An Exact Biobjective Shortest Path Method with Gateway Heuristic and Supported Point Upper-Bounds

F. Antonio Medrano and Richard L. Church

Project 301CR, GeoTrans Report 2013-06-01
June 2013

Photos courtesy of DOE/NREL

University of California, Santa Barbara
Department of Geography
1832 Ellison Hall
Santa Barbara, CA 93106
medrano@geog.ucsb.edu
church@geog.ucsb.edu

An Exact Biobjective Shortest Path Method with Gateway Heuristic and Supported Point Upper-Bounds

F. Antonio Medrano
Richard L. Church

Corridor Location Project
GeoTrans Laboratory
Department of Geography
University of California, Santa Barbara
June 30, 2013

This report has been developed as a part of the corridor location research project at the University of California, Santa Barbara. The goal of this project is to take a fresh look at the process of corridor location, and develop a set of algorithms that compute path alternatives using a foundation of solid geographical theory in order to offer designers better tools for developing quality alternatives that consider the entire spectrum of viable solutions. And just as importantly, as data sets become increasingly massive and present challenging computational elements, it is important that algorithms be efficient and able to take advantage of parallel computing resources. Please cite this report as: Medrano, FA, and RL Church (2013) "An Exact Biobjective Shortest Path Method with Gateway Heuristic and Supported Point Upper-Bounds" (Report #06-13-01), GeoTrans Laboratory, UCSB, Santa Barbara CA.

I. Introduction

Heuristic approaches are often implemented to define lower and/or upper bounds to the optimal solutions of difficult-to-solve problems (Danna *et al.* 2005, Hewitt *et al.* 2010). These bounds then restrict the solution space over which a solver computes, and can often speed up the search for the exact optimal solution. In Medrano and Church (2014), we introduced a fast heuristic that approximates the Pareto frontier of a biobjective shortest path problem. This approach was shown to efficiently calculate a set of paths that closely approximate the exact Pareto optimal set when evaluated in objective space. This approximation approach represents an excellent method to generate a candidate set from which to select viable alternatives, particularly in a real-world design scenario where there exists inherent uncertainly in the spatial data, and where data sets are large and present formidable computation issues. It is logical to ask the question of whether this approximation approach could be harnessed to speed up the computation of an exact Pareto optimal solution set to a biobjective shortest path problem. This report introduces a new exact algorithm for solving a biobjective shortest path problem based on an enumerative approach first introduced by Raith and Ehrgott (2009), but using the approximation heuristic as an improved upper bound to speed up overall computation times. In addition, valid bounds are introduced to avoid enumerating solutions already dominated by supported solutions. This further limits the enumeration search region in hopes of improving the performance of enumerative schemes.

Specialized exact algorithms for the biobjective shortest path problem fall under two categories: labeling approaches and enumeration approaches. Raith and Ehrgott (2009) reviewed the state-of-the-art approaches of the time, and introduced two-phase variants of a label-setting algorithm by Guerriero and Musmanno (2001) and a label-correcting approach by Skriver and Andersen (2000). They also introduced a new two-phase enumerative technique called 2NSP that used the Near Shortest Path (NSP) enumeration algorithm developed by Carlyle and Wood (2005). They then evaluated the performance of those methods on three types of networks: 1) regular grid networks with random discrete uniform distribution costs, 2) random networks generated by the NetMaker network generation technique described by Skriver and Andersen (2000), and 3) road networks acquired from the Tiger Road Networks for the 9th DIMACS implementation challenge, which were originally sourced from U.S. Census data. After developing our improvements to 2NSP, we replicated the experiments on these same networks with one exception in order to compare the performance of the new approach to the existing published methods. In lieu of the random-cost grid networks, we chose to test on spatial terrain-based networks of the kind that would be used in a transmission corridor location.

II. Near Shortest Path (NSP) Biobjective Shortest Path (BSP) Algorithm

Raith and Ehrgott (2009) introduced a two-phase enumerative biobjective shortest path (BSP) algorithm based upon the Near Shortest Path (NSP) enumeration technique developed by Carlyle and Wood (2005). Earlier enumerative BSP algorithms had used *k*-shortest path subroutines (Coutinho-Rodrigues *et al.* 1999), but Carlyle and Wood showed in their paper that NSP was a much faster approach to path enumeration due to not being restricted to find the paths in ascending path-length order. The NSP algorithm searches for all paths that are within some threshold $D = (1 + \varepsilon) \times L_{sp}$, where D is the maximum length path to be returned, ε is a positive real number, and L_{sp} is the length of the shortest path. For example, if $\varepsilon = 0.05$, then the NSP will output all paths that are within 5% of the length of the shortest path.

The Raith and Ehrgott NSP-BSP algorithm is a two-phase method, which is denoted here as 2NSP[1]. In the first phase, the supported solutions are found by solving weighted composite single-objective shortest path problems (Cohon *et al.* 1979). The remaining solutions are found in the Boundary of Unsupported Solution Search (BUSS) regions, sometimes referred to in the literature as the *duality gap*. These BUSS regions initially are defined only by the supported solutions, and consist of triangular areas between supported non-dominated (Pareto) solutions where unsupported non-dominated solutions may exist.

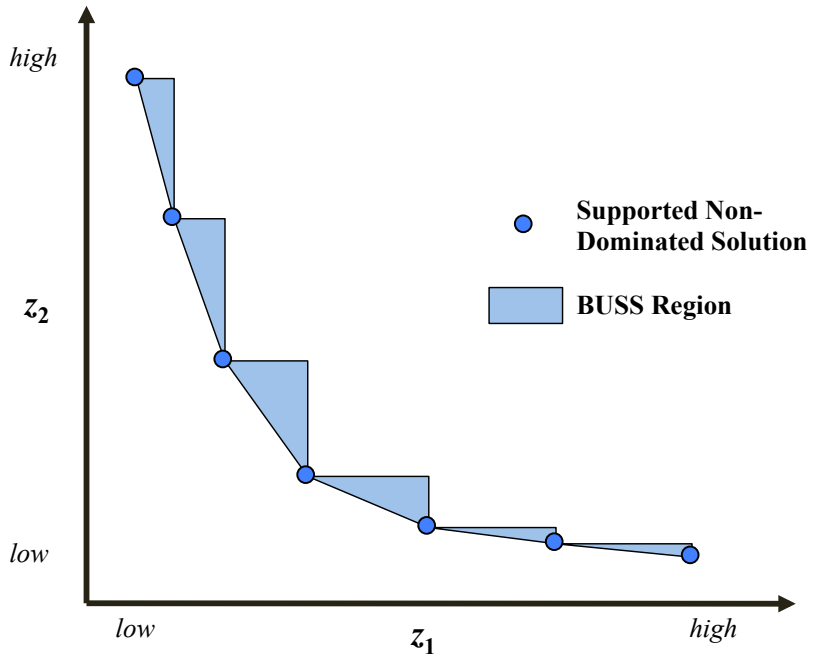

Figure 1. Initial BUSS regions between the supported non-dominated solutions

[1] Raith and Ehrgott call it NSPD, where the D means it uses Dijkstra's algorithm for phase-1

The second phase searches for the unsupported non-dominated solutions within each individual BUSS region. For each BUSS region, the composite weighting for the graph is set so that the two supported solutions that define the specific BUSS region have equal composite objective value (points A and B in Figure 2). This defines a projection axis from which composite objective values are measured in a direction perpendicular from this projection line. Two solutions with different *component* but equal *composite* objective values for the given weight will lie on a line parallel to the projection axis.

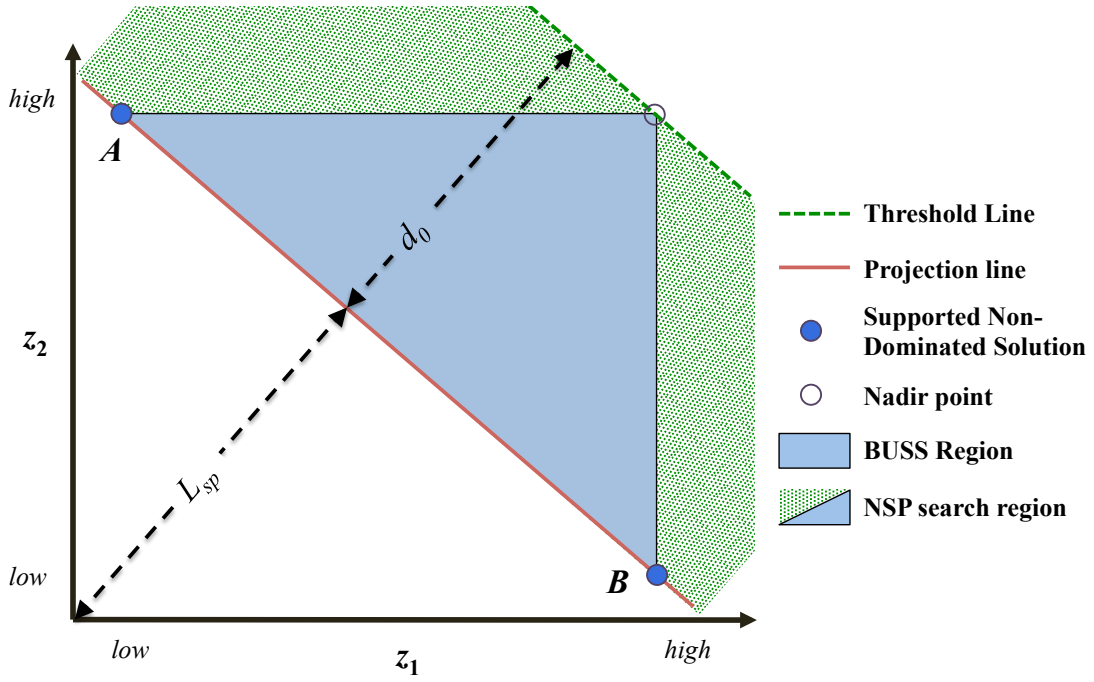

Figure 2. Setting the initial threshold for a 2NSP BUSS region. BUSS region is a subset of the NSP search region.

Recall that the NSP method enumerates all solutions within a threshold $D = (1 + \varepsilon) \times L_{sp}$. If we define $d = \varepsilon \times L_{sp}$, where d represents the amount of length greater than the shortest path length to reach the total threshold D, then this can be rewritten as $D = L_{sp} + d$. The 2NSP algorithm evaluates each BUSS region independently, and defines the initial threshold $D_0 = L_{sp} + d_0$, where d_0 is the perpendicular distance from the projection line to the most distant nadir point of the BUSS region (see Figure 2). This nadir point is initially located at the right-angle vertex of the BUSS right triangle. In objective space, this threshold is a line parallel to the projection axis that intersects the nadir point, and defines a search region that is an infinite strip between the projection line and the threshold line, and contains the entire BUSS region. Despite an infinite search area, feasible solutions will only exist in a bounded polygon sub-region due to the convexity of the supported solutions, which represent a lower bound on the feasible solution set.

The 2NSP method begins by enumerating paths with a composite objective $z_c < D_0$. Many of the output paths may fall outside of the BUSS region, but a path found that has z_1 and

z_2 objective values that place it inside the BUSS region will in-turn dominate some portion of that region, at which point the shape of the BUSS must be updated to reflect this. If the dominated region includes the nadir point that limited the value of threshold D, then the threshold is updated, further restricting the area to be searched and reducing computation to completion. This new threshold $D_{new} = L_{sp} + d_{new}$ is set to the most distant nadir point of the updated BUSS region, as referenced from the projection line (see Figure 3).

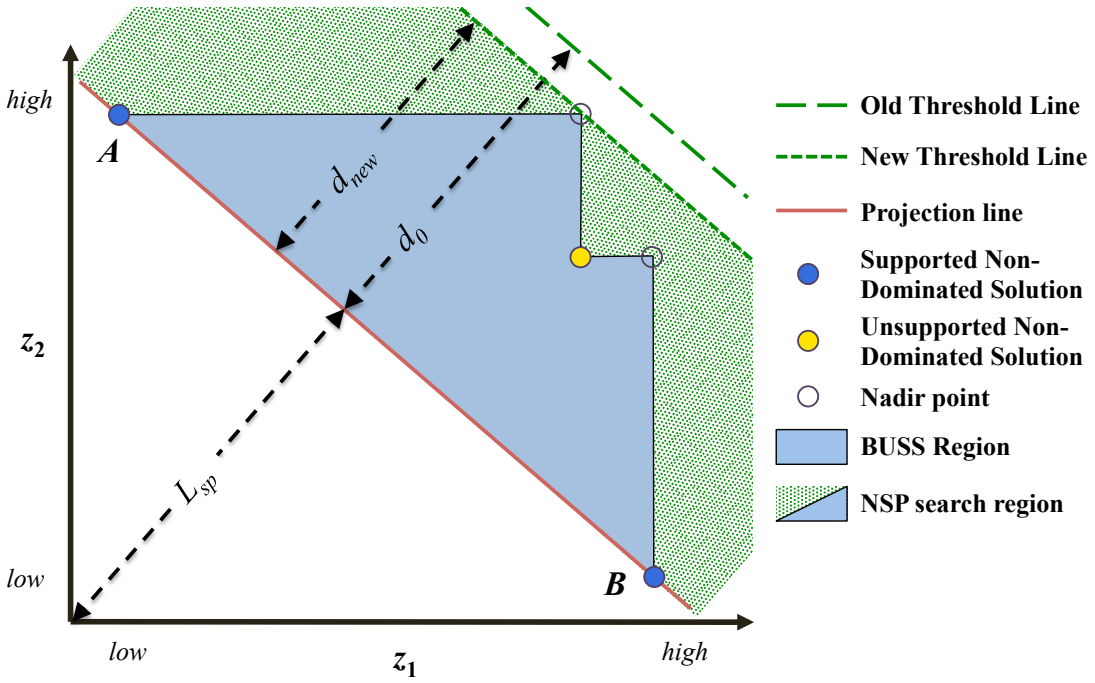

Figure 3. Updating the 2NSP threshold as solutions reduce the area of a BUSS region

Each new solution that falls into the updated BUSS region dominates some portion of the region, requiring an update to the set nadir points. A new solution may also dominate one or more of the previously found solutions, thus the program must check for this and remove any dominated solutions from the Pareto set. The NSP algorithm runs until all paths within the threshold have been enumerated, at which point all unsupported non-dominated solutions between the two supported solutions used to define the BUSS region have been found. The 2NSP method is then repeated for each adjacent pair of supported solutions until all BUSS regions have been analyzed and the entire Pareto set has been found.

III. Improvements to the NSP BSP Algorithm

A. Gateway Heuristic Upper Bound

In Medrano and Church (2014), a new heuristic method was developed that was capable of efficiently computing an effective approximation of the unsupported Pareto solution set to a biobjective shortest path problem. Since this method offers excellent solution sets with little computational effort, it is proposed here to use it as an initial solution upper-bound for the biobjective NSP approach. Thus, it begins with heuristically solving for an approximation of the non-dominated points using either the biobjective gateway node (GWN) or gateway arc (GWA) heuristic. Doing so requires twice as many shortest path solver iterations for the supported points as the standard 2NSP algorithm. The heuristic solutions are then added to the candidate Pareto solution set and the NSP threshold D is updated according to the same rules as discussed in the prior page and in Figure 3. NSP path enumeration (or any combinatorial path enumeration for that matter) has a tendency to grow exponentially as the threshold value increases, thus a more tightly restricted threshold value will result in significant reductions in overall computational effort. Two versions of the 2NSP algorithm with an initial gateway upper bound were implemented: one using gateway node candidate paths (called GWN2NSP), and another using gateway arc candidate paths (called GWA2NSP). Gateway node paths generate n path candidates for each weighting, where n is the number of nodes in the network. Gateway arcs generate m path candidates for each weighting, where m is the number of arcs in the network. As path candidates are generated they are added to a lexicographically sorted list of non-dominated path candidates. When a new candidate is added to the list there must also be a check to see if the new candidate dominates any existing candidates in the list. Overall, this adds some to the computational overhead, first in doubling the number of shortest path solver iterations during the first phase of the algorithm (the NISE supported solution phase) in order to generate the gateway paths, and also with a binary search which is needed to identify if a path is non-dominated, and if so then inserting the path into the list and removing any paths dominated by the newly inserted path. If the enumerative second phase of NSP-based algorithm takes an exceptional amount of time to compute, then the tighter NSP bounds generated by the presence of an initial approximate non-dominated path set from the gateway heuristic has the potential to significantly reduce overall computation times.

B. Supported Solution Dominance Bounds

The single objective Carlyle and Wood (2005) NSP algorithm uses depth-first-search (DFS) to find all paths with length below a given threshold, D. The DFS uses a first-in last-out stack data structure to build paths by adding arcs to the stack so long as they could result in a path that meets that threshold criterion. A naïve approach would add arcs to the stack so long as the total path length is less than D, and not add an arc if adding it would cause the path length to exceed D. The original NSP algorithm developed by Byers and Waterman (1984) added an elegant innovation to this idea by using information from an initial reverse shortest path tree in order to improve the efficiency of the enumeration. As an example, suppose one is enumerating near shortest paths from S to T. NSP uses a stack data structure to contain arcs that make up a path P from node S to

node x, with length $L(x)$. An arc (x,y) from node x to node y will have a cost $c(x,y)$, and the shortest path from a node x to the destination node T (as computed initially by the reverse shortest path tree) has cost $d'(x)$. Let us consider a node $u \neq T$. Some path P of length $L(u)$ led us from node S to node u having a path length $\leq D$. Let us now consider adding arc (u,v) to the path P. The arc is added to the path if the following is true.

$$L(u) + c(u,v) + d'(v) < D \quad (1)$$

This means that arc (u, v) is added only if the sum of the existing path length to u, plus the cost of the arc from u to v, plus the shortest possible path length from v to the destination, is less than the threshold D. Using the $d'(v)$ information allows the algorithm to know if there does not exist any feasible possibility of reaching the destination without exceeding the limit D. If so, it is no longer necessary then to search that portion of the graph. This approach may be considered similar to the approximation of the shortest path length from a candidate node to the destination used in the A* algorithm developed by Hart *et al.* (1968).

The biobjective variant of the algorithm, 2NSP, is the same except it uses a composite weighted single objective function z_c that is a function of the individual competing objectives z_1 and z_2, weighted by α, where $z_c = \alpha \times z_1 + (1 - \alpha) \times z_2$, and α is determined by the slope of the supported points that define the particular BUSS region being analyzed. The 2NSP method for the bi-objective case at each iteration checks if the composite function meets the threshold criterion.

$$L_c(u) + c_c(u,v) + d'_c(v) < D_c \quad (2)$$

This results in paths enumerated that all fall within the given BUSS region used to define the composite weighting. But this boundary also encompasses areas outside of the BUSS region, resulting in numerous paths enumerated that are dominated by the previously found supported non-dominated solutions (see Figure 4).

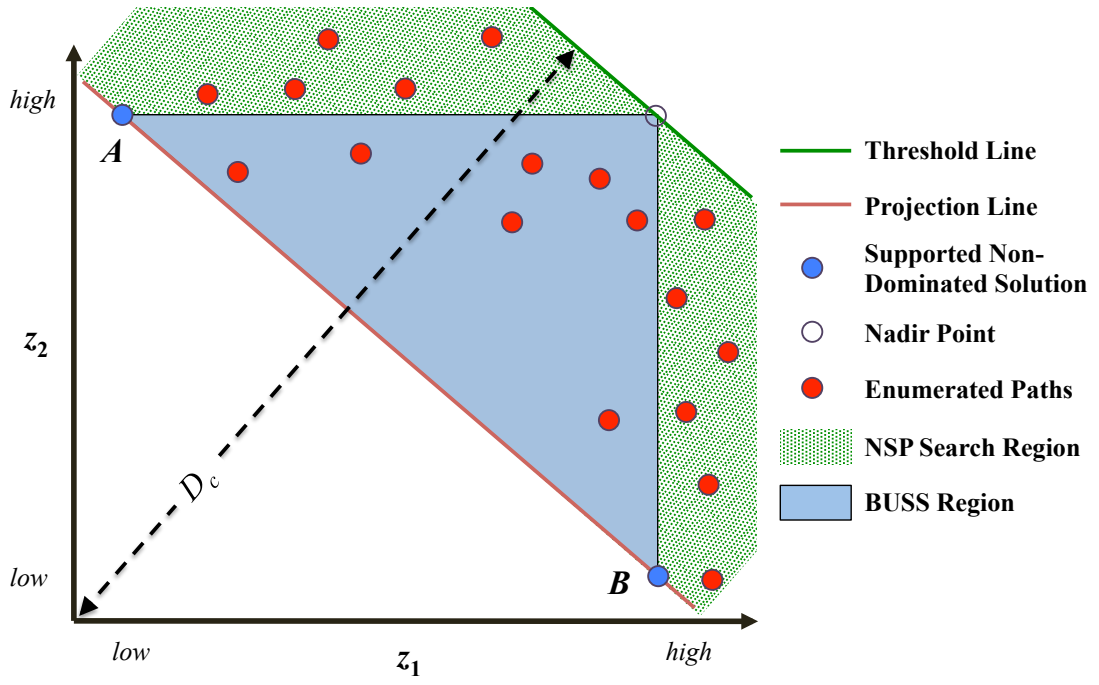

Figure 4. Paths enumerated by 2NSP, both inside and outside the BUSS region

The paths found that are outside of the BUSS region constitute wasted computational effort, as they are always dominated by a supported solution. Our tests on various raster networks have shown that a minimum of 75%, and as many as 99.88% of the paths enumerated by 2NSP were located in objective space outside of the BUSS regions.

Given this, it is logical to add a check that eliminates these paths from being found, by not only testing if a proposed arc added to the stack would exceed the threshold D_c, but also if it would result in a path dominated by either of the supported points. This requires storing two additional labels during the initial NISE supported solution computation. Initially, biobjective NISE solves for α weightings of 0 and 1, corresponding to independent single-objective solutions that ignore the existence of other objectives. Storing the reverse shortest path tree distance for these weightings for each node gives labels for the best possible future outcome with respect to each independent objective. We call these labels $d'_1(v)$ and $d'_2(v)$ for the z_1 and z_2 objectives, respectively. Then, at the point where 2NSP queries an arc to determine if it should be added to the stack (equation 15), we add the following additional queries, where A and B are the supported points of the BUSS region.

$$L_1(u) + c_1(u,v) + d'_1(v) < z_1(B) \quad (3)$$

$$L_2(u) + c_2(u,v) + d'_2(v) < z_2(A) \quad (4)$$

This puts additional bounds on the NSP computation, dramatically reducing the size of the NSP search region to only that of the BUSS region (see Figure 5), and reducing the total number of paths enumerated. We denote this bounded algorithm as B2NSP.

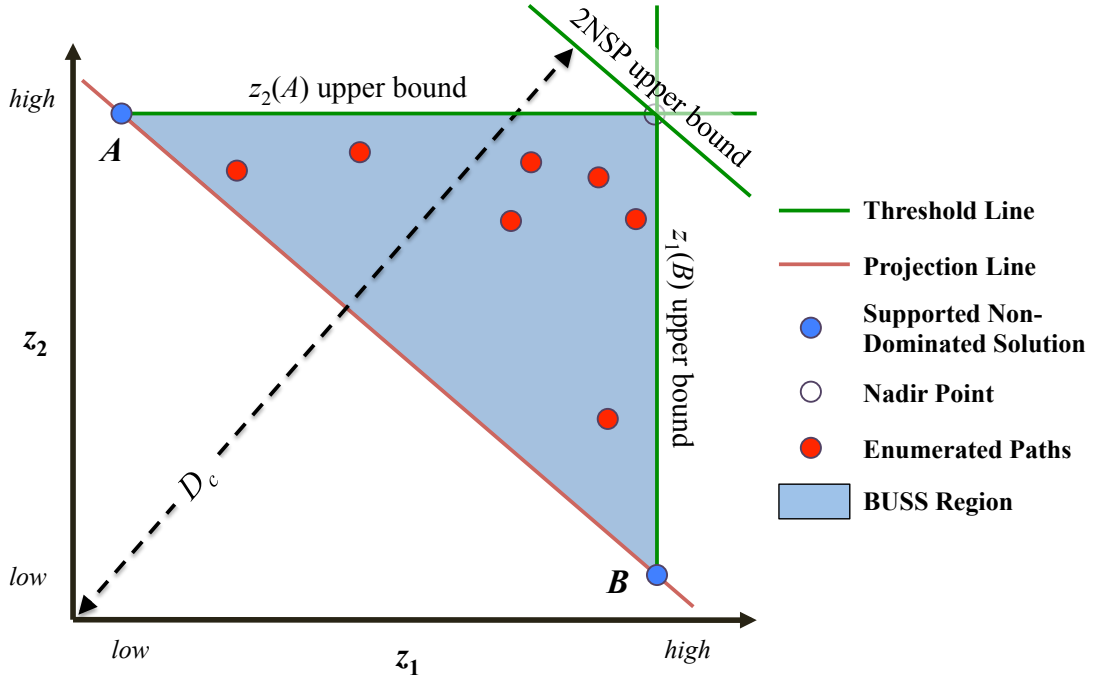

Figure 5. B2NSP supported solution bounds restricting enumerated paths to only within the initial BUSS region

C. Combining Both Improvements

The two improvement approaches described above are not mutually exclusive, and can be combined to further improve the efficiency of the enumerative method. We added the combined algorithms to our experiments, and denote the supported bounds gateway node variant as BGWN2NSP, and the supported bounds gateway arc variant as BGWA2NSP.

IV. Numerical Experiments

A. Data Sets

Similar to Raith and Ehrgott (2009), the algorithms were tested on three types of networks: grid, random, and road networks. The random and road networks were the same as those used by Raith and Ehrgott, while we have changed the grid networks to be terrain-based GIS raster networks representative of the type that would be used for a transmission line corridor location application. Details on each network type are described below.

1. Raster Networks

The first type of network Raith and Ehrgott (2009) used to test their biobjective shortest path algorithms were orthogonal grid networks with random integer arc costs. Because the main focus of this research is on corridor location, it was decided to test the bi-objective shortest path approach on spatial terrain-based networks of the kind that would be used in a transmission location problem. While terrain networks are similar to grid networks in that they contain a regular connectivity pattern, they can be made more dense when they incorporate queen's move connectivity and/or knight's move connectivity between nodes (Goodchild 1977). Such connectivity also requires the use of non-integer arc costs to account for the geometry, thus the programs we implemented use floating point arithmetic rather than integer math.

The particular terrain networks used were 1) a 20x20 manually fabricated raster, 2) an 80x80 subset of the Maryland Automated Geographic Information (MAGI) system database, and 3) a 100x160 subset of the same MAGI database. The first two networks originally included just a single objective cost layer, so a second objective node cost, z_2, was randomly generated and scaled to match z_1 ranges. The 100x160 network contains two cost layers, with z_1 pertaining to an economic cost layer, and z_2 pertaining to an environmental impact layer.

Each raster was used to define networks of three r-radius types: 1) an orthogonal ($r = 0$) network, 2) a "queen's move" ($r = 1$) orthogonal and diagonal network, and 3) a "queen's plus knight's move" ($r = 2$) network. Table 1 contains pertinent statistics of these networks, including size, origin-destination node locations, r radius value, number of supported Pareto (non-dominated) solutions, and the total number of Pareto solutions.

Table 1. Raster Test Networks

Name	Dimensions and OD	r	Nodes	Arcs	Supported Pareto Solutions	All Pareto Solutions
R1	20x20	0	400	1,520	10	28
R2	SW to NE	1	400	2,964	11	49
R3		2	400	5,700	16	103
R4	80x80	0	6,400	25,280	29	120
R5	SW to NE	1	6,400	50,244	36	371
R6		2	6,400	99,540	57	1339
R7	100x160	0	16,000	63,480	11	22
R8	SW to NE	1	16,000	126,444	19	199
R9		2	16,000	251,340	35	718
R10	100x160	0	16,000	63,480	9	20
R11	NW to SE	1	16,000	126,444	17	109
R12		2	16,000	251,340	44	495

2. Random NetMaker Networks

When Skriver and Andersen (2000) published their label correcting biobjective shortest path algorithm, they tested their approach on random networks. Originally they had tested their algorithm on NETGEN networks in order to replicate earlier work by Huarng *et al.* (1996), but they found that the simple structure of the NETGEN networks resulted in data sets with very few non-dominated paths. They then developed a new random network generator that they called NetMaker that used random Hamiltonian cycles to generate random networks with more non-dominated paths. They felt that this structure would better approximate real life problems, although it is unclear why then they did not simply try using real world data. The two objective costs for NetMaker networks are randomly assigned integers with approximately half of the arcs having a high c_{ij}^1 value and a low c_{ij}^2 value, and the other half of the arcs have a low c_{ij}^1 and a high c_{ij}^2. Full details on the structure and methods used to generate the networks can be found in Skriver and Andersen (2000).

Raith and Ehrgott (2009) also used the NetMaker networks in order to compare the different algorithms. They coded their own version of NetMaker using the same rules developed by Skriver and Anderson, and included additional features to further modify the network properties. Raith (2010) noted in a later paper that an error in their NetMaker generator code created networks with shortcuts from the origin to the destination, resulting in experiments where the computations solved instantaneously no matter what size of network was used. Their networks had very few non-dominated solutions because of the shortcut paths dominating all other possible routes. We were able to acquire the original NetMaker generator program developed by Skriver and Anderson which did not include this error, and we used that code to generate our own errorless random networks for use in our computational experiments that are consistent with the previous literature.

The NetMaker generator program allows for certain input parameters when generating new networks. It first prompts for how many nodes in the network, and creates a random Hamiltonian cycle with all of the nodes. Then it prompts for parameters defining the maximum and minimum number of outgoing arcs that should emanate from each node.

Finally, the I_{node} parameter is the node interval, i.e. the maximum allowed range of any outgoing arc. Essentially, if at node 100, and the I_{node} = 20, then outgoing arcs can terminate at any node within the number range 90 to 110. Table 2 lists the NetMaker networks used in our experiments, their parameters, and their properties. Networks NM1 to NM20 match the parameters of the Raith and Ehrgott experiments, where they tested on networks of up to 21,000 nodes and approximately 530,000 arcs. This work extends the experiments to networks up to 15 times larger than the Raith and Ehrgott networks (NM21 to NM30), the largest with 315,000 nodes and 7,876,183 arcs. Interestingly, the number of non-dominated solutions did not increase with network size, and seemed to correlate more with the parameters for minimum and maximum outgoing arcs per node.

