Efficient Multistate Route Computation

David L Rhodes, *Senior Member, IEEE*

***Abstract*—A multi-state graph is defined as a graph where each edge is associated with a stochastic, rather than fixed, value. For cases where the stochastic value is discrete and represents a distance or time, an algorithm is presented which efficiently finds the single-source shortest paths and distances for all combinations of edge values. Infinite and negative weight values are allowed as long as there are no negative weight cycles. Efficiency is achieved by both leveraging dynamic programming and only finding solutions for the set of dominant states that correctly cover any possible edge metric setting for the graph. Although the method has exponential complexity in the worst case, some samples and example graphs are used to illustrate the savings over a brute-force combinatoric approach. Furthermore by leveraging some real-world assumptions, a pseudo-polynomial version is presented that can be applied to larger scale problems. The results of the method are then shown to enable the computation of *most-likely shortest paths*.**

I. Introduction

Efficient algorithmic computation of shortest paths through weighted graphs is a problem of great interest, dating back at least to the 1950's. It is extremely important to many areas such as: computer networking, travel, flow-networks, circuit board and very-large scale integration (VLSI) connection routing, etc. The *single-source shortest-path* version of the problem is to find the shortest distance or a shortest path from a single 'source' vertex to all other vertices in directed graph with weighted edges, while the *all-pairs* version requires finding shortest distances/paths from all vertices to all other vertices. Dijkstra's single-source shortest path algorithm (SPA) published in 1959 [5] applies to graphs with non-negative edge weights, while the single-source Bellman-Ford algorithm [1], [7] solves the more general problem by allowing negative edge weights as long as there are no loops with a total negative weight (although these are detected).

However, there are many domains where graphs with uncertain or random-valued edge values more appropriately represent the problem space. For example, in a road network travel time between points may be subject to statistical variation due to failures, accidents or construction; link delay buffering in a communication network may be subject to dynamic conditions, etc. In these cases, the delay between graph vertices would best be represented as a probability density or mass function, rather than a given (fixed) value. In such stochastic networks, there are several figures of merit or problem formulations worthy of study. One is finding a path with the least expected time (LET) from a given source to destination which is sometimes used to establish routing choices.

In shortest path problems, path weights are added as edges are traversed; since expected values and variance squared are also additive parameters, one can nominally compute the LET path by setting each stochastic edge metric to its expected value and solving using a standard shortest path algorithm [15], [4]. However, when infinite-values are allowed as stochastic edge metrics, expected edge weights are not finite therefore such an approach cannot be used. Infinite valued metrics might represent cases of link failure or closure, etc. When infinite-values are allowed there may in fact be no path with finite expected path length, however there are still paths that are preferential.

The *a priori* form of the problem determines a statistically shortest path from a given source to destination(s) from the outset, while the various dynamic forms of the problem consider adapting path selections as points are reached or edges traversed. Early work by Frank [8] considered determining path distributions for graphs with edge delays of continuous random variables. However, the method also requires the enumeration of all paths from source to destination in order to perform pairwise comparisions. Mirchandani [17] computed a priori statistical shortest paths for graphs with discrete, two-state random edge values. The edge values are a nominal value and infinity, the latter of which represents a disconnected state. The method requires an enumeration of paths, however.

In the dynamic formulation of the problem, edge delays are often times determined only as travel reaches each vertex [3], [21]. As a further extension, edge values may also be both stochastically valued and time-dependent (e.g. [10], [16], [27]). While particular edge delay assignments may be discovered upon arriving at a vertex in dynamic versions of the problem to allow path selection recourse, computed a priori statistically shortest paths are often used to provide statistical guidance for the route beyond the next edge selection.

Another important routing problem is finding the most reliable path between source and destination [17]. Nie and Wu [19] consider the problem of finding a path with a given probability of reaching the destination on time, defining a path as b-reliable if it provides a path with probability b of reaching the destination on time. These formulations were extended by Pan, Sun and Ge [20] to the time-dependent case, defining a path as (t, b)-reliable essentially if the path is b-reliable when departing at time t. A related problem is the 'absolute shortest path problem' that aims to find a path that minimizes the maximum distance over all cases [26]. Szeto and Wong [25] provide a review of stochastic routing problem formulations and approaches.

Related to problems with stochastic edge delays are stochastic flow problems. In these formulations, the delay of each link is typically fixed while flow constraints are represented with random variables. The problem is often stated as finding paths with particular probability of sending d-units of data within T-units of time [13], [28]. However, all minimal paths are often required as a basis in their solution [13].

This paper presents an approach for computing a priori

'most-likely' shortest paths for networks with random discrete edge (arc) weights that may have negative or infinite settings. The computation leverages a new algorithm, called the *Multi-State, Dynamic Shortest Path Algorithm* (MSD-SPA), that more efficiently computes shortest path routes for the multi-state shortest-path single-source problem. Although it has an exponential computational runtime in the worst case, it leverages both dynamic programming and the concept of partial edge metric settings to more efficiently generate a solution set that covers any possible edge metric setting. The results of MSD-SPA are then used in finding most-likely shortest paths (see Section V). A specialized application of MSD-SPA was presented in [22] but neither negative edge weights, application to most-likely shortest paths nor correctness proofs were included. While the a priori most-likely shortest path result is important in itself, the results can also be applied as a foundation in other problems as outlined above.

II. Problem Statement

A graph, $\mathbf{G} = \{\mathbf{V}, \mathbf{E}\}$, is composed as a set of vertices, $\mathbf{V}$, and edges, $\mathbf{E}$. Edges (arcs) are directed, although extension of MSD-SPA to undirected edges is expounded upon in the concluding remarks section.

The set of vertices is $\mathbf{V} = \{v_1, v_2, \cdots\}$ and the set of edges $\mathbf{E}$ is composed of a number of edges, $e_{i\to j}$ where $e_{i\to j}$ is the edge that goes from v_i to v_j. Edges have one or more weights associated with them which is contained in the set $\mathbf{W}$. As is typical for shortest path problems, this weight is interpreted as the distance or time encountered while traversing the edge. Edge weights for an edge $e_{i\to j}$ are expressed as $W(e_{i\to j}) = \{w_1, w_2, \cdots\}$ meaning that the edge $e_{i\to j}$ can take on any of the weight values of w_1, w_2, etc. We assume that the edge weights are sorted from lowest to highest value. These multi-valued weights may represent various conditions for the connection represented by the edge. For example, $W(e_{1\to 2}) = \{10, 15, \infty\}$ states that the edge metric for $e_{1\to 2}$ may take on any value of '10', '15', or '∞'. As expanded upon later, each of the edge weights may also have a probability associated with them; in that case the sum of such probabilities for any given edge must sum to one. Application areas for stochastic graphs so defined are discussed in Section VI.

A particular setting of weights for an edge is called an edge metric setting (Definition 1):

Definition 1: **Individual Edge Weight Metric Setting.** We define an edge metric setting for edge e as either: i) a singular value from that edge's weight set (from $W(e)$), ii) a particular subset of values from $W(e)$, or iii) as a don't care. These will be indicated for an edge e as $M(e) = w \in W(e)$, $M(e) = j\rightharpoonup$, or $M(e) =$ '—', respectively. In the first case the edge metric is set to a single value w in $W(e)$. In the second case the metric is set to the value $j\rightharpoonup$ which is the subset of weights in $W(e)$ from index j in $W(e)$ up to and including the final value in $W(e)$.[1] Such indices will start at one and extend to $|W(e)|$. This form is used to indicate that the edge metric can take on the subset of values in $W(e)$ as $W(e)[j], W(e)[j+1], \ldots, W(e)[|W(e)|]$ where $W(e)[j]$ is the j^{th} in $W(e)$. As will be seen later, these are the type of settings that will be part of the dominant set solution, see Section III for further exposition. ■

With this definition of edge metric setting, we consider settings that *dominate* (or cover) others. For example, suppose there is an edge e with edge metrics $W(e) = \{10, 15, 30\}$. The the metric setting '—'(don't care) includes any setting; a setting of $M = 2\rightharpoonup$ includes the elements of $W(e) = \{15, 30\}$. In these cases we say that a metric setting, M, dominates another setting if it includes all the values of the latter. In the example here, $M_1 = 2\rightharpoonup$ dominates $M_2 = 3\rightharpoonup$ and $M_1 = 2\rightharpoonup$ dominates $M = 30$ since in each case the former includes all the value(s) of the latter.

This definition of domination of a setting for a single edge is now expanded to consider all of the edges of a graph:

Definition 2: **Dominant Metric Settings.** A particular setting of edge metrics, K, *dominates* (or covers) another setting L ($K \neq L$) if for each individual edge metric setting $L_{M(e)}$ $\forall e \in \mathbf{E}$, dominates the setting in $K_{M(e)}$ $\forall e \in \mathbf{E}$. This extends the definition of individual metric setting domination to the full set of edges in $\mathbf{E}$. We say that K is a dominant metric setting over L. Essentially, each edge metric setting in L is the same as in K or K's corresponding metric is broader; all the edge metric settings in L are included in the metrics for K. ■

This definition is then provides a basis for defining a set of unique settings that provide shortest path trees from a source:

Definition 3: **Dominant State.** A Dominant State is a particular edge metric setting for the graph (Definition 2) where changing one or more metric in the settings would change the shortest path distance from a given source vertex to one or more vertices. So L is a *dominant state* if and only if there is no metric setting K that has dominant metric settings over L while providing the same shortest paths and distances to each vertex in G. That is, a dominant state is a setting of edge weights such that altering any of the edge metric settings will change the shortest path distances and paths to some (any) vertex and where there isn't another edge metric setting that dominates this setting while providing the same paths and distances. ■

Finally, finding the *dominant set* is a goal of MSD-SPA, essentially defining a solution to the multi-state shortest path problem that covers the complete combinatorial set:

Definition 4: **Dominant Set.** The dominant set of dominant states is the set of dominant states such that the associated graph is 'covered,' meaning that any possible graph state can be matched to a member in the dominant set. A graph state M matches a member N in a set of states $\mathbf{N}$ if for each edge metric setting in M there is one in N that includes it. By Definition 3, none of the states in the dominant set dominate any other. ■

A. Problem Illustration

To illustrate the definitions and problem, consider the graph in Figure 1. The edges of $\mathbf{G_1}$ each have a nominal value and an infinite metric value as shown. For example, $e_{1\to 2}$ has weights $W(e_{1\to 2}) = \{5, \infty\}$. For the nominal value setting, that is

[1] recall W is sorted

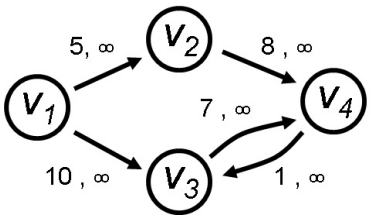

Fig. 1. Simple four vertex graph $\mathbf{G_1}$. Vertex v_1 is the source.

the lowest value setting for each edge weight, from vertex 1 the shortest path tree is composed of edges $e_{1\to2}, e_{2\to4}$ and $e_{1\to3}$ and the reachability metric (d) for each vertex is $d[1] = 0, d[2] = 5, d[3] = 10$ and $d[4] = 13$.

Each of the five links has two possible metric values, and hence in this case there are $2^5 = 32$ graph states if enumerated (there are $\prod_{e\in E}|W(e)|$ combinatorial graph states). This would, of course, imply that 32 runs of a SPA would be required to determine shortest paths for all possible combinations using a brute-force approach. In the next section, it will be shown that the MSD-SPA algorithm can completely analyze this case considering only nine graph states while also leveraging a dynamic programming approach that further lowers computational cost. As will be proven later, these nine graph states are the dominant set for the graph $\mathbf{G_1}$ which is shown in Table I. Note that these nine states 'cover' any possible metric setting, for example, the graph state $M = \{e_{1\to2} = 5, e_{1\to3} = \infty, e_{2\to4} = 8, e_{3\to4} = 7, e_{4\to3} = \infty\}$ is covered by the seventh entry in Table I (the value of $e_{3\to4}$ is a don't care in the table, so it includes any setting for this edge). The last column shows the number of combinatorial states that are covered by the dominant state, note that this adds to 32, the number of combinatorial cases.

III. The Multi-State Dynamic Shortest Path Algorithm, MSD-SPA

This section defines the MSD-SPA method, while completeness and correctness of the method follows in the next section. The variables and notation used is:

- $\mathbf{G} = \{\mathbf{V}, \mathbf{E}\}$ is a directed graph. A vertex $s \in \mathbf{V}$ is the 'start' or source vertex. $\mathbf{W}$ is the multiple weight settings for each edge, such that $W(e)\ \forall e \in \mathbf{E}$ is the set of edge weights. These have indices $j = 1, \ldots, |W(e)|$ so that W(e)[j] is the j^{th} weight value for edge e. Later probabilities for each edge weight will be added as part of the input, but these are not required for determining the dominant set.
- In relation to the solution, $\mathbf{T}$ will contain a set of shortest path trees for each member of the dominant set. Each tree $t \in \mathbf{T}$ has properties:
  - a dominant state setting, $t_{\mathbf{M}}$, associated with the tree.
  - $t_{\mathbf{Q}} \subseteq \mathbf{V}$ is the set of vertices included in this shortest path tree so far.
  - $t_{d[v]}$ is the shortest path distance from s to v
  - $t_{I[v]}$ is the incoming edge to vertex v that provides a shortest path

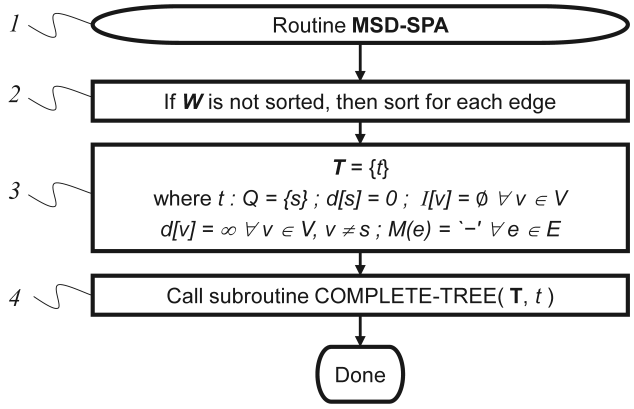

Fig. 2. The MSD-SPA initialization routine

- a predicate function, PRED, that can be used to control recursive exploration at the expense of losing completeness and optimality. It can be initially ignored in terms of understanding the core algorithm.

When the context is clear, we simplify the notation by dropping the t in $t_{M(e)}$, $t_{\mathbf{Q}}$, $t_{I[v]}$, and $t_{d[v]}$ to just $M(e)$, $\mathbf{Q}$, $I[v]$, and $d[v]$, respectively.

The initialization routine to be used in the MSD-SPA method is shown in Figure 2. As can be seen, the MSD-SPA initialization routine first sorts (ascending) the edge metrics for each edge should they not already be provided as sorted sets of values. Duplicates can also be removed as they do not add a new edge weight setting.[2] Next, the shortest path tree solution set, $\mathbf{T}$, is initialized to include a single tree, t that in turn is initialized such that $t_Q = \{s\}$, $t_{d[s]} = 0$, $t_{d[x]} = \infty\ \forall x \in V, x \neq s$, $t_{I[v]} = \varnothing \forall x \in V$, $t_{M(e)} = $ '—' $\forall e \in \mathbf{E}$. Note that $\varnothing$ here is indicating a 'null' value, not an empty set. The variable $t_{I[v]}$ indicates an incoming edge for vertex v, and $\varnothing$ is used to indicate 'not set'. The initial part of the MSD-SPA algorithm then calls the COMPLETE-TREE routine (the core part of the method described in this paper), passing arguments of $\mathbf{T}$ and t. For simplicity, the inputs ($\mathbf{G}$, start vertex s and edge weights $\mathbf{W}$) are considered globals that are shared between the two routines.

The recursive part of the MSD-SPA method is shown in Figure 3. Subscript notation for tree t is dropped for Q, d, I, and M in the figure. The COMPLETE-TREE routine takes arguments of $\mathbf{T}$ and t. It begins (step 2) by trying to find an edge, e_{MIN} that starts at some vertex $u \in Q$ and extends to a vertex v (v can be in $\mathbf{Q}$), where the value of the current shortest path to u plus the edge metric ($d[u] + W'(e_{MIN})$) is minimal (ties are broken arbitrarily as in many other SPA methods, but this topic is expounded upon later). The function W' returns the edge weight for an edge e based on the setting

[2] if each also has an associated probability these would have to be combined as well

TABLE I
THE NINE DOMINANT STATES OF THE GRAPH IN FIG. 1. '—' IS A DON'T CARE SETTING.

