Shared Shortest Paths

Problem and Motivation:

In undirected graphs, shortest paths are well understood; however, the concept of shared shortest paths is a novel extension of this classic concept. A shared path is a path in which the cost of one or more edges is evenly split between at least two journeys. A journey is a pair of vertices (s, t) that searches for the lowest cost of traversing from s to t, possibly sharing edges with other journeys. A shared shortest path is a shared path where no journey has incentive to change its path to have a lower cost. Thus a shared shortest path is a stable agreement between the journeys to share costs. Finding shared shortest paths can be related to reducing shipping costs for multiple shippers, carpooling, and specialized networking problems.

Background and Related Work:

The knowledge needed to understand this research is an understanding of graphs, shortest paths, and a basic knowledge of Dijkstra's algorithm [4] and the Floyd-Warshall algorithm [5]. A simple knowledge of game theory, especially of cooperative game theory is useful but not necessary. The concept of shared shortest paths is new but based upon work done on (single) shortest paths.

The problem begin as an extension of a problem in cooperative game theory, in which multiple agents fairly share the costs of shipping from (or to) a common source vertex to (or from) other vertices in a tree. Because in a tree there is only one path between any two given vertices, the original problem is not concerned with finding the optimal (or stable) paths for the agents to share. Research has been done in allocating costs in a shared spanning tree network (for example, [1][2]). However, these algorithms compute the minimum spanning tree of the graph, and then allocate costs to all journeys based on how much of the spanning tree was used by each journey. The cost of traversing the spanning tree may not be the shortest cost available to each journey. Steiner trees [3] try to minimize the total cost of visiting the given set of terminals, as opposed to our approach which tries to find the lowest-cost shared path for each journey, where feasible.

Approach and Uniqueness:

If this game theory problem is expanded to include weighted graphs, finding an optimal (or stable) shared path is no longer a trivial task. Likewise, the original problem can be expanded and complicated by allowing the agents, or journeys, to not share a common source or destination vertex. Through analyzing the shared shortest path problem, we found several new concepts. First, we found and implemented a polynomial-time algorithm to solve the two-journey case. Second, in analyzing the three journey case, we found a problem called "bouncing" that makes an optimal, stable solution for more than two journeys impossible for some problems. Thus, we designed and implemented heuristics for problems containing more than two journeys.

Results and Contributions:

Two-Journey Algorithm

The two-journey algorithm, which was developed and implemented, takes two journeys J_0 (s_0, t_0) and J_1 (s_1, t_1) and a graph G as input. This algorithm finds the shared shortest path by examining all possibilities of sharing edges and returns the path where both journeys have the most savings, if a path exists where both journeys save over their single shortest path. This algorithm uses the Floyd-Warshall algorithm to find the shortest paths for all pairs of vertices and uses some of the simplest ideas of cooperative game theory to find the shared shortest path. This algorithm finds and then compares the cost of the two journeys' shared edges between all possible pairs of

vertices in the graph. This algorithm finds the optimal solution in $O(V^3)$ time, where V is the number of vertices in the given graph.

The algorithm:
Step 1: Find the cost of going between any u and v, using the Floyd-Warshall algorithm.
Step 2: Find the total cost of each shared path (each possible i-j pair) for J_0. For each i-j pair, add the cost from J_0's source (s) to i, the shared cost from i to j, and finally the cost of going from that j to J_0's terminal (t).
Step 3: Repeat Step 2 for J_1.
Step 4: Order the costs of each i-j pair for J_0, in order of increasing cost.
Step 5: Repeat Step 4 for J_1.
Step 6: Remove all i-j pairs from the list for J_0 that have higher cost than the shortest path of J_0.
Step 7: Repeat Step 6 for J_1.
Step 8: Find the shared path that has the lowest total cost for each journey in their lists of costs. This shared path is the shared shortest path because it is stable.

This algorithm is based on the fact that two journeys will never share more than one path in a shared shortest path. As an informal proof by counterexample, let J0 and J1 be journeys on graph G where the shared shortest path of both journeys can be broken down into sections: S_0, S_1, S_2, S_3, and S_4. Let these sections be defined as shown in Figure 1. S_2 leads to a contradiction. For any journey to traverse between i and j either there is one shortest path, or there are multiple paths of the same cost. If there is only one shortest path between i and j, then at least one of the journeys is not using the shortest path and therefore could save by using the shortest path, which contradicts the assumption that S_2 is part of a shared shortest path. On the other hand, if there are at least two shortest paths of the same cost between i and j where J_0 and J_1 are traveling on different paths for the same cost, both journeys could increase their savings by sharing one shortest path, which again contradicts the assumption that S_2 is part of a shared shortest path. Therefore, two journeys can at most share one path. Thus all shared shortest paths for two journeys can be broken down into three, possibly empty, sections: S_0, S_1, and S_2. Let these sections be defined as shown in Figure 2. Viewing the problem this way makes finding the shared shortest path relatively simple, since finding the i and j between which both journeys agree to share finds the shared shortest path.

