Optimal Configuration of OSPF Aggregates

Yuri Breitbart **Minos Garofalakis** **Amit Kumar*** **R. Rastogi**

Bell Laboratories
Lucent Technologies
600 Mountain Avenue
Murray Hill, NJ 07974
{yuri,minos,amitk,rastogi}@research.bell-labs.com

February 20, 2001

Abstract

Open Shortest Path First (OSPF) is a popular protocol for routing within an Autonomous System (AS) domain. In order to scale for large networks containing hundreds and thousands of subnets, OSPF supports a two-level hierarchical routing scheme through the use of *OSPF areas*. Each area consists of a set of interconnected subnets and traffic across areas is handled by routers attached to two or more areas, known as *Area Border Routers (ABRs)*. OSPF ABRs are typically configured to aggregate the subnet addresses in their areas and to advertise these aggregates in the remainder of the network instead of individual subnet addresses. Address aggregation within areas is a crucial requirