Joint Routing in Networks[1]

Ali Boroujerdi, Changhui Dong, Qiuming Ma, and Bernard M.E. Moret
Department of Computer Science
University of New Mexico
Albuquerque, NM 87131-1386

ABSTRACT

Joint routing of objects through a network consists of finding a set of time-disjoint routes for the objects that maximize the number of objects routed from the source(s) to the sink(s) and minimize the total cost. We show that maximizing the number of objects sent along time-disjoint routes can be modelled as a dynamic network flow problem. We study both this problem and its minimum cost version under several variants and on different types of networks, establishing tight bounds on the running time of exact solutions through applications of existing and some new methods.

1 Introduction

Joint routing is the problem of generating jointly optimal routes through a network for vehicles or other mobile objects. Applications include mission planning, scheduling of flight corridors, design of transport routes for hazardous materials, and also communications within a network. Parameters and constraints may include edge lengths and costs, edge or node capacities, time windows (all objects must arrive at the sinks within some prespecified interval), and others. The objective is to generate jointly optimal routes that obey the constraints and get as many objects as possible (or some prespecified number of objects) from sources to sinks.

A number of related problems appear in the operations research literature, notably the vehicle routing and scheduling problem and other transportation problems: surveys include [2] and [8] and a recent issue of *Operations Research* [6] was devoted to the problem of aircraft scheduling and routing. Of all these models, however, only one closely resembles the joint-routing problem: the dynamic traffic assignment problem as modelled through flow in a space-time network [10], a direct application of the dynamic network flow technique of Ford and Fulkerson [3]. The important difference between the two is that traffic assignment models traffic over a predetermined period of time, while joint routing (at least in the absence of time windows) has an open-ended time scale.

In the following, we briefly review the dynamic network flow model of Ford and Fulkerson, discuss its application to joint routing, and analyze the running time of various approaches. In particular, we prove that, if there exists a feasible set of routes

[1]This work is supported by the Naval Research Laboratory under contract N00014-92-C-2144.

in a network, then there exists a set of routes where each route crosses at most $O(|V|)$ edges, even on networks with cycles; this result allows us to bound the number of time steps that must be considered in the joint routing problem and thus allows us to conclude that joint routing can be solved in polynomial time.

2 The Problem

A formal statement of our problem (in its basic version) can be written as follows.

- Given a graph with two distinguished vertices, a sink and a source, route objects from the source to the sink, subject to the constraints:

  1. objects are initially placed at the source;
  2. at each step, every object crosses an edge of the graph (no object can stop at a vertex for one or more steps);
  3. edge or vertex capacities limit the number of objects that can cross an edge or a vertex at any given step.

  In addition, there may be a time window during which all objects must reach the sink and there may be multiple sources or multiple sinks.

  The goal may be to maximize the number of objects that can be routed according to the constraints; or to find the routing that sends the largest number of objects for the least cost; or to find a least-cost routing for a given number of objects.

A key assumption underlying the problem is that objects and parameters are uniform. In the absence of such uniformity, the problem is generally $\mathcal{NP}$-hard.

Theorem 1. The joint routing problem is $\mathcal{NP}$-hard under any of the following conditions:

- objects move at different speeds
- certain edges can only be crossed by certain objects
- the cost of crossing an edge depends on the object crossing it
- the cost of crossing an edge depends on the time step
- the total cost is the sum of the costs of edges that are crossed at least once

□

The first four of these $\mathcal{NP}$-hardness results are new, but not unexpected (we omit our proofs, since the constructions are not of special interest); the fifth is known—it is the "minimum edge-cost flow" problem mentioned in Garey and Johnson [4, p. 215].