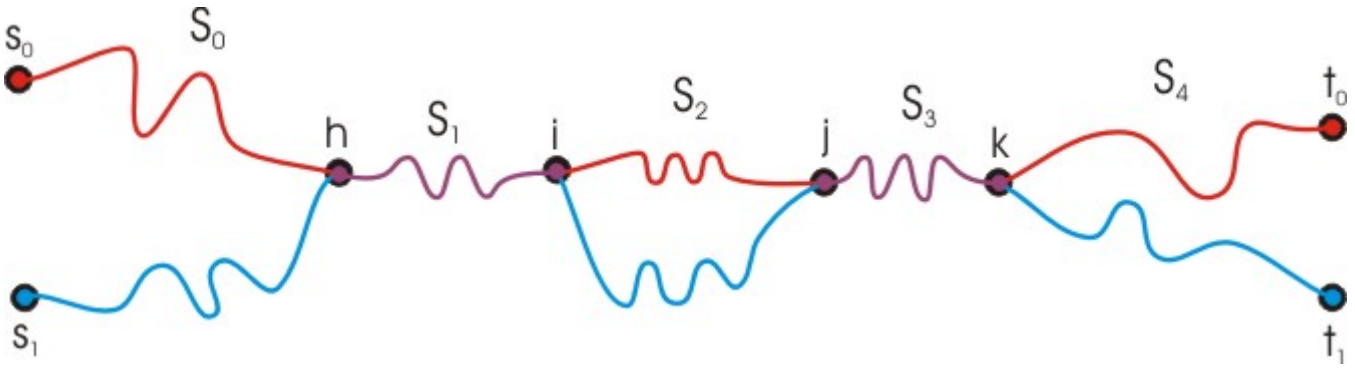

Figure 1

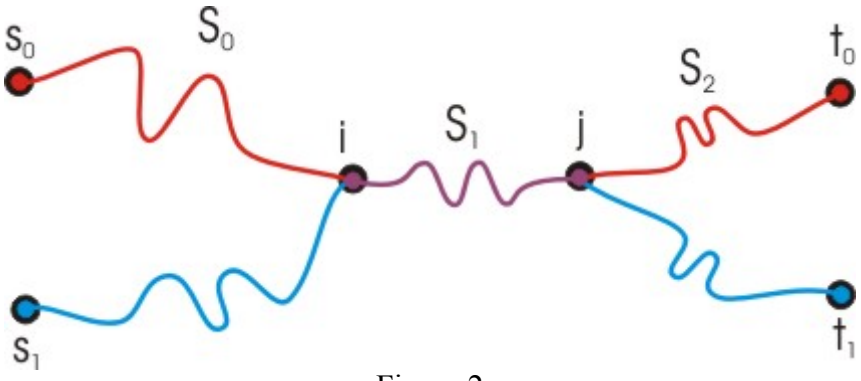

Figure 2

Bouncing Problem

As mentioned above, the problem of bouncing can arise when there are more than two journeys in the given problem. Bouncing occurs when journeys try to outbid each other to minimize their individual costs by offering to lower the cost of other journeys in a repetitious infinite loop. When it arises, there is no stable solution that reaches an equilibrium point where no journeys can find a lower-cost shared path.

One of the simplest examples is as follows: The graph in Figure 3 contains three journeys (s_{All} = S): J_0 (t_0 = X), J_1 (t_1 = Y), and J_2 (t_2 = Z). Since there is no shared path that all three can traverse that decreases the cost for all three journeys, two journeys pair off to form shared paths that do not contain the third journey. This can lead to any one of the shared paths in Figure 4, 5, or 6. However, whichever journey is not part of the current shared path such as J_2 in Figure 4 is now willing to make a shared path that is cheaper for one of the other journeys such as J_1 in Figure 4. So these two journeys would share this shared path in which they both pay less such as Figure 4 becoming Figure 5. But this leaves one of the three journeys left out, and the same type of switching occurs, which leads to bouncing. Thus, Figure 4 becomes Figure 5, Figure 5 becomes Figure 6, and Figure 6 becomes Figure 4. Thus there is no shared shortest path for these three journeys on this graph.