$e_{1\to2}$	$e_{1\to3}$	$e_{2\to4}$	$e_{3\to4}$	$e_{4\to3}$	$d[v_2]$	$d[v_3]$	$d[v_4]$	Cases Covered
5	10	8	—	—	5	10	13	4
∞	10	—	7	—	∞	10	17	4
∞	∞	—	—	—	∞	∞	∞	8
∞	10	—	∞	—	∞	10	∞	4
5	∞	8	—	1	5	14	13	2
5	∞	∞	—	—	5	∞	∞	4
5	∞	8	—	∞	5	∞	13	2
5	10	∞	7	—	5	10	17	2
5	10	∞	∞	—	5	10	∞	2

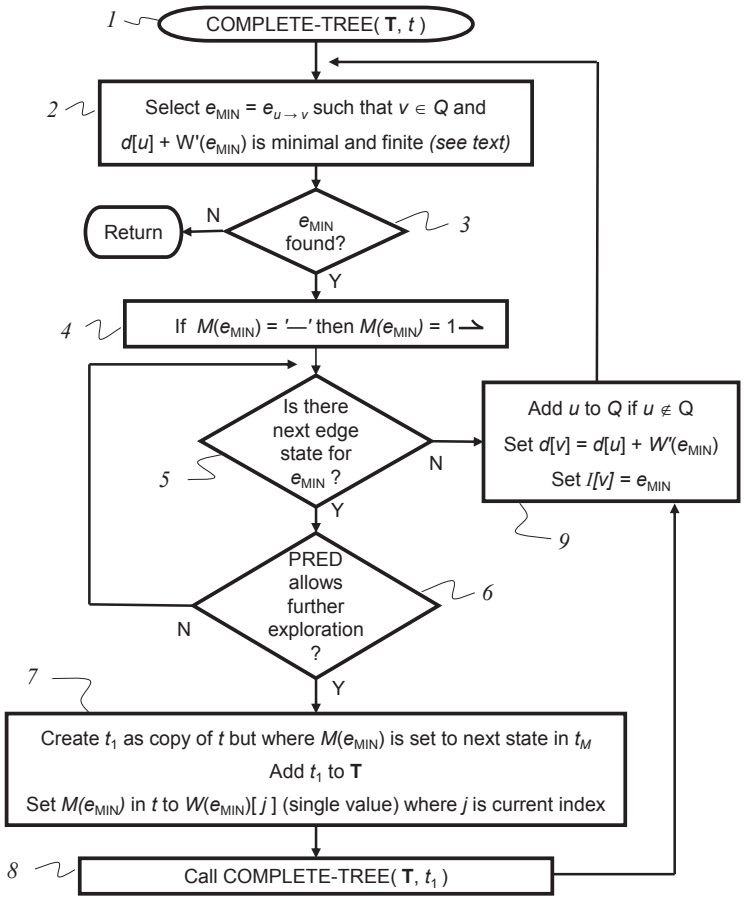

Fig. 3. COMPLETE-TREE routine.

in M, as shown in Equation 1.

$$W'(e) = \begin{cases} W(e)[0] & \text{if } M(e) = \text{'—'} \\ W(e)[j] & \text{if } M(e) = \text{value at index } j \text{ in } W(e) \\ W(e)[j] & \text{if } M(e) = j\rightharpoonup \end{cases} \quad (1)$$

This is essentially choosing the 'current value' in $W(e)$ for the edge based on the metric setting for the edge in M, with the extension of choosing the lowest value ($W(e)[0]$) for don't care settings. The setting at step 4 follows that of Equation 1, if the first option was selected (M was a don't care) then $M(e_{MIN}) = 1\rightharpoonup$ which is the all the values of $W(e_{MIN})$ starting with the lowest. This is essentially saying that the value for this edge can be from the base value and up and is an initialization step that changes the don't care initialization that occurred in Step 3 of the procedure in Figure 2. Otherwise, it is not changed. Should such an edge e_{MIN} not be found (with a finite value) at step 3, then this means that either the shortest path tree is complete or the graph is not fully connected and the method returns. Detection of negative weight cycles can be readily incorporated into the routine but this is not shown; this will be elaborated in the next section.

At the next step (step 5), a check is made to determine if there is an additional edge metric associated with the edge e_{MIN}. By virtue of the sort that was performed during initialization in the MSD-SPA method (step 2 in Figure 2), additional metrics to be considered will have a *larger* value than the metric for this edge in the current state. If there is no such additional state to consider, then the method proceeds (step 9) by adding v to $\mathbf{Q}$ if it is not already in $\mathbf{Q}$ and defining $d[v] = d[u] + W'(e_{MIN})$ (current state value), $I[v] = e_{MIN}$, and continues to step 2 to find the next edge. It is important to realize that $e_{MIN} = e_{u \to v}$ is chosen such that $d[v]$ is minimal across all possible selections to vertex v from any vertex $\in \mathbf{Q}$.

Should an additional (higher) value be available for the edge e_{MIN} (including an infinite-value setting) at step 5, the method next checks the control predicate, PRED, at step 6 to determine if the recursion should be allowed to continue. If not, the algorithm loops back to step 5 to check further states on e_{MIN}. If further exploration is allowed, the algorithm continues at step 7 by copying the current shortest path tree, t with its attributes into a new tree called t_1, except the value $M(e_{MIN})$ in t_1 reflects the next higher valued edge metric subset for e_{MIN}. For example, if the current $M(e_{MIN})$ is $j\rightharpoonup$ or v such that v is the j^{th} element in $W(e_{MIN})[j]$, then it is set to $M(e_{MIN}) = (j+1)\rightharpoonup$ in t_1. The setting for $M(e_{MIN})$ in the current tree is then set to a single value of $W(e_{MIN})[j]$. That is, the metric for the current tree is now the single value at $W(e_{MIN})[j]$ while the metric for the new tree t_1 is $(j+1)\rightharpoonup$ which includes all the edge weights from $W(e_{MIN})[j]$ and higher. This new tree is then completed by recursively calling COMPLETE-TREE with $\mathbf{T}$ and t_1 as arguments (step 8). After return, the algorithm continues at step 9.

Step 7 is the key step in a dynamic programming approach, it takes the current partial solution and uses it as the starting point of another solution. The viability of dynamic programming in any domain is the leveraging of optimal substructure across solutions. In this case, the distance vectors (d), the incoming edges I and edge metric settings M are suitable candidates for a dynamic programming approach since the sorting of edge weights eliminated the possibility of lower distance vectors within the partial solution (d, I, M, and Q) since lower valued edge metrics are considered before higher valued ones. Correctness of this method is proven in the following section.

IV. Correctness and Completeness of MSD-SPA

The correctness proof requires a few steps, and is shown by first showing that the trees found by the method are indeed shortest paths and that there are no states of significance that are missed notwithstanding the control provided by $PRED$ or ties (although the method can be extended to include these equivalent shortest path trees as well as is discussed in the concluding section). Throughout this discussion, it may be useful to refer to Figure 4.

Lemma 1: An input where steps 5, 6, 7, and 8 of COMPLETE-TREE are not employed is first considered, as would be the case for graphs with only a single value for each edge metric (e.g. the deterministic case). In this case, along with the initialization of MSD-SPA (Figure 2), steps 2, 3, 4 and 9 of Figure 3 find a shortest path tree and the attribute $d[v]$ will be the shortest distance from vertex s to v for all reachable vertices $v \in V$. Attribute $I[v]$ will be the incoming edge for a shortest path to vertex v.

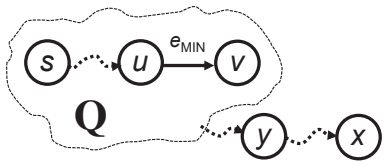

Fig. 4. Illustration of the set Q, start vertex s, vertices v and u, and edge e_{MIN}. Vertex v is shown inside of set $\mathbf{Q}$ but it may be outside as well. See text concerning vertices x and y.

Proof: Since dynamic programming is to be used for subsequent edge metric settings, relaxation or other approaches that alter graph data beyond the set $\mathbf{Q}$ cannot be used. However, the basic approach utilizes similar greedy techniques as found in traditional shortest path algorithms (and is a shortest path algorithm itself). For the first part of the proof, it is shown that all reachable vertices $\in V$ will eventually be added to $\mathbf{Q}$. For the purpose of contradiction, assume that there is a vertex x that is reachable (with a finite metric) from s but that is never added to $\mathbf{Q}$, and furthermore assume that vertex y is the first vertex $\notin \mathbf{Q}$ along the path to x (see Figure 4). Since there are no negative-valued cycles in G, eventually e_{MIN} cannot be chosen so as to reduce the value of any $d[v]$ for any $v \in \mathbf{Q}$. Since the value of $d[y]$ is initialized to ∞ and since, by hypothesis, y is reachable (with a finite edge metric from $\mathbf{Q}$), an edge with finite value from a vertex in $\mathbf{Q}$ to y will eventually be chosen as e_{MIN}. This argument can then be applied to all vertices along the path to x and therefore vertex x must eventually be added to set $\mathbf{Q}$.