3 Dynamic Network Flow

In a 1958 paper, Ford and Fulkerson [3] showed how to transform the *maximum dynamic flow problem* to a conventional network flow problem. In a dynamic network flow problem, arcs have traversal times as well as capacities and flow received at a node can either be moved immediately or held over for later discharge; the question is to find the maximum flow that can be moved from the source to the sink in some given period of time T. The reduction uses the given network, $G = (V, E)$, to build a new network, $G' = (V', E')$, as follows. For each node, $v \in V$, create $T + 1$ copies: $v_0, v_1, \ldots, v_T$; in addition, set up a supersource, s^*, and a supersink, t^*. Let $t(v, w)$ be the traversal time of arc (v, w); introduce an arc from v_i to w_j (of capacity equal to that of arc (v, w)) if and only if $j - i = t(v, w)$; in addition, introduce an arc from v_i to v_{i+1} with infinite capacity for each $v \in V$ and $0 \leq i < T$; finally, connect the supersource to every copy of the original source and connect every copy of the original sink to the supersink, all with arcs of infinite capacity. Then the maximum amount of flow that can be moved from the source to the sink in T time units in G equals the maximum flow from s^* to t^* in G'. If the flow received at a node, v, must leave it immediately, the reduction omits the arcs from v_i to v_{i+1} for all $v \in V$ and all i and omits the supersource—the source for the new network is simply s_0. Note that these reductions create a new graph that is T times larger than the original.

Dynamic network flow has since been used by many researchers in modelling various vehicle routing and scheduling problems, particularly in aircraft problems (see, e.g., [2, 6, 7]). We observe that dynamic network flow can be used to solve the basic version of our joint routing problem.

Theorem 2. Moving as many objects as possible from the source to the sink in T time units while avoiding collisions is equivalent to solving a dynamic network flow problem (in which flow cannot be stored at nodes) on a network with vertices of unit capacity. $\square$

Our earlier discussion establishes the correctness of this theorem; we are not aware of any previous statement of it, although uses of dynamic network flow in vehicle routing and aircraft scheduling offer a close parallel.

4 Results on Joint Routing Without Costs

In order to establish a bound on the running time of a solution algorithm, we need to bound the number of time steps, T, that must be used in the reduction to dynamic network flow.

Theorem 3. Routing a maximum number of objects from source to sink without collisions can be solved in polynomial time. $\square$

Proof: We have seen that we can answer the question by solving a dynamic network flow problem; hence we need only place a bound on T that remains polynomial in the size of the network. If the network is a directed acyclic graph, there is no problem: no object can take longer than $n-1$ steps (where n is the number of vertices in the graph) before reaching the sink. If the network allows cycles (either as an undirected or as a directed graph), a tight bound is more difficult to derive (see below), but a safe upper bound of n^2 steps is easily established as follows.

Consider the state of the system during the first n steps; if, at the end of n steps, one of the objects has reached the sink, then the problem reduces to routing $n-1$ objects through the network and the conclusion follows. If no object has reached the sink by step n, then there must exists some index $i < n$ at which the object closest to the sink (in terms of the number of edges along a shortest path to the sink) is $n-i$ edges away from it. Pick the largest such index: the object closest to the sink can now safely be moved directly to the sink along the shortest path, since no other object can collide with it along that path; the other objects are moved as in the original sequence. The new sequence now sends at least one object to the sink in n steps without collision, reducing the problem to the first case.

Q.E.D.

When time windows are added (say a minimum time of T_0 and a maximum time of T_1), they can be included in the dynamic network by the simple expedient of (i) duplicating the network only for the first T_1 steps; and (ii) removing all arcs leading into the sink from vertices with associated times smaller than T_0. Note that deadlines do not affect the proof of the theorem—nor, indeed, any of the results below.

In fact, the number of edges crossed by any of the objects on its way from source to sink remains linear in the size of the graph, rather than quadratic.

Theorem 4. Given a graph with two distinguished vertices (a source and a sink), if k objects can be routed from the source to the sink without collision, then they can be so routed in $O(n)$ steps, where n is the number of vertices of the graph. □

Proof: We provide a proof for the case of undirected graphs with edges of unit lengths. A proof for the case of directed graphs would involve a detailed case analysis. We prove that, in an undirected graph, object $i+1$ need never arrive more than 2 units of time after object i; since the first object to arrive can always arrive in at most n steps, this establishes a bound of $n+2(n-1)$ steps on the arrival time of the last object.