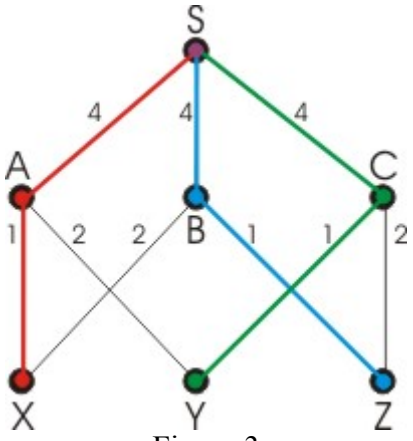

Figure 3

Single Shortest Path Costs

J_0 (S, X) Cost: 5

J_1 (S, Y) Cost: 5

J_2 (S, Z) Cost: 5

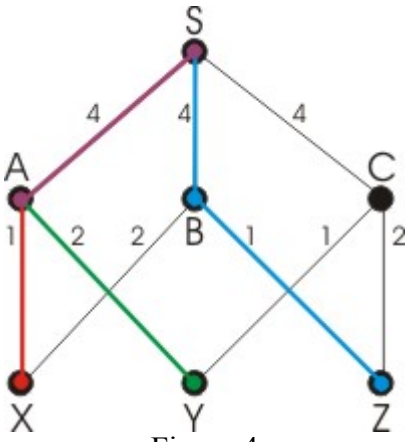

Figure 4

Shared Path Costs

J_0 Cost: 3

J_1 Cost: 4

J_2 Cost: 5

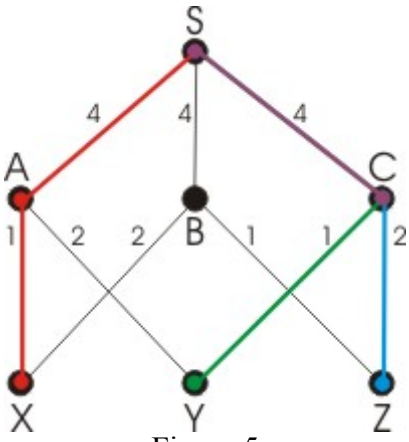

Figure 5

Shared Path Costs

J_0 Cost: 5

J_1 Cost: 3

J_2 Cost: 4

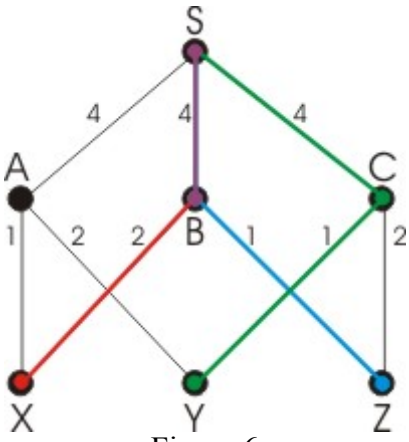

Figure 6

Shared Path Costs

J_0 Cost: 4

J_1 Cost: 5

J_2 Cost: 3

Heuristics

Twenty-three heuristics for a variable number of journeys were designed that are based on adding one journey at a time to the graph along its shortest path. Each heuristic had a different criterion for deciding the next journey to add to the shared path. Two examples are lowest cost of shortest path and the lowest number of edges in their shortest path. These heuristics avoid the problem of bouncing by not allowing any journey that has already been added to the shared path in the graph to change its path even if there is a shorter path that could be used by sharing different edges with other journeys.

The heuristics in general saved on average 42 percent, based upon 56 tests run covering seven different styles of graphs, and three different orders of graphs (10, 100, or 500 vertices), using various numbers of journeys with ten random graphs in each test. The criteria for the top five heuristics in order of most saved on average are lowest cost of single edges, lowest cost of all edges, lowest number of single edges, general least expensive edge, and least expensive edge. This ranking is based on the tests run as described above. Larger graphs understandably were more likely to yield savings and shared paths than small graphs were. The only graphs to produce no savings at all were some 10-vertex graphs. Some styles such as a highway style were also more likely to yield savings. For a concrete example, Figure 7 shows a graph with three journeys. Figure 8 shows the shared path that was found by 18 of the heuristics, which include the heuristics listed above as the five best. Figure 9 shows the shared path found by the other five heuristics. Of these shared paths, in Figure 8, all three journeys save on their shared path cost over their single shortest path cost. However, in Figure 9, the journey from S to P and the journey from B to O save more on their shared path cost here than in the shared path in Figure 8. On the other hand, in Figure 9, the journey from E to M saves nothing over its single shortest path. Therefore, Figure 8 provides the most journeys savings, and Figure 9 provides greater savings to fewer journeys.