Table 2. NetMaker Test Networks

Name	I_{node}	Min Outgoing Arcs	Max Outgoing Arcs	Nodes	Arcs	Supported Pareto Solutions	All Pareto Solutions
NM1	20	5	15	3,000	30,088	11	25
NM2	20	1	20	3,000	31,590	4	9
NM3	50	5	15	3,000	30,046	9	22
NM4	50	1	20	3,000	31,746	8	22
NM5	50	10	40	3,000	74,942	7	31
NM6	20	5	15	7,000	70,057	10	38
NM7	20	1	20	7,000	73,676	5	11
NM8	50	5	15	7,000	69,483	7	10
NM9	50	1	20	7,000	73,162	8	19
NM10	50	10	40	7,000	174,208	15	37
NM11	20	5	15	14,000	139,989	13	41
NM12	20	1	20	14,000	147,288	6	18
NM13	50	5	15	14,000	140,094	8	24
NM14	50	1	20	14,000	145,755	10	33
NM15	50	10	40	14,000	350,600	8	32
NM16	20	5	15	21,000	209,770	13	33
NM17	20	1	20	21,000	223,245	9	39
NM18	50	5	15	21,000	209,575	10	24
NM19	50	1	20	21,000	220,495	10	32
NM20	50	10	40	21,000	526,615	11	38
NM21	20	20	5	28,000	279,658	6	26
NM22	50	50	10	28,000	699,870	14	36
NM23	20	20	5	70,000	700,845	9	28
NM24	50	50	10	70,000	1,747,489	12	43
NM25	20	20	5	100,000	1,002,682	10	42
NM26	50	50	10	100,000	2,500,462	10	47
NM27	20	20	5	210,000	2,101,252	11	38
NM28	50	50	10	210,000	5,247,056	15	40
NM29	20	20	5	315,000	3,147,674	10	30
NM30	50	50	10	315,000	7,876,183	11	43

3. Road Networks

The third type of network we used in the experiments reported here were road networks acquired from the Tiger Road Networks for the 9th DIMACS implementation challenge[2]. These networks were originally sourced from U.S. Census TIGER data, but then cleaned for the competition to contain only road data. The two costs for each arc are the travel time in seconds, and the travel distance in meters. The travel time was determined by DIMACS by multiplying the travel distance of an arc by one of four different road quality factors. One can see that these two objectives are correlated with each other, rather than competing, so overall the Pareto frontiers have relatively few solutions for a given network size. Like Raith and Ehrgott (2009), this experiment used the Washington DC (DC), Rhode Island (RI), and New Jersey (NJ) networks. They did not list which nodes were used for their origin and destinations, so instead OD pairs were randomly selected. Table 3 lists the road networks used and their properties. It should be noted that Raith and Ehgott used versions with an artificially high-cost Hamiltonian cycle added to the network to ensure connectivity. Instead, the base network without the added cycle was used, and different OD pairs were selected if a pair turned out to be not connected.

Table 3. Road Test Networks

Name	State/ District	Nodes	Arcs	Average Pareto Solutions	Minimum Pareto Solutions	Maximum Pareto Solutions
DC1–11	Washington, DC	9,559	29,818	22.6	3	76
RI1–11	Rhode Island	53,658	138,426	20.2	2	92
NJ1–11	New Jersey	330,386	872,072	82.9	12	221

B. Computational Results

All algorithms were coded in Java 7 using the Processing library (www.processing.org) for graphical support in visualizing networks and results. The Processing library does slightly slow down code runtimes, but since it was used for all codes the relative performance between algorithms is consistent. All computational tests were performed on an Apple laptop with an Intel Core i7-3820QM, 2.7 GHz quad-core processor, 16GB of RAM, running OS X 10.9. Whenever the computation exceeded 10 hours, computation was stopped. Such instances are indicated by a dash in the table of results.

1. Raster Networks

thwhile on this type of network.

Table 4 and Table 5 include the computational runtimes and the total number of paths enumerated for the labeling and enumeration algorithms on the raster test networks. Figure 6 displays the runtime results in a graphical chart. The first observation is the exponential runtime behavior of the enumeration methods on the raster terrain networks as the networks grew in size. Runtimes for all of the methods were all less than 0.1 seconds on all of the 20x20 networks. On R4-R6, the gateway and supported solution

[2] http://www.dis.uniroma1.it/challenge9/data/tiger/

upper bounds both made significant improvements over 2NSP, with the best results coming from combining the two. On R4, the BGWN2NSP and BGWA2NSP methods sped up computation over 2NSP by greater than 843 times by requiring less than 0.07% of the number of paths to be enumerated. Tests on R5 showed similar significant speedup, and experiments on R6 made the problem solvable using BGWA2NSP and BGWN2NSP in 1.5 hours, while 2NSP and B2NSP were stopped incomplete after 10 hours. Despite the improvements though, all problems failed to solve after 10 hours for all enumerative methods on R7-R12, while both labeling methods (LCor and 2LCor) solved these problems in less than a minute. The GWA bounded algorithms typically performed equal to or better than their corresponding GWN versions, making the additional overhead of generating a GWA candidate set worthwhile on this type of network.

Table 4. Algorithm runtimes on raster networks

	LCor	2LCor	2NSP	B2NSP	GWN2NSP	GWA2NSP	BGWN2NSP	BGWA2NSP
	time (sec)	time (sec)	time (sec)	time (sec)	time (sec)	time (sec)	time (sec)	time (sec)
R1	0.008	0.018	0.021	0.019	0.024	0.023	0.020	0.022
R2	0.018	0.031	0.028	0.025	0.024	0.027	0.021	0.027
R3	0.066	0.056	0.057	0.044	0.070	0.070	0.046	0.059
R4	0.190	0.430	7870.111	25.453	71.173	71.023	9.280	9.327
R5	1.396	1.490	59.373	35.420	12.033	12.135	7.491	7.534
R6	38.779	22.039	–	–	10073.280	9786.610	5706.956	5585.135
R7	0.132	0.388	–	–	–	–	–	–
R8	3.124	1.891	–	–	–	–	–	–
R9	50.817	25.323	–	–	–	–	–	–
R10	0.120	0.358	–	–	–	–	–	–
R11	0.415	0.941	–	–	–	–	–	–
R12	3.099	4.867	–	–	–	–	–	–

Note: results indicated with a dash did not solve in the imposed time limit of 10 hours

Table 5. Number of enumerated paths on raster networks

	LCor	2LCor	2NSP	B2NSP	GWN2NSP	GWA2NSP	BGWN2NSP	BGWA2NSP
	paths	paths	paths	paths	paths	paths	paths	paths
R1	n/a	n/a	2,213	260	814	814	145	145
R2	n/a	n/a	2,136	328	469	465	115	114
R3	n/a	n/a	11,983	2,816	6,397	5,314	1,754	1,549
R4	n/a	n/a	3,569,495,619	4,362,303	27,524,313	27,517,643	2,483,849	2,482,336
R5	n/a	n/a	37,788,197	9,418,048	6,694,418	6,694,081	1,792,104	1,792,051
R6	n/a	n/a	–	–	6,287,222,438	6,246,979,125	2,102,200,145	2,098,403,081
R7	n/a	n/a	–	–	–	–	–	–
R8	n/a	n/a	–	–	–	–	–	–
R9	n/a	n/a	–	–	–	–	–	–
R10	n/a	n/a	–	–	–	–	–	–
R11	n/a	n/a	–	–	–	–	–	–
R12	n/a	n/a	–	–	–	–	–	–

Note: results indicated with a dash did not solve in the imposed time limit of 10 hours