The second part of the proof shows that at conclusion that all $d \in \mathbf{Q}$ values are a shortest distance value from s. From the first part of the proof, we know that all reachable vertices are eventually added to $\mathbf{Q}$. Suppose at conclusion of the method that for some vertex v' that $d[v']$ is not the actual shortest path distance. At step 2 of the algorithm, along with all other edges eminating from each vertex in $\mathbf{Q}$, every edge incoming to vertex v' is examined and if a lower value for d' to vertex v' from any vertex $\in \mathbf{Q}$ it may be a candidate for e_{MIN} selection. Since the routine concluded it must be the case that such an e_{MIN} does not exist, so in order for $d[v']$ to not be the shortest path, it must be that the distance value for the vertex from the incoming edge to v', say its u', and thus $d[u']$ must not be a shortest distance. Similarly, the same argument can then be made for u' and since the reverse path from any node starts at s, and since $d[s]$ is correct (initialized to zero), the assumption that there is a $d[v]$ for any $v \in \mathbf{Q}$ that is not the shortest distance must be invalid. Since e_{MIN} was chosen as the shortest incoming edge for vertex v (as saved in $I[v]$), this same iterative argument can be used to show that $I[v]\ \forall v \in \mathbf{Q}$ is an edge that provides a shortest path from s. ■

Although not shown in Figure 3 for simplicity, negative cycles can be checked in the course of the method as well.

For each execution of steps 2 and 9, an edge $e_{MIN} = e_{u \to v}$ is chosen such that the currently-known distance $d[v]$ to vertex v is reduced to the lowest value among all e_{MIN} selections. Consider Figure 4 at the point when a new vertex y is added to $\mathbf{Q}$. There may be many edges $e_{y \to q}$ where q is a vertex in $\mathbf{Q}$ which are negative valued. This may lower the currently known distance(s) to one or more members in $\mathbf{Q}$. However, since the *smallest* new distance $d[q]$ is chosen in step 2, in subsequent iterations of step 2, vertex distances to members in $\mathbf{Q}$ can only be updated at most once (without a new vertex being added to $\mathbf{Q}$). There are at most $|V| - 1$ vertices added to $\mathbf{Q}$ and each addition can cause at most one update to each vertex in $\mathbf{Q}$ (if no negative-valued cycles). Including the original update when the vertex was added to $\mathbf{Q}$, the most times a vertex distance can be updated is $|V|$. If any vertex is updated more than this (step 9) then a negative-valued cycle exists. Therefore, an update counter for each vertex can be added to the method to detect negative-valued cycles.

Computational complexity of this part of the method can be seen to be $O(|E||V|^2)$. At step 2, selection of e_{MIN} requires evaluating the distance plus edge metric for each edge from $\mathbf{Q}$ (this is $O(|E|)$). Since each vertex can have its distance reduced at most $O(|V|)$ times (for a graph with no negative cycles) and there are $O(|V|)$ vertices, the overall computational complexity of this part method is therefore $O(|E||V|^2)$. This is higher than the traditional Bellman-Ford algorithm which is $O(|E||V|)$ since we cannot relax all edges at each iteration and are only, in a sense, relaxing one at a time.

The next step, utilizing and extending Lemma 1, is to show that the recursive part of the routine (steps 5, 6, 7, and 8) produces a set of shortest path trees for each metric setting.

Lemma 2: The trees and distances produced by the COMPLETE-TREE routine are shortest path trees/distances for their associated edge metric settings (t_M).

Proof: A close examination of the recursion in COMPLETE-TREE shows that the when a new graph state and solution tree is created (t_1), it is a copy of the present tree, t, including the current distances d *prior* to setting $d[v]$ in step 9. As any additional edge metric states to be considered in the future, either for e_{MIN} or any other edge, will have *larger* values (due to the sorting step at initialization), there is no way that the distances in $\mathbf{Q}$ discovered so far for t can be lower in t_1. Note that on return the procedure continues at step 9 and by Lemma 1 the shortest path tree/distance is completed. Thus, the COMPLETE-TREE routine does indeed produce shortest path trees/distances for various edge metric settings (leveraging dynamic programming by utilizing partial shortest path trees/distances discovered so far right before recursion).

As this reasoning is readily extensible to the recursive series, it can be concluded that all trees produced by the algorithm are indeed shortest path trees for their associated t_M metrics. ■

The final part of the overall proof is a completeness criteria. Note that Lemma 2 shows that the trees found by MSD-SPA are all shortest path trees for their associated edge metric settings (t_M), but it does not elaborate on *which* trees are found. Theorem 1 will show that (ignoring the possibility of ties when selecting the edge e_{MIN}) that the method actually finds the solutions for the dominant set.

Theorem 1: The MSD-SPA algorithm finds the dominant set with shortest distances and a shortest path trees.

Proof: For the purposes of contradiction, assume that there is a dominant state with edge metric settings, $\mathbf{M}_{nc}$, not included in the solution set $\mathbf{T}$ found by the algorithm. Let t_{nc} be the tree created by the edge selections in $\mathbf{M}_{nc}$. Let Q_{nc} and d_{nc} be the set of vertices and distances, respectively, in t_{nc}. Let the dominant state solution t_{pm} be the dominant state in T such that it has the best 'partial match' to t_{nc}. Here, the best partial match to M_{nc} and it's associated trees t_{nc} is the selection of a dominant state M_{pm} and associated tree t_{pm} such that the largest number of edge metric values in t_T as rooted from starting vertex s have the same metric values as those in t_{nc}. As all solutions start with the vertex s with a distance of $d[s] = 0$, such a match is always possible. Assume that $e_{x \to y} \in E_{nc}$ (x may be s) in the solution for M_{nc} is the first edge (closest to the source) that is not matched by any solution found by the algorithm.

Now consider the action of the MSD-SPA algorithm at the point where Q_{nc} is the set of vertices in the shortest path tree found so far. Note that prior to continuing to step 9 and adding u to Q, the algorithm in its recursions will check the sum $d[v] + W'(e)$ as various edges are selected as e_{MIN} and their weight assigned to their next increased metric value in each recursion. This process is continued until a finite valued $W'(e_{MIN})$ cannot be selected from any vertex in Q. For the case where $e_{x \to y}$ is finite valued in M_{nc}, it must be selected (recall that we are not concerned with multiple equal choices) in one of these iterations. After completing any further recursions, this choice is finalized in step 9 in the tree for the current metric setting on this edge. If the setting for $e_{x \to y}$ is infinite in M_{nc}, then this must be explored by the algorithm as well. In this case, edge $e_{x \to y}$, with a lower valued metric[3], must have been selected as e_{MIN} in prior recursions since, by assumption, the metric of infinity is relevant and a lowered valued metric for edge $e_{x \to y}$ would have been chosen in prior recursions. The value of ∞ for $e_{x \to y}$ would have then been explored by construction at steps 5 through 8. Thus, this proves the contradiction false. As this argument readily applies to the remaining vertices of the state M_{nc}, no such state can exist. This then implies that no dominant states are 'missed' by the algorithm.

The final part of the proof shows that *only* dominant states are found. In order for a solution found by the algorithm to not be dominant, some of the edge metric values within the solution found could be converted to other values without impact. When the algorithm stops (step 3 when an edge, e_{MIN}, cannot be found with a finite value), the set Q contains all the vertices reachable via a shortest path tree. Obviously, none of the edge metric settings, t_{nc}, for these edges can be altered without changing at least one of these shortest path distances. This implies that all metric values for edges

[3]a metric settings of *only* ∞ for an edge is not allowed—the edge is simply never part of the graph in this case

determined to be significant cannot be changed to don't care settings.

By Lemma 2 which shows that distances, d, and trees, t, found are shortest path distances/trees and since the algorithm has been shown here not to miss any dominant states, and that it only finds dominant states, the algorithm has been shown to find the dominant set. ■

V. Most-Likely Shortest Path

The dominant set of edge metric settings does not depend upon the probability of the edge weight settings, however the probability of these states occurring does. In the algorithm of Figure 3 the coverage of all possible edge weight combinations in terms of dominant states is computed. We can extend the input slightly by assigning probabilities to each edge weight, denoted as $Pr(e)[j]$ which is the probability of the j^{th} edge weight ($W(e)[j]$) occurring for edge e. For each edge, the sum of the probabilities for each edge weight should add to one, that is $\sum_{k=1..|W(e)|} Pr(e)[k] = 1 \ \forall e \in \mathbf{E}$.