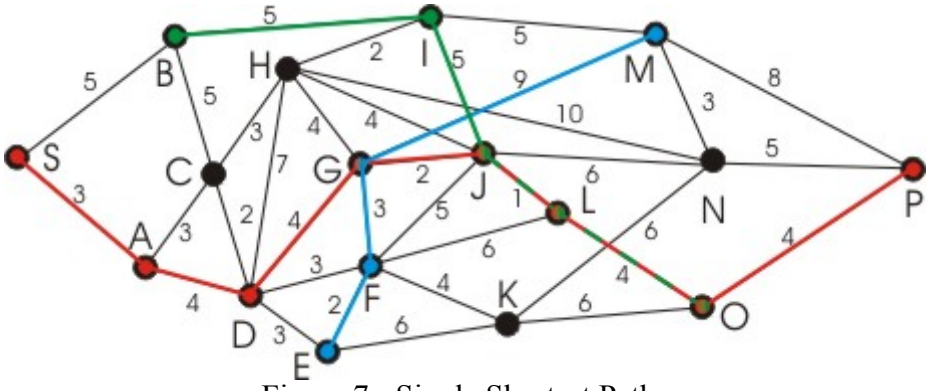

Figure 7 - Single Shortest Paths

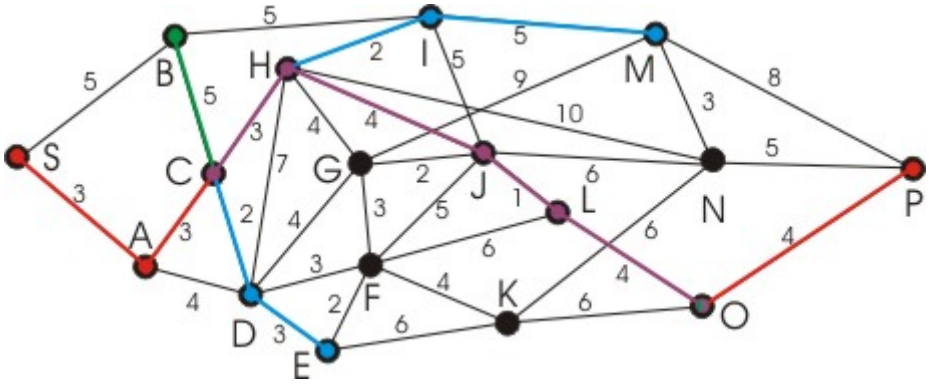

Figure 8 - A Shared Path

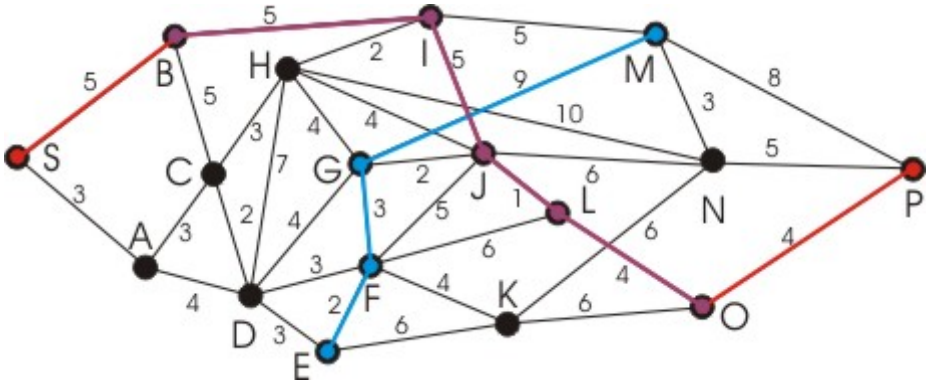

Figure 9 - A Shared Path

This research opens up opportunities for further study into and development of shared shortest paths. We can incorporate our optimal solution for two journeys into potentially better heuristics that add two journeys at a time for problems that use more than two journeys.

[1] Bird, C.G. On Cost Allocation for a Spanning Tree: A Game Theoretic Approach. Networks. 6, 335-350
[2] Claus, A and Kleitman, D.J. Cost Allocation for a Spanning Tree. Networks. 3, 289-304
[3] Courant, R. and Robbins, H. What is Mathematics? Oxford University Press, New York, 1941.
[4] Dijkstra, E.W. A Note on Two Problems in Connexion with Graphs. Numerische Mathematik. 1, S. 269–271
[5] Floyd, Robert W. Algorithm 97: Shortest Path. Communications of the ACM. 5:6. 345