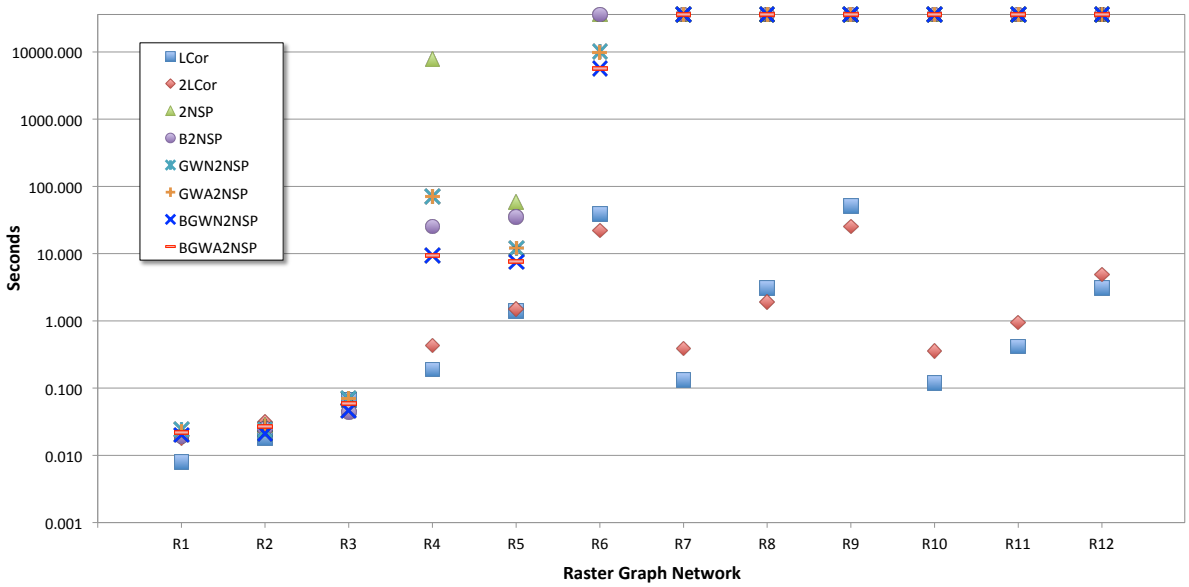

Figure 6. Algorithm runtimes on raster networks

Grid and raster networks were shown to be inherently challenging for enumerative approaches. The regular arc pattern and relatively small number of distinct values of arc costs means that as the threshold D increases linearly, the number of path combinations increase exponentially. This is exemplified by the massive increase in paths enumerated when comparing the 80x80 enumeration results to the 20x20 results, and looking at the intractability of the NSP methods on the 100x160 grids. The node labeling methods were not as encumbered as network sizes increased, since their runtimes depended more on simply the number of nodes in the network and proved to be much more efficient in these cases. Overall, for the terrain based raster networks 2LCor appears to be the best algorithm, with LCor being the second best.

2. Random NetMaker Networks

Table 6 and Table 7 include the computational runtimes and the total number of paths enumerated for the labeling and enumeration algorithms on the NetMaker test networks. Figure 7 displays the runtime results in a graphical chart. Earlier we observed that the number of non-dominated solutions on NetMaker networks depend more on the generation parameters rather than the size of the network, and the number of paths enumerated did not seem to vary much by network size. In fact, all of the enumeration techniques generated very few paths to find the complete Pareto set, so the gateway and supported solution upper bounds did not yield much improvement to the overall computation time. The overhead required in the gateway approaches, where the number of shortest path computations is doubled in order to generate a gateway path set, always overshadowed the benefits of reduced enumeration effort. Similarly, for 2LCor, the overhead of computing the supported points first did not yield faster computation times over LCor on these networks.

Table 6. Algorithm runtimes on NetMaker networks

	LCor	2LCor	2NSP	B2NSP	GWN2NSP	GWA2NSP	BGWN2NSP	BGWA2NSP
	time (sec)	time (sec)	time (sec)	time (sec)	time (sec)	time (sec)	time (sec)	time (sec)
NM1	0.281	0.418	0.149	0.139	0.230	0.251	0.231	0.252
NM2	0.248	0.277	0.050	0.050	0.149	0.150	0.154	0.176
NM3	0.243	0.359	0.129	0.130	0.217	0.233	0.222	0.232
NM4	0.256	0.335	0.100	0.100	0.202	0.226	0.213	0.225
NM5	0.559	0.614	0.184	0.183	0.294	0.347	0.300	0.401
NM6	0.538	0.735	0.268	0.284	0.494	0.618	0.510	0.559
NM7	0.525	0.612	0.139	0.148	0.304	0.340	0.313	0.349
NM8	0.489	0.513	0.068	0.073	0.378	0.422	0.388	0.421
NM9	0.496	0.634	0.222	0.222	0.430	0.547	0.442	0.557
NM10	1.411	1.607	0.550	0.542	1.140	1.374	1.170	1.374
NM11	1.049	1.615	0.737	0.724	1.514	1.676	1.516	1.648
NM12	1.138	1.286	0.268	0.271	0.827	0.892	0.843	0.900
NM13	0.863	1.146	0.488	0.488	1.066	1.150	1.087	1.159
NM14	0.952	1.343	0.563	0.537	1.247	1.360	1.252	1.369
NM15	2.309	2.445	0.660	0.646	1.761	1.937	1.795	1.943
NM16	1.446	2.446	1.254	1.275	2.649	2.826	2.725	2.868
NM17	1.598	2.113	0.865	0.875	1.904	2.091	1.986	2.146
NM18	1.274	1.992	1.069	1.028	2.148	2.228	2.158	2.240
NM19	1.487	1.952	0.939	0.911	2.216	2.414	2.253	2.434
NM20	4.045	4.692	1.575	1.577	4.093	4.519	4.064	4.561
NM21	2.221	2.548	0.828	0.819	1.912	2.083	1.934	2.104
NM22	5.730	7.159	2.953	3.061	8.066	8.853	8.207	8.808
NM23	6.545	8.915	3.569	3.548	12.105	12.900	12.372	12.944
NM24	20.173	27.049	15.825	16.025	32.980	34.841	33.816	34.956
NM25	9.952	20.043	11.344	11.042	27.094	28.302	26.951	28.257
NM26	32.009	45.246	22.671	22.136	52.077	55.031	52.063	55.087
NM27	24.385	66.291	41.079	41.257	125.815	129.540	126.357	129.552
NM28	68.948	197.415	154.987	153.856	352.190	357.076	351.135	357.642
NM29	47.199	107.098	56.539	56.603	186.031	190.220	186.075	190.510
NM30	109.945	343.840	228.322	228.316	548.658	557.424	549.109	556.172