When each of the edges in a graph have their individual metrics set per Definition 1, we refer this an edge metric state, or simply *state*. That is, a state is merely a term for given edge metric settings M for **all** the edges in the graph, be each of them set to individual values, a range of values or a don't care setting. If we assume that edge weight settings are independent from each other, we can readily compute the *combinatorial probability that each dominant state covers*, $P(M)$, via Equation 2.

$$
\begin{aligned}
Pr(e)[j] \quad &= \text{probability for edge } e \text{ at weight setting } j \\
p(e,M) \quad &= \begin{cases} 1 & \text{if } M(e) = \text{'—'} \\ \sum_{j..|W(e)|} Pr(e)[k] & \text{if } M(e) = j\rightharpoonup \\ Pr(e,M)[j] & \text{if } M(e) = v \text{ where } v \text{ is the value at index } j \text{ in } W(e) \end{cases} \\
P(M) \quad &= \prod_{\forall e \in \mathbf{E}} p(e,M) \qquad \text{where } p(e) \text{ is the probability factor for each edge per above}
\end{aligned}
\tag{2}
$$

The routine of Figure 3 can be adjusted to compute these state probabilities and counts in-process to avoid requiring an additional post-computation as in Equation 2, but this is not shown here.

The following definition provides a new, simple notion for defining shortest path selection:

Definition 5: **Most-likely Shortest Path (MLSP).** The most-likely shortest path is the path that provides the greatest chance of being a shortest path. ■

For illustration, consider Figure 5 which shows two similar small graphs that each have two edges from source to destination; they vary in the edge weight assignments. In Figure 5(A), e_{TOP} has three possible metric settings of 10, 20 and 26 with probabilities of 0.25, 0.25 and 0.50 respectively. e_{BOT} has two possible metric settings of 11, and 29 each with probability of 0.5 (these edge metrics are chosen to avoid shortest path ties for clearer illustration).

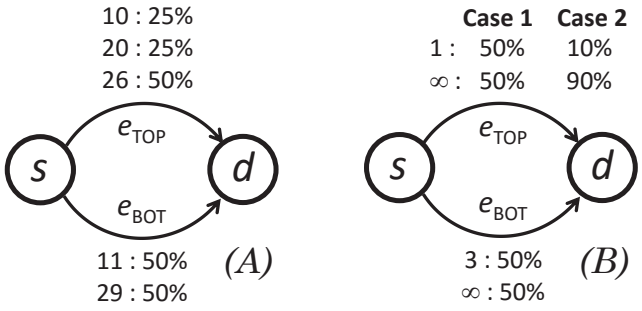

Fig. 5. Simple graphs with source s and destination d.

TABLE II
Combinatorial states of the graph in Fig. 5(A)

e_{TOP}	e_{BOT}	P	shortest d	Best Path
10	11	$1/8$	10	TOP
20	29	$1/8$	20	TOP
26	11	$1/4$	11	BOT
10	29	$1/8$	10	TOP
20	11	$1/8$	11	BOT
26	29	$1/4$	26	TOP

The expected distance for traveling to destination d along edge e_{TOP} is 20.5 and the expected distance for traveling along edge e_{BOT} is 20.0, so e_{BOT} might be chosen as the LET path. However, in contrast consider Table II which enumerates all the combinations of edge metric settings[4]. Note that e_{TOP} is the best choice $5/8$ of the time. Thus, even though e_{BOT} has a shorter expected distance, there is a higher likelihood that chosing e_{TOP} will result in experiencing a shortest path.

The notion of most-likely shortest path is even more important for graphs that allow infinite-valued metrics (failures). In these cases, there is no means of using expected value as an edge value distance metric as it is not finite.

Consider the graph of Figure 5(B) which defines two cases, the only difference in which are the probabilities associated with the metrics for edge e_{TOP}. As seen in Table III, all possible combinations of metric settings for each of the two edges (states) and the associated shortest distance and edge selection for that distance is given. Note that the shortest path distance and edge selection does not depend on edge weight (and hence state) probabilities. But the last two columns show state probabilties and this *is* important in selecting the MLSP.

In case 1, each of the edge metrics for each of the two edges is 50% and the probability of each of these states occuring is given in the fifth column. Intuitively, and via MLSP (Definition 5), we would choose e_{TOP} since that provides a better chance of being the shortest; in this case there is a 50% chance of encountering a shortest path along e_{TOP} while only a 25% chance for e_{BOT}. However, when the edge metric probablities are altered to Case 2, then the better choice is

[4]note that MSD-SPA covers these 6 combinatorial settings with a dominant set of 4 entries, but the table shows all 6 combinations for clarity

TABLE III
SHORTEST PATHS FOR GRAPH IN FIG. 5(B)

e_{TOP}	e_{BOT}	shortest d	SP edge	State P Case 1	State P Case 2
1	3	1	TOP	0.25	0.05
∞	3	3	BOT	0.25	0.45
1	∞	1	TOP	0.25	0.05
∞	∞	∞	n/a	0.25	0.45

e_{BOT}. This is the better choice as it has a 45% chance of providing the shortest path, while e_{TOP} only provides a 10% chance. We next consider how to compute MLSP in the context of the MSD-SPA result.

A. Computing MLSP

At the conclusion of MSD-SPA we have the shortest distance d, and incoming edge that provides that distance I, for each vertex in the graph from the source s for every possible state of edge metric settings, or state (in 'compressed' form per the dominant set). For cases where probabilities are assigned $W(e)[j]\ \forall e \in E$, we also know the probability of each of these states occurring from Equation 2. This equation assumes that each edge metric is independent from others, but the method could be altered to address correlated or other situations as well; determing the MLSP only requires that the probability of each state be defined.

The method is iterative starting at the destination d and working backwards to the source. For all edges incoming to d, sum their probability of being in the shortest path. This merely entails examining the incoming shortest path edge I at vertex d across all the states M (edge metric settings) in the dominant set in accordance with the probability of this state occurring. The incoming edge with the largest probability value is therefore part of a MLSP[5]. This process is then iteratively applied to the source vertex of the incoming edge selected all the way back to the source. The resulting edges so selected then form a MLSP as proven in Lemma 3 below.

As additional information, by examining the outgoing edges for each vertex along the MLSP, we can determine the probability (certainty) of each selection in regards to providing a MLSP. For a vertex v, for each outgoing edge $e_{v\to q}$ we can sum the probability weighted number of times that each edge is part of the discovered MLSP. This value provides the probability that such an edge at that vertex is the best choice in providing a MLSP. For example in very simple case of Figure 5(A), the probabilty that edge e_{TOP} is the best choice to provide a MLSP from vertex s is $5/8$. And e_{TOP} was firstly chosen as the MLSP since it was the incoming edge to d that provided the shortest path $5/8$ of the time.

Lemma 3: The constructive method described in this section produces a MLSP.

Proof: MSD-SPA finds shortest paths for the dominant set, which in turns cover the entire combinatorial space. In the process outlined in this section, we first select the incoming edge to d as part of the MLSP as the one that provides the highest probability of being on a shortest path. Since the entire, weighted combinatorial space is considered in this selection, and notwithstanding ties, it is not possible that a different incoming edge to d provides a greater probabilitistic number of shortest paths. This argument can then be applied to each vertex found at the tail of each of these edges back to the source. The source has no incoming edges in any shortest path tree and hence the set of edges in the MLSP at the source is empty. ■

[5]as with deterministic shortest paths, most-likely shortest paths need not be unique

TABLE IV
COMPARING MSD-SPA FOR GRAPH OF FIG. 1 WITH VARIOUS METRIC PROBABILITIES. Q SIZE AVERAGES AND BEST ROUTE PROBABILITIES ARE ROUNDED TO THREE DECIMAL POINTS.

Input	Comb. case count	# dom. states	Avg. size of Q at recursion	MLSP to vertex 4
$\mathbf{G_1}$	32	9	47.222%	$1 \to 2$ p = 0.571 $2 \to 4$ p = 1
$\mathbf{G_2}$	32	12	56.250%	$1 \to 2$ p = 0.750 $2 \to 4$ p = 1
$\mathbf{G_3}$	243	41	60.976%	$1 \to 2$ p = 0.608 $2 \to 4$ p = 1

VI. RESULTS AND EXAMPLES

As discussed in part before, such results can be applied in many application domains including: network and traffic planning and provisioning in identifying critical links; in alternate route computation and quality of service applications; as well as in other ways and in other application areas. In this section, a set of example inputs are used to demonstrate the algorithm. This is done using three application areas, which are: (i) synthetic, (ii) computer networks, and (iii) transportation.