If there is a feasible joint routing schedule for the n objects, then there is a feasible schedule which minimizes the sum of all arrival times; in this schedule, we index each object by its order of arrival to the sink—we shall use induction on this index. Consider then the ith object and look at the last cycle it traces on its path from source to sink. This cycle is present to avoid a collision with some object j at some vertex v; from v to the sink, object i has a path without cycle. We claim that we must have $j < i$; if the cycle were to avoid collision with a slower object ($j > i$), we could reduce the sum of all arrival times by letting object j use object i's path from v to the sink and thus arrive faster with no collision. Now, observe that the last cycle traced by object i has length 2; otherwise, we could reduce the sum of all arrival times by shortening the cycle to a length of 2 without creating new collisions for object i (rather than going around the cycle, object i would simply traverse the first edge of the cycle forward and backward). Hence object i arrives at vertex v 2 units of time after object j and thus arrives no more than 2 units of time after object j at the sink. Since $j < i$, object j obeys the induction hypothesis and our conclusion follows.

Q.E.D.

A more detailed analysis shows that there exists a schedule in which the last object to arrive to the sink arrives after at most n times units—not $3n - 2$. If object $i + 1$ does arrive 2 units after object i (for $i > 1$), then the network includes at least one extra vertex associated with the path taken by object $i + 1$, since the forced delay implies either a collision forcing object $i+1$ to loop back or a separate path. In either case, we need at least one extra vertex, unless the objects involved are 1 and 2, in which case the extra vertex can simply be the source. On the shortest path of length s, there are $s + 1$ vertices; there are at least $k - 1$ other vertices connected to the source to allow routing of k objects; there are thus at most $e = n - s - k$ "extra" vertices. Now, the first object takes s steps and the second takes at most $s+2$ steps; by using the extra vertices, up to $\min(e, k - 2)$ of the remaining objects can also be delayed by 2 steps each, with the rest being delayed by 1; so we get a maximum delay of $(s+2)+2\min(e, k-2)+(k-2-\min(e, k-2)) = s+k+\min(e, k-2)$. We conclude that, in a graph with n vertices in which we can route k objects and in which the shortest path from source to sink has length s, there exists a routing in which the last object to arrive arrives after at most $s + k + \min(n - s - k, k - 2) \leq n$ steps.

While these results indicate that joint routing without edge costs is polynomially solvable, the solution algorithm appears inefficient, since it is a maximum flow algorithm running on a graph expanded by a factor of n; as most maximum flow algorithms scale according to n^3, the expansion appears to cost us a factor of n^3 in running time. However, the networks used in our model have unit vertex capacities, so as to avoid collisions at vertices. In this case, the expansion creates, in effect, a unit network (i.e., a network where all capacities are unity and every vertex has either in- or out-degree of one). In a unit network of n vertices, only $O(\sqrt{n})$ stages are needed using a blocking flow method. Thus, given a network $G = (V, E)$, we can expand it according to Ford and Fulkerson and to our Theorem 2 to a network with $\Theta(|V|)$ time steps and thus with $\Theta(|V|^2)$ vertices and $\Theta(|E| \cdot |V|)$ edges; replacing vertices of unit capacity does not change the asymptotic sizes and yields a unit network, on which a blocking flow algorithm thus runs in $O(|E| \cdot |V|^2)$ time. Edmonds and Karp's algorithm (the simplest of all network flow algorithms, which augments along the shortest augmenting path) also runs in $O(|E| \cdot |V|^2)$ time, since it uses up to $|V|$ augmentations (because the maximum flow cannot exceed $|V|$), each running a breadth-first search in a graph with $|E| \cdot |V|$ edges. In practice, the fastest network flow algorithm appears to be the gravity flow algorithm (also called the push-relabel algorithm) of Goldberg and Tarjan. (For a survey of network flow algorithms, see the recent text of Ahuja, Magnanti, and Orlin [1]; for evidence about the efficiency of the gravity flow algorithm, see the results of the 1992 DIMACS Challenge [5].) We are unaware of results concerning the behavior of the gravity algorithm on a unit network; we can see that the algorithm profits from the unit capacity of edges, since this property makes every push into a saturating one and thus reduces the running time to $O(mn)$ on a graph of n vertices and m edges.

If our concern is not to maximize the number of objects sent from source to sink, but simply to find a joint routing of k objects for some feasible $k < |V|$, the running time can be further improved. All augmentation methods increase the flow by at least one unit at each step; since the total desired flow is simply k, at most k augmentations will be needed.

Finally, in a sparse network, we expect that only a few of the n duplicates (of vertices or edges) will be used. In order to reduce storage costs, we can maintain a list of used time slots for each vertex and for each edge. The list manipulations add some expense to the running time, but gain considerable space and, on average, still expend very little time per edge or vertex in identifying the correct slot.

5 Results on Joint Routing with Costs

The min-cost max-flow problem has long resisted strongly polynomial solutions. Tardos [9] first provided such an algorithm for minimum-cost circulations and thus also for min-cost max-flow problems. However, in the case of networks with polynomially bounded maximum flow, min-cost max-flow problems are easily solved in (strongly) polynomial time by the simplest of all min-cost flow algorithms: at each step, augment along a path of minimum cost. At each step, the flow must increase by at least one unit; since the max-flow cannot exceed some polynomial bound B (in our case, $B = |V|$), the number of augmenting steps is polynomial. Each step requires solving a shortest-path problem, which can be done with Dijkstra's algorithm (implemented with Fibonacci or relaxed heaps) in $O(|E| + |V| \log |V|)$ time. The complete algorithm thus runs in $O(B \cdot (|E| + |V| \log |V|))$ time.

We can show that Theorems 3 and 4 remain true for min-cost flow problems on unit (or constant) capacity networks, for the same reason: the maximum delay for the last object remains $O(|V|)$. Thus we can obtain joint routes of minimal total cost for a maximum number of objects in $O(|V|^2 \cdot (|E| + |V| \log |V|))$ time, while joint routing of k objects at minimum cost with the same algorithm runs in $O(k|V| \cdot (|E| + |V| \log |V|)$ time. In other words, on a network of reasonable density, the minimum-cost joint routing problem can be solved within the same bounds as the basic joint routing problem.

6 Further Work

Extensions to multiple sources and sinks is straightforward; other parameters can also be included with no effect on the asymptotic running time.

We have implemented several versions of these algorithms for joint routing both with and without costs. In all cases, the simplest methods are the fastest on our (to date somewhat limited) suite of test cases; in particular, the scaling circulation methods proposed for the min-cost problem are slow when compared to the simple method of augmenting along a path of minimum cost—the more sophisticated methods spend too much time in establishing labellings at each iteration. Further experimentation is needed to determine where the bottlenecks are and whether they can be ameliorated.

However, these exact methods suffer from two serious flaws for applications in aircraft routing: (i) the running time and space are too large for many real applications, where the network is extracted from a geographic database and may include millions of vertices and edges; and (ii) the assumption of fixed edge costs is unrealistic—in practice edge costs depend on the type of aircraft and, due to FAA regulations or military engagement rules, edge costs may depend on the routes chosen for other aircraft. In other words, the real-world models lead to $\mathcal{NP}$-hard problems. Gravity flow methods appear well suited for the development of fast heuristics for these intractable versions.

References

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows.* Prentice-Hall, Englewood Cliffs, N.J., 1993.

[2] M. Bielli, G. Calicchio, B. Nicoletti, and S. Ricciardelli. The air traffic flow control problem as an application of network theory. *Comput. and Operations Research*, 9(4):265–278, 1982.

[3] L.R. Ford and D.R. Fulkerson. Constructing maximal dynamic flows from static flows. *Operations Research*, 6:419–433, 1958.

[4] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to NP-Completeness.* W.H. Freeman, 1979.

[5] D.S. Johnson and C.C. McGeoch. DIMACS implementation challenge workshop: algorithms for network flows and matching. Tech. Report 92-4, DIMACS, 1992.

[6] *Operations Research*, 41(1), 1993. (Special Issue on Vehicle and Aircraft Routing and Scheduling.)

[7] J.B. Orlin. Maximum throughput-dynamic networks flows. *Mathematical Programming*, 27:214–231, 1983.

[8] M.M. Solomon. Time window constrained routing and scheduling problems. *Transportation Science*, 22(1):1–13, 1988.

[9] É. Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5:247–255, 1985.

[10] D.J. Zawack and G.L. Thompson. A dynamic space-time network flow model for city traffic congestion. *Transportation Science*, 21(3):153–162, 1987.