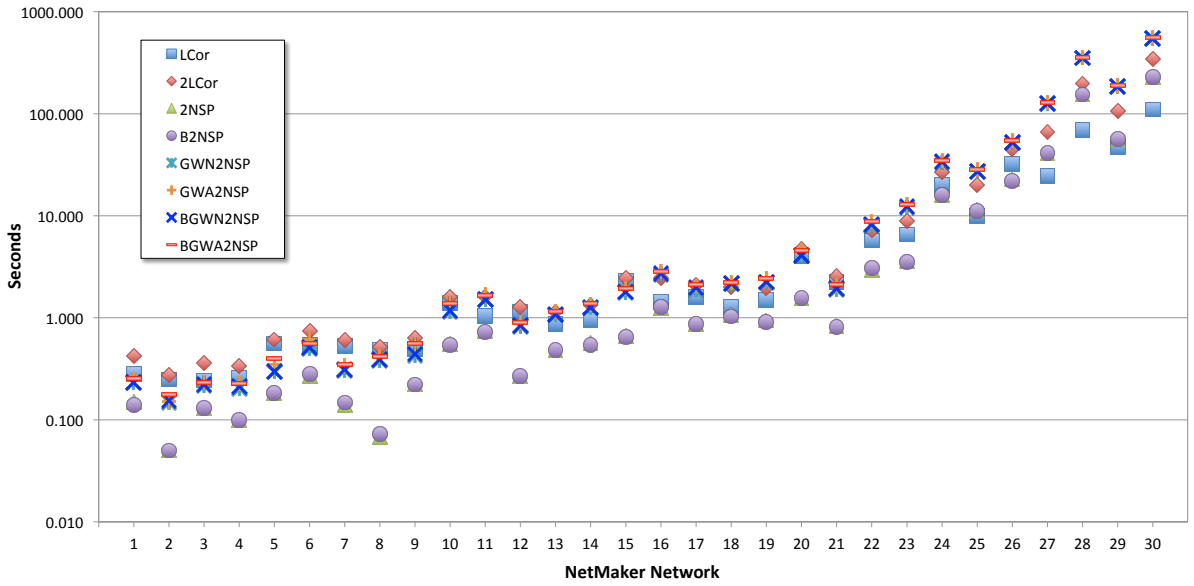

Figure 7. Algorithm runtimes on NetMaker networks

Table 7. Number of enumerated paths on NetMaker networks

	LCor	2LCor	2NSP	B2NSP	GWN2NSP	GWA2NSP	BGWN2NSP	BGWA2NSP
	paths	paths	paths	paths	paths	paths	paths	paths
NM1	n/a	n/a	287	53	187	187	44	44
NM2	n/a	n/a	68	7	46	45	7	7
NM3	n/a	n/a	192	45	151	151	39	39
NM4	n/a	n/a	275	54	158	157	46	46
NM5	n/a	n/a	1,174	423	520	491	250	247
NM6	n/a	n/a	592	98	381	324	72	63
NM7	n/a	n/a	140	68	64	47	37	23
NM8	n/a	n/a	119	16	88	88	15	15
NM9	n/a	n/a	214	48	184	184	45	45
NM10	n/a	n/a	545	64	461	410	50	44
NM11	n/a	n/a	615	138	380	343	79	76
NM12	n/a	n/a	182	52	124	123	33	32
NM13	n/a	n/a	165	41	124	124	28	28
NM14	n/a	n/a	345	78	145	145	47	47
NM15	n/a	n/a	857	314	212	177	110	90
NM16	n/a	n/a	443	68	313	272	57	53
NM17	n/a	n/a	396	67	178	169	48	48
NM18	n/a	n/a	348	77	194	189	39	39
NM19	n/a	n/a	232	38	179	179	34	34
NM20	n/a	n/a	509	86	265	234	59	54
NM21	n/a	n/a	764	201	283	283	92	92
NM22	n/a	n/a	1,591	106	958	958	88	88
NM23	n/a	n/a	823	85	400	334	60	54
NM24	n/a	n/a	1,738	377	729	619	245	216
NM25	n/a	n/a	1,476	271	649	617	125	123
NM26	n/a	n/a	1,186	287	368	358	122	117
NM27	n/a	n/a	457	82	301	277	64	62
NM28	n/a	n/a	1464	222	579	556	121	120
NM29	n/a	n/a	673	182	228	189	54	45
NM30	n/a	n/a	1334	315	313	294	116	105

On the NM1 to NM26 networks, the 2NSP and B2NSP algorithms performed equally well as the best approaches. For the largest four networks, LCor took over as the best algorithm. Since Netmaker networks do not emulate any sort of real-world network problem, we did not explore if this trend would continue for even larger networks, or for different parameters in generating the networks. If an interesting application with this network structure is found, then it may be beneficial to further explore a wider variety of parameters and their effects on computation times on the various methods.

3. Road Networks

Table 8 and Table 9 include the computational runtimes and the total number of paths enumerated for the labeling and enumeration algorithms on the road test networks. Figure 8 displays the runtime results in a graphical chart. For the small DC road network, B2NSP was the fastest for 8 out of the 11 problems, while 2LCor was fastest for the other 3. All algorithms converged on all problems except for 2NSP, which timed out after 10 hours on one instance. For the medium RI road networks, B2NSP was the fastest for 9 out of the 11 problems, while 2LCor was fastest for the other 2. All enumeration techniques timed out on RI1; and 2NSP, GWN2NSP, and GWA2NSP timed out on RI2.

For the largest NJ road networks, 2LCor was fastest for 5 problems, BGWN2NSP was fastest for 4 problems, and B2NSP was the fastest for 2 problems. On the problems where 2LCor was fastest, almost all of the enumeration techniques failed to converge in less than 10 hours. From this data we can observe that the NSP improvements, in particular the supported solution dominance bounds, provided significant gains in speed to the enumeration techniques, often resulting in the fastest computation times. But instances where the NSP methods could not solve a problem in a reasonable amount of time indicate the instability and exponential nature of enumeration for these types of road networks. On the other hand, labeling techniques were often not the fastest, but they always solved without fail in less than the cutoff time, with 2LCor always performing better than LCor. This dependability must be taken into account when selecting a best approach, so that even though B2NSP was fastest in 19 out of the 33 problems, we must recommend the 2LCor as the best method for road networks. If one can solve with two methods in parallel, then we recommend both 2LCor and B2NSP be used, since B2NSP performed best in most cases, but failed in others.

Table 8. Algorithm runtimes on road networks

	LCor	2LCor	2NSP	B2NSP	GWN2NSP	GWA2NSP	BGWN2NSP	BGWA2NSP
	time (sec)	time (sec)	time (sec)	time (sec)	time (sec)	time (sec)	time (sec)	time (sec)
DC1	0.865	0.604	3.261	0.804	0.707	0.724	0.609	0.651
DC2	0.820	0.532	–	11.415	22.940	23.015	1.682	1.695
DC3	0.638	0.469	0.538	0.324	0.347	0.380	0.351	0.369
DC4	0.724	0.468	0.140	0.085	0.213	0.227	0.308	0.298
DC5	0.522	0.194	7.366	0.278	1.127	0.844	0.381	0.328
DC6	0.420	0.166	0.050	0.045	0.069	0.082	0.175	0.182
DC7	0.382	0.132	0.117	0.103	0.167	0.183	0.277	0.226
DC8	0.479	0.218	0.134	0.102	0.157	0.178	0.155	0.170
DC9	0.817	0.345	0.098	0.089	0.105	0.129	0.252	0.251
DC10	0.374	0.212	0.074	0.072	0.177	0.203	0.254	0.255
DC11	0.404	0.192	0.196	0.133	0.194	0.216	0.301	0.251
RI1	22.022	6.322	–	–	–	–	–	–
RI2	15.588	5.825	–	848.618	–	–	251.176	252.675
RI3	17.570	1.675	0.111	0.095	0.295	0.311	0.358	0.358
RI4	8.588	0.967	2.058	0.346	1.406	1.452	0.673	0.722
RI5	5.397	1.754	0.878	0.426	1.016	1.058	0.816	0.852
RI6	40.880	5.749	0.309	0.159	0.849	0.852	0.803	0.823
RI7	16.093	4.407	0.227	0.135	0.663	0.659	0.577	0.645
RI8	11.152	3.976	0.260	0.268	0.636	0.650	0.623	0.660
RI9	14.077	2.350	0.168	0.162	0.702	0.731	0.647	0.696
RI10	7.067	1.873	0.082	0.077	0.367	0.382	0.403	0.410
RI11	6.027	1.720	0.310	0.260	0.735	0.755	0.701	0.716
NJ1	922.217	214.033	–	–	–	–	–	–
NJ2	697.355	112.949	–	–	–	–	4503.596	4506.357
NJ3	301.700	107.596	–	–	–	–	–	–
NJ4	514.996	46.948	0.927	0.498	5.475	5.416	4.989	5.164
NJ5	356.477	127.828	100.460	59.99	29.775	29.256	19.264	20.121
NJ6	492.078	133.286	–	–	–	–	–	–
NJ7	579.252	47.396	–	45.849	2245.270	2252.835	24.421	25.142
NJ8	466.655	364.094	–	–	–	–	–	–
NJ9	1149.296	85.207	82.830	10.294	17.733	18.219	8.649	9.035
NJ10	527.648	171.456	139.124	60.982	99.417	99.255	44.161	45.296
NJ11	925.478	17.286	0.405	0.294	3.663	3.843	3.444	3.677

Note: results indicated with a dash did not solve in the imposed time limit of 10 hours

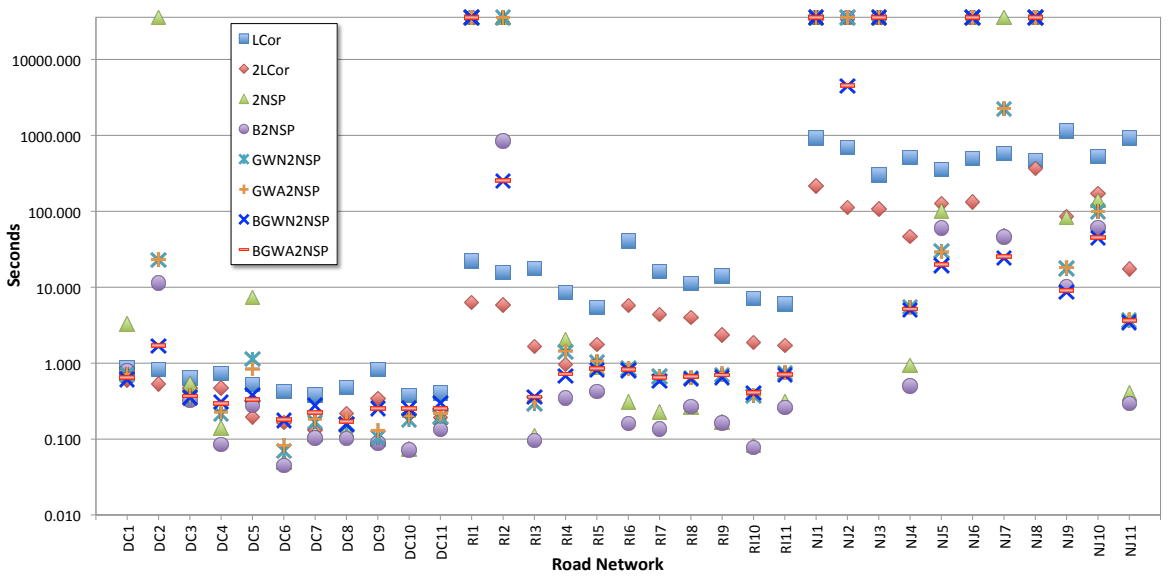

Figure 8. Algorithm runtimes on road networks

Table 9. Number of enumerated paths on road networks

	LCor	2LCor	2NSP	B2NSP	GWN2NSP	GWA2NSP	BGWN2NSP	BGWA2NSP
	paths	paths	paths	paths	paths	paths	paths	paths
DC1	n/a	n/a	1,121,078	103,548	332,083	332,067	39,253	39,249
DC2	n/a	n/a	–	596,264	8,532,377	8,532,377	89,152	89,152
DC3	n/a	n/a	72,904	2,237	18,886	18,885	1,333	1,332
DC4	n/a	n/a	16,454	137	11,493	11,492	134	134
DC5	n/a	n/a	2,805,278	736	978,939	978,939	578	578
DC6	n/a	n/a	35	2	31	31	2	2
DC7	n/a	n/a	124	8	124	124	8	8
DC8	n/a	n/a	6,336	2,978	1,239	1,239	753	753
DC9	n/a	n/a	3,054	21	2,068	2,068	21	21
DC10	n/a	n/a	234	54	126	126	25	25
DC11	n/a	n/a	10,104	425	7,139	7,139	325	325
RI1	n/a	n/a	–	–	–	–	–	–
RI2	n/a	n/a	–	15,449,205	–	–	13,742,893	13,742,893
RI3	n/a	n/a	14	0	14	14	0	0
RI4	n/a	n/a	94,443	42	92,098	92,098	29	29
RI5	n/a	n/a	126,657	5,498	106,511	106,511	5,393	5,393
RI6	n/a	n/a	5,604	162	1,601	1,601	112	112
RI7	n/a	n/a	1,089	354	991	991	322	322
RI8	n/a	n/a	32	3	32	32	3	3
RI9	n/a	n/a	248	23	162	162	22	22
RI10	n/a	n/a	22	4	21	21	4	4
RI11	n/a	n/a	28,690	2,383	16,335	10,723	2,158	2,055
NJ1	n/a	n/a	–	–	–	–	–	–
NJ2	n/a	n/a	–	–	–	–	149,462,202	149,462,202
NJ3	n/a	n/a	–	–	–	–	–	–
NJ4	n/a	n/a	391,296	524	290,031	290,031	520	520
NJ5	n/a	n/a	16,283,617	5,801,726	2,757,406	2,757,406	540,023	540,023
NJ6	n/a	n/a	–	–	–	–	–	–
NJ7	n/a	n/a	–	9,184,415	561,307,649	561,228,326	2,772,362	2,772,272
NJ8	n/a	n/a	–	–	–	–	–	–
NJ9	n/a	n/a	68,732,270	98,455	8,434,103	8,434,063	87,969	87,964
NJ10	n/a	n/a	15,834,882	1,935,798	9,162,035	9,162,023	854,286	854,284
NJ11	n/a	n/a	93,634	2,602	85,376	85,376	2,416	2,416

Note: results indicated with a dash did not solve in the imposed time limit of 10 hours

V. Concluding Remarks

In this report, we proposed improvements to the Raith and Ehrgott (2009) near shortest path enumeration algorithm (2NSP) for solving a biobjective shortest path problem. The improvements tightened the upper bound on the enumeration via 1) a gateway node/arc heuristic offering an initial candidate set of unsupported non-dominated solutions, 2) eliminating the computation of paths that would be dominated by the supported non-dominated solutions, and 3) a combination of both approaches. All of the enumeration approaches as well as two label correcting methods were tested on three types of network problems: terrain-based raster grid networks, NetMaker randomly generated graph networks, and American statewide road networks. Raith and Ehrgott concluded that while 2NSP showed excellent promise in certain problem scenarios, that the exponential behavior of the enumeration made certain problems take far longer to solve with 2NSP than with labeling approaches. The additional bounds that we tested showed significant computational improvement over the original 2NSP algorithm and often resulted in being the fastest of all the methods tested, but they still displayed instances where the combinatorial nature of enumeration resulted in exceedingly large computation times. Overall then, we conclude that the labeling approaches are the most dependable algorithms for solving a biobjective shortest path problem in a reasonable amount of time.

Steiner and Radzik (2008) published some interesting work on enumeration approaches for a biobjective minimum spanning tree (MST) problem. In their case, they used a *k*-Best MST algorithm rather than a near-Best approach. The most interesting innovation of their paper was that they tried solving for more than one adjacent BUSS region at a time, since if one BUSS region is larger than another, then the algorithm may enumerate all solutions in the smaller region during the process of enumerating for the larger region. This approach for biobjective NSP enumeration may be of limited benefit since solution times for large BUSS regions take exponentially more time to solve than smaller ones, but it may be worth testing in future research. Perhaps this could breathe new life into the *k*-best enumerative biobjective shortest path approach of (Coutinho-Rodrigues *et al.* 1999).

References

Byers, T. & M. Waterman, (1984). Determining all optimal and near-optimal solutions when solving shortest path problems by dynamic programming. *Operations Research,* 32, 1381-1384.

Carlyle, W.M. & R.K. Wood, (2005). Near-shortest and k-shortest simple paths. *Networks,* 46, 98-109.

Cohon, J.L., R.L. Church & D.P. Sheer, (1979). Generating multiobjective trade-offs: An algorithm for bicriterion problems. *Water Resources Research,* 15, 1001-1010.

Coutinho-Rodrigues, J., J. Climaco & J. Current, (1999). An interactive bi-objective shortest path approach: Searching for unsupported nondominated solutions. *Computers & Operations Research,* 26, 789-798.

Danna, E., E. Rothberg & C. Le Pape, (2005). Exploring relaxation induced neighborhoods to improve mip solutions. *Mathematical Programming,* 102, 71-90.

Goodchild, M., (1977). An evaluation of lattice solutions to the problem of corridor location. *Environment and Planning A,* 9, 727-738.

Guerriero, F. & R. Musmanno, (2001). Label correcting methods to solve multicriteria shortest path problems. *Journal of Optimization Theory and Applications,* 111, 589-613.

Hart, P., N. Nilsson & B. Raphael, (1968). A formal basis for heuristic determination of minimum path cost. *IEEE Transactions on on Systems Science and Cybernetics,* 4, 198.

Hewitt, M., G.L. Nemhauser & M.W. Savelsbergh, (2010). Combining exact and heuristic approaches for the capacitated fixed-charge network flow problem. *INFORMS Journal on Computing,* 22, 314-325.

Huarng, F., P. Pulat & L. Shih, (1996). A computational comparison of some bicriterion shortest path algorithms. *Journal of the Chinese Institute of Industrial Engineers,* 13, 121-125.

Medrano, F.A. & R.L. Church, (2014). Corridor location for infrastructure development: A fast bi-objective shortest path method for approximating the pareto frontier. *International Regional Science Review,* 37, 129-148.

Raith, A., (2010). Speed-up of labelling algorithms for biobjective shortest path problems. *Proceedings of the 45th annual conference of the ORSNZ. Auckland, New Zealand*, 313-322.

Raith, A. & M. Ehrgott, (2009). A comparison of solution strategies for biobjective shortest path problems. *Computers & Operations Research,* 36, 1299-1331.

Skriver, A.J.V. & K.A. Andersen, (2000). A label correcting approach for solving bicriterion shortest-path problems. *Computers & Operations Research,* 27, 507-524.

Steiner, S. & T. Radzik, (2008). Computing all efficient solutions of the biobjective minimum spanning tree problem. *Computers & Operations Research,* 35, 198-211.