Table IV shows several results for a series of synthetic inputs to introduce result formats. For each input, the number of combinatorial cases (as would be needed in a brute-force approach), the number of dominant states found for the input by MSD-SPA and the average size of the set Q at the point of recursion (i.e., at step 7 in Figure 3) are shown. This latter measure gives an idea of savings provided by the dynamic programming part of the algorithm. The final column shows the most-likely shortest path found. The input descriptions are:

- $\mathbf{G_1}$ is the graph of Figure 1 where each edge setting probability is 50% base value and 50% failed. The dominant set solution for this case is shown in Table I.
- $\mathbf{G_2}$ is the graph of Figure 1 but where the edge state ∞ is replaced for each edge with double its low metric setting (e.g. $W(e_{1\to2}) = \{5, 10\}$ instead of $W(e_{1\to2}) = \{5, \infty\}$). Each edge setting probability is 50% base value and 50% the doubled value.
- $\mathbf{G_3}$ is the graph of Figure 1 but where the doubled, low value edge metric is added as well (e.g. $W(e_{1\to2}) = \{5, 10, \infty\}$). It can also be viewed as the graph $\mathbf{G_2}$ with the ∞ metric added for each edge. Each edge setting

probability is 50% base value and 30% the doubled value and 20% the ∞ value.

In these examples, the MLSP route is the same, namely vertices: $1 \rightarrow 2 \rightarrow 4$, but the probability at vertex 1 (the start) for selecting the MLSP varies. In the case of $\mathbf{G_1}$, the certainty that the $e_{1\rightarrow 2}$ will result in the MLSP is approximately 57.1%. However, once at vertex 2, taking the edge $e_{2\rightarrow 4}$ is always the best choice to reach vertex 4. In $\mathbf{G_2}$, where the ∞ edge weight metrics are replaced by the double base value, it is 75% certain that taking this path is the best. Likewise for $\mathbf{G_3}$ taking the first step on this route is about 60.8.2% certain to give the shortest path.

The next area considered is computer networking. Computer and Internet routing is broadly divided into two classes, namely exterior gateway protocols and interior gateway protocols. The former includes methods such as Border Gateway Protocol (BGP), while the latter includes link-state types such as Open Shortest Path First (OSPF) and Intermediate System to Intermediate System (IS-IS) and distance-vector methods such as Routing Information Protocol (RIP) and Interior Gateway Routing Protocol (IGRP) [9]. It is the link-state interior methods where MSD-SPA can be utilized to advantage. In short, routers using these methods exchange link information ('link state advertisements') such that all routers have complete network topology and metrics. Each router then employs a shortest path algorithm to determine routing from itself to each address or address group within the network such that it contains a 'next-hop' entry for each destination.

In [30], it is pointed out that OSPF re-convergence after a link failure is a lengthy process that can require tens of seconds; the authors also point out that this problem is much exacerbated in the face of multiple failures. A 'local' response to re-routing upon a single link failure has been proposed [18] that requires participating routers to recognize that traffic is coming into an interface that would not normally be part of the shortest path and therefore should be treated differently (e.g. the receiving router should recognize that a link failure has occurred that it is not aware of). Using MSD-SPA we can pre-compute routing for any or all link failure states so it can be used in a similar manner for multiple failures. Rather than keeping a single next hop entry for each destination, the router can keep several entries that include failure routes. These could be as large as the entire dominant state table, or a subset of these. Also, with various edge metric probabilities that reflect different failure conditions, the routers could use various MLSP routes, instead of just the shortest path for the base metrics.

The OSPF example from Figure 1 in [23] is repeated here as Figure 6. This example has 26 directional edges, each of which have the nominal metric shown as well as an added ∞ weight, which requires $2^{26} = 67,108,864$ combinatorial states. MSD-SPA covers all of these combinations with 508 dominant states. In Section VII, we will consider how to use the predicate feature to limit the combinatorial space to no more than k-simultaneous failures. If we limit this example to no more than two simultaneous failures using the predicate control feature, then full coverage is obtained with only 46 dominant states.

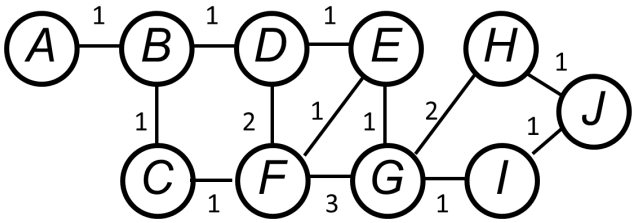

Fig. 6. Computer network subject to OSPF routing with nominal edge weights. Edges shown are bidirectional.

TABLE V
DETERMINING MLSP ROUTES FOR MULTIPLE ROUTE FAILURES

Instance	MLSP Route
10% fail rate for all links	$A \rightarrow B$ [p = 1] $B \rightarrow D$ [p = 0.782518] $D \rightarrow E$ [p = 0.963656] $E \rightarrow G$ [p = 0.998564] $G \rightarrow I$ [p = 0.840336] $I \rightarrow J$ [p = 1]
50% failure rate for links with vertex D, 10% fail rate for all other links	$A \rightarrow B$ [p = 1] $B \rightarrow C$ [p = 0.70908] $C \rightarrow F$ [p = 1] $F \rightarrow E$ [p = 0.795874] $E \rightarrow G$ [p = 0.997397] $G \rightarrow I$ [p = 0.840336] $I \rightarrow J$ [p = 1]

Table V shows the results for the shortest statistical path technique from vertex A to vertex J. The first entry in the table sets a 10% failure rate for all links. This is done by setting the nominal edge weight metric shown to a 90% probability and a ∞ metric to 10% probability for each edge. The second entry adds a 50% failure probability for vertex D, which is modeled by altering the ∞ metric setting to 50% for links to and from vertex D.

There are several interesting observations that can be made. The first is that MLSP preferred route changes from $A \rightarrow B \rightarrow D \rightarrow E \rightarrow G \rightarrow I \rightarrow J$ in the first case to $A \rightarrow B \rightarrow C \rightarrow F \rightarrow E \rightarrow G \rightarrow I \rightarrow J$, avoiding vertex D. However, even with a rather high failure rate for vertex D, the selected edge from vertex B to vertex C in the latter case is only preferred at a rate of about 70.9%. Since the only alternate route from vertex B is to D, this means that there is about a 29.1% probability that going through D, even with its high failure rate, will result in taking the shortest path.

Another observation is that each route goes through vertex E and then uses $e_{E\rightarrow G}$, at a high rate of over 99.7% in each case. Why not 100%? Arrival at vertex E was through vertex D or F, respectively, for the two cases so this would basically mean going backwards (recall the edges are bidirectional for this example). It means that there are a few cases among all the combinations that are shorter from vertex E by going to either D or F and hence a small chance that going back through these nodes will be the shortest route. The shortest path selection in MSD-SPA could be altered to not consider outgoing edges to vertices that are already part of the shortest path found so far; if this were done then the $e_{E\rightarrow G}$ selection

TABLE VI
DETERMINING MLSP ROUTES FOR MULTIPLE DELAY SITUATIONS.

Instance	Dom Set Size	MLSP Route
1: miami to sanfran 50% chance of 1 hour delay, all links	49,728	miami → atlanta [p = 1] atlanta → dallas [p = 0.50000] dallas → phoenix [p = 1] phoenix → la [p = 1] la → sanfran [p = 1]
2: miami to sanfran 50% chance of 3 hour delay, all links	68,904	miami → atlanta [p = 1] atlanta → stlouis [p = 0.496094] stlouis → denver [p = 1] denver → sanfran [p = 1]
3: miami to sanfran 50% chance of 4 hour delay in la, 1 hour other links	46,720	miami → atlanta [p = 1] atlanta → stlouis [p = 0.796875] stlouis → denver [p = 1] denver → sanfran [p = 1]
4: washdc to phoenix 50% chance of 1 hour delay, all links	43,008	washdc → atlanta [p = 0.65625] atlanta → dallas [p = 1] dallas → phoenix [p = 1]
5: washdc to phoenix 50% chance of 3 hour delay in dallas, 1 hour other links	44,032	washdc → chicago [p = 0.734375] chicago → denver [p = 1] denver → phoenix [p = 1]
6: washdc to phoenix 30% 2 hour and 20% 4 hour delays in chicago and dallas only	378	washdc → atlanta [p = 0.6082] atlanta → dallas [p = 0.997369] dallas → phoenix [p = 1]

rate would be 100% and what might seem a reversal of path would be completely eliminated. This would, however, then erroneously eliminate valid paths from consideration.

