p + 1)$$

and $$D(1, m) = \theta(m); D(k, 1) = \theta(k).$$

This is easily solved (see, for example, Liu [1977]) to show

$$D(k, m) < 2gkm^2 + \theta(k + m).$$

The run time is at most a little more than twice that of the less space efficient method.

One could, of course, determine not just the midpoint on the initial pass, but nodes k/r, $2k/r$, $\cdots$ $(r-1)k/r$ of the shortest route for any fixed r. The space requirements increase by roughly a factor of r over those of finding the length of the shortest path; however, the run time will be roughly $gm^2kr/(r-1)$ in the case of a dropping graph.

The technique we have outlined has been implemented and found useful in solving a problem of data base reorganization. As modifications are made to a data base, the original structures used may become inefficient or downright inappropriate. For example an originally balanced binary tree could become unbalanced or a hash table could fill. Even worse, the mix of requests could change, and so, a hash table which is ideal for finding correctly specified elements could be used, with alarming frequency, to attempt to determine the closest match to a given request. Clearly in such an environment one must eventually pay the price of reorganizing the data and perhaps using completely different structures. The question is "When to do so?" This problem is modeled as a dropping level graph (Tompa and Ramirez [1979]) and the method outlined in this paper can be employed in its solution. We feel this example simply illustrates the potentially wide range of applications of the method. Such applications may occur not only in determining shortest paths through particular types of networks, but also in other problems whose conventional solutions have high storage requirements.

ACKNOWLEDGMENT

The authors are pleased to thank Frank Tompa for his comments and for many fruitful discussions on this and closely related subjects. This research was supported under NSERC grant A8237.

REFERENCES

BELLMAN, R. 1957. *Dynamic Programming.* Princeton University Press, Princeton, N.J.

DANTZIG, G. B. 1960. On the Shortest Route Through a Network. *Mgmt. Sci.* **6,** 187–189.

FLOYD, R. W. 1962. Algorithm 97, Shortest Path. *Commun. ACM* **5,** 345.

LIU, C. L. 1977. *Elements of Discrete Mathematics.* McGraw-Hill, New York.

MINTY, G. J. 1957. A Comment on the Shortest Route Problem. *Opns. Res.* **5,** 121–128.

PAUL, W. J., R. E. TARJAN AND J. R. CELONI. 1976. Space Bounds for Games on Graphs. In *Proceedings of the 8th Annual ACM Symposium on Theory of Computing,* pp. 149–160.

TOMPA, F. W., AND R. J. RAMIREZ. 1979. Selection of Efficient Storage Structures, Computer Science Dept. Research Report CS-79-40, Univ. of Waterloo.