The final application area presented relates to transportation. In a road network, the driver would like to choose the most-likely shortest path given that any or all roadways may have unexpected congestion or failures with given probability. Using roadway and travel time data, along with associated online data, in part from [14], [11], a roadway map with nominal and congested travel times between cities was developed as shown in Figure 7. The figure shows several city locations that are connected via edges which have multi-weight edge sets. The edges are bi-directional but are only shown as a link for simplicity; there are 25 links so there are 50 directed edges in the graph. The base value for each edge is set to the nominal drive time between the locations, and additional multi-weight values are set in accordance with various congestion models.

Table VI shows some results for various conditions from the graph in Figure 7. The first three examples are for a *miami* to *sanfran* trip, and the last three are for a *washdc* to *phoenix* trip. In each case a short description of the expected conditions, the expected drive time along the most-likely shortest path with certainties at each vertex are shown. Since for cases 1 thriough 5 there are two states for each of the fifty edges, note that there are $2^{50} = 1,125,899,906,842,624 \approx 1.13 \times 10^{15}$ combinatorial states for the graph. For Case 6, there are $3^{18} = 387,420,489 \approx 3.87 \times 10^{8}$ combinatorial states.

For cases #1 and #4, the edge metric for each edge is the nominal (base) drive time value and a second weight that is 1 hour higher, each with a probability of 50%. For example, the edge weight for the *portland* to *seattle* link is nominally 3.308 (hours). So for the one-hour delay model, the multi-weight edge values for this link would be $\{3.308, 4.308\}$, each at 50% probability. The same technique is applied to all other links. For case #2, a similar modeling approach is used, but instead of a one hour delay, there is a 50% chance of a 3 hour delay, so the *portland* to *seattle* edge weight in this case would be $\{3.308, 8.308\}$.

Case #3 applies the one hour delay model to all links, except for links to and from *la* which instead has a 50% probability of a 4 hour delay. Case #5 uses a 50% probability for a 3 hour delay for links into and out of *dallas* and the same 1 hour/50% situation for other links. Finally, for case #6 all links have only their nominal value, except for links into and out of *dallas* and *chicago* which have a 50% chance of encountering the nominal edge weight and a 30% of having a 2 hour delay and a 20% chance of a 4 hour delay. To illustrate the multi-weight edge for one of these links, the nominal drive time for *washdc* to *chicago* is 13.423 hours. In the model for the link the weights would be set as $\{13.423, 15.423, 17.423\}$, with probabilities of 50%, 30%, and 20%, respectively.

With the scenarios defined, a discussion of the results now follows. For each case, the size of the dominant set that fully covers the combinatorial space (discussed earlier) is shown in column 2 of Table VI; not shown is the average size of the Q-set at the point of recursion in Figure 3, but it runs from 60 to 80% for these cases.

In cases #1, #2 and #3, the first step is always from *miami* to *atlanta* with probability 1. This step is obvious from the graph topology. For Case #1, after arriving in *atlanta*, the best option is to *dallas* with a probability of about 50.0%. Via extra internal data inspection, the other option here is a 50% chance of encountering a most-likely shortest path via *stlouis*. Each of the subsequent choices from *dallas* are with probability 1, so for this delay model, there is no chance that alternates from here are better. Now, Case #2 shows what happens when there is a 3 hour delay instead of 1 hour on each link (at 50%). The preferred route now changes to go through *stlouis*. In the final example for the *miami* to *sanfran* cases, Case #3 considers the same situation as Case #1, namely 1 hour delays at a 50% rate for each link, but with the one change that there is instead of a 1 hour delay, there is 4 hour delay with 50% probability in *la*. This also causes the same change to the *stlouis* route from Case #1.

Moving on to the *washdc* to *phoenix* cases, metric settings for Case #4 are the same as in Case #1, that is a 50% chance of a 1 hour delay on each link. In this case, the initial choice of going to *atlanta* is with a 65.6% certainty. The changes in Case #5, which has additional delay in *dallas* causes the best route to change to start with *chicago*, with a rather high certainty of 73.4%. Case #6 has the same most-likely shortest path as Case #4, but with lower confidence in the choices from *washdc* and from *atlanta*.

VII. PSEUDO-POLYNOMIAL SIMPLIFICATION

Each of the members of the dominant set can be found with computational complexity of $O(|E||V|^2)$ but there may in

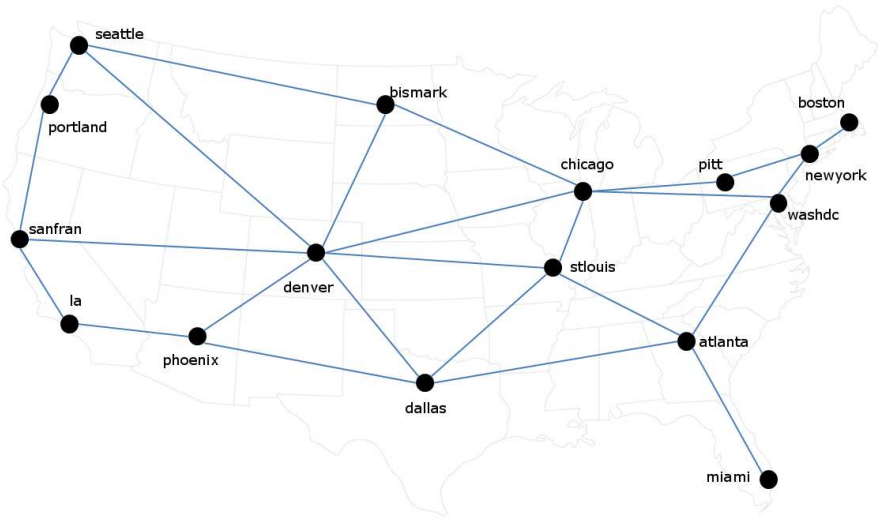

Fig. 7. Roadway map. Actual routes follow the roadway system but this graphic shows connections as straight lines. Links are bidirectional.

the worst-case be an exponential number of them, despite the savings from MSD-SPA. At the risk of not finding all dominant states nor covering the complete combinatorial space, the PRED control feature (refer to Figure 3) can be used to reduce complexity. For example, this predicate can be designed to only permit a fixed number of non-base (lowest value) metrics simultaneously, to halt further searching when distances get too great, or to limit the number of dominant states ($|\mathbf{T}|$) found so far to a fixed total, etc.

In particular, it is not unreasonable to assume that in many practical problems that the lowest value for each edge distance (metric) is the 'normal' condition and that higher values represent abnormal conditions. In a transportation network, the lowest value edge setting would be the normal travel time while the higher values would represent cases of excessive traffic, road construction, total closure (a metric of ∞). A similar argument can be made for communication networks, project planning and other realms. With this in mind, it seems reasonable to consider situations wherein only a given number of out of normal (that is non lowest edge metric values) edge settings are allowed.

For example, if each of the edges has two metrics, and we limit the number of simultaneous non-base values to no more than k, then the cases considered are the base case, the cases where each edge metric is set to its second (non-base) metric, then all the combinations of two non-base metrics, etc. This is expressed Equation 3.

$$\binom{|E|}{0} + \binom{|E|}{1} + \binom{|E|}{2} + \cdots + \binom{|E|}{k} \tag{3}$$

For $k = 2$ this is evaluated as:

$$\begin{aligned}\binom{|E|}{0} + \binom{|E|}{1} + \binom{|E|}{2} &= 1 + |E| + |E|(|E| - 1)/2 \\ &= \frac{|E|^2 + |E| + 2}{2} = O(|E|^2)\end{aligned} \tag{4}$$

Combined with the complexity of the COMPLETE-TREE routine, this would result in an overall complexity of $O(|E|^3|V|^2)$.

TABLE VII
DETERMINING MLSP ROUTES FOR k-LIMITED SITUATIONS.

k	Comb. Size	Dom Set Size	MLSP Route
2	1,225	122	miami → atlanta [p = 1] atlanta → dallas [p = 0.808044] dallas → phoenix [p = 1] phoenix → la [p = 1] la → sanfran [p = 1]
4	230,300	2,046	miami → atlanta [p = 1] atlanta → dallas [p = 0.729431] dallas → phoenix [p = 1] phoenix → la [p = 1] la → sanfran [p = 1]
6	15,890,700	11,383	miami → atlanta [p = 1] atlanta → dallas [p = 0.643707] dallas → phoenix [p = 1] phoenix → la [p = 1] la → sanfran [p = 1]

Similar expressions can be derived for cases where edges have more than two or a varied numbers of metric settings, as well as for other means for setting the PRED control. Of course, MSD-SPA will still reduce the combinatorial space to the extent possible via coverage by the dominant set.

Table VII revisits the data from Case 1 of Table VI while applying a limit on the number of non-base value metrics. With up to and including all combinations of non-base metrics up to $k = 2, 4$ and 6, we see dominant set sizes of 122, 2,046 and 11,383 respectively as compared to 49,728 for all possible combinations in Table VI. The MLSP route found is the same for each of these cases, but the certainty of selection at *atlanta* varies. As the number of combinatorial cases allowed increases (larger k all the way up to unlimited in Table VI), the certainty of the selection is lower. Since the coverage is complete for the cases considered, e.g. all base, one- and two-non base value combinations are included for $k = 2$, this means that one is actually more likely to encounter a shorter path from *atlanta* to

dallas when any two or fewer non-base metrics are allowed. Conversely, this means that fewer shortest paths follow this route when multiple non-base metrics are allowed.

VIII. Concluding Remarks

A new method of finding shortest path distances and trees for multi-state graphs has been defined, along with practical applications. A practical and reasonably efficient solution, MSD-SPA, has been demonstrated for addressing such problems. Although the method is exponential in the worst case, the use of dynamic programming and finding solutions only for the dominant set of graph states lowers typical computational cost. The results of this method were then used to determine the most-likely shortest path (MLSP). A pseudo-polynomial simplification that leverages the idea that only a given number of non-base value metrics might be expected was also shown.

While the shortest reachable distance vector, d, is unique for any particular edge metric setting, the edges comprising the tree used to reach any vertex from s are not necessarily unique. Multiple possible tree solutions will manifest themselves as ties at step 2 (Figure 3) and the method can be extended such that a tree is created for each of these possibilities to enable finding *all* shortest path trees for a particular metric setting. This would be very similar to the modifications to Dijkstra's method made by Shaikh *et al* [24] to find multiple equivalent distance paths for networking quality of service applications. A related problem is the 'k-shortest simple path' problem [12], [29], [6] which can also be solved with MSD-SPA by modifying the input graph (to the original problem) by adding an infinity metric for each edge and then finding the k lowest finite distances from the source to destination in the solution set $\mathbf{T}$. Note, however, that the reverse does not apply. A k-shortest paths algorithm doesn't find solutions to multi-metric edge graphs, it finds the k-shortest paths for graphs with singular edge metric values.

The translation of flow problems to shortest path duals for planar graphs was mentioned in the introduction. Therefore the application of the method to problems such as Braess' paradox [2] may be possible. Also, since a path that is most-likely to follow a shortest path is found, there may be a relation between the most reliable route and the MLSP [19] but that is outside of the present scope. Although they focused on using random measured/sampled distance and path data input, Xing and Zhuo [26] define the absolute and percentile robust shortest path problems as finding paths that minimize the maximum distance or meet a percentile requirement over all cases.

Since the approach here provides analysis for all possible random combinations, it could be used to address many such problems mentioned above, but this is left for future investigation. One might also note that the core concepts for treating graphs with multi-valued edge metrics may also be applicable in other problem domains amenable to greedy solutions such as minimum spanning tree, maximum flow, and bipartite matching.

References

[1] R. E. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[2] Dietrich Braess, Anna Nagurney, and Tina Wakolbinger. On a paradox of traffic planning. *TRANSPORTATION SCIENCE*, 39(4):446–50, November 2005. Translated from the original German: Braess, Dietrich. 1968. Über ein Paradoxon aus der Verkehrsplanung. *Unternehmensforschung* **12** pp 258-68.

[3] Raymond K Cheung. Iterative methods for dynamic stochastic shortest path problems. *Naval Research Logistics (NRL)*, 45(8):769–789, 1998.

[4] Liang Deng and Martin D. F. Wong. An exact algorithm for the statistical shortest path problem. In *Proceedings of the 2006 Asia and South Pacific Design Automation Conference*, ASP-DAC '06, pages 965–970, Piscataway, NJ, USA, 2006. IEEE Press.

[5] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[6] D. Eppstein. Finding the *K* shortest paths. *SIAM Journal of Computing*, 28(2):652–673, 1998.

[7] L. R. Ford Jr. Network flow theory. Technical Report P-923, RAND Corporation, Santa Monica, California, 1956.

[8] H Frank. Shortest paths in probabilistic graphs. *Operations Research*, 17(4):583–599, 1969.

[9] Sam Halabi. *Internet Routing Architectures*. Cisco Press, 2nd edition, 2001.

[10] Randolph W Hall. The fastest path through a network with random time-dependent travel times. *Transportation science*, 20(3):182–188, 1986.

[11] Cambridge Systematics Inc. Traffic congestion and reliability. Technical report, U.S. Department of Transportation / Federal Highway Administration, September 2005.

[12] E. L. Lawler. A procedure for calculating the *K* best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18:401–405, 1972.

[13] Yi-Kuei Lin. Extend the quickest path problem to the system reliability evaluation for a stochastic-flow network. *Computers & Operations Research*, 30(4):567–575, 2003.

[14] William Mallett, Crystal Jones, Joanne Sedor, and Jeffrey Short. Freight performance measurement: Travel time in freight-significant corridors. Technical Report FHWA-HOP-07-071, U.S. Department of Transportation / Federal Highway Administration, December 2006.

[15] Elise Miller-Hooks. Adaptive least-expected time paths in stochastic, time-varying transportation and data networks. *Networks*, 37(1):35–52, 2001.

[16] Elise Miller-Hooks and Hani Mahmassani. Path comparisons for a priori and time-adaptive decisions in stochastic, time-varying networks. *European Journal of Operational Research*, 146(1):67–82, 2003.

[17] Pitu B Mirchandani. Shortest distance and reliability of probabilistic networks. *Computers & Operations Research*, 3(4):347–355, 1976.

[18] Srihari Nelakuditi, Sanghwan Lee, Yinzhe Yu, Zhi-Li Zhang, and Chen-Nee Chuah. Fast local rerouting for handling transient link failures. *IEEE/ACM Trans. Networking*, 15(2):359–372, April 2007.

[19] Yu Marco Nie and Xing Wu. Shortest path problem considering on-time arrival probability. *Transportation Research Part B: Methodological*, 43(6):597–613, 2009.

[20] Yiyong Pan, Lu Sun, and Minli Ge. Finding reliable shortest path in stochastic time-dependent network. *Procedia-Social and Behavioral Sciences*, 96:451–460, 2013.

[21] George H Polychronopoulos and John N Tsitsiklis. Stochastic shortest path problems with recourse. *Networks*, 27(2):133–143, 1996.

[22] David L Rhodes. Efficient routing in ad hoc networks with directional antennas. In *Military Communications Conference, 2004. MILCOM 2004. 2004 IEEE*, volume 2, pages 1003–1009. IEEE, 2004.

[23] Aman Shaikh and Albert Greenberg. Experience in black-box OSPF measurement. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, IMW '01, pages 113–125, New York, NY, USA, 2001. ACM.

[24] Anees Shaikh, Jenifer Rexford, and Kang Shin. Efficient precomputation of quality-of-service routes. In *Proc. IEEE Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 15–27. IEEE, July 1998.

[25] W.Y. Szeto and S.C. Wong. Dynamic traffic assignment: model classifications and recent advances in travel choice principles. *Central European Journal of Engineering*, 2(1):1–18, 2012.

[26] Tao Xing and Xuesong Zhou. Reformulation and solution algorithms for absolute and percentile robust shortest path problems. *IEEE Transactions on Intelligent Transportation Systems*, 14(2):943–954, 2013.

[27] Tianzhang Xing and Xiaoxin Zhou. Reformulation and solution algorithms for absolute and percentile robust shortest path problems. *Intelligent Transportation Systems, IEEE Transactions on*, 14(2):943–954, 2013.

[28] Wei-Chang Yeh, Wei-Wen Chang, and Chuan-Wei Chiu. A simple method for the multi-state quickest path flow network reliability problem. In *Reliability, Maintainability and Safety, 2009. ICRMS 2009. 8th International Conference on*, pages 108–110. IEEE, 2009.

[29] J. Y. Yen. Another algorithm for finding the K shortest loopless network paths. In *41st Management Operations Research Society of America*, volume 20, 1972.

[30] Dan Zhao, Xiaofeng Hu, and Chunqing Wu. A study on the impact of multiple failures on OSPF convergence. *Int. J. Hybrid Information Technology*, 6(3):65–73, May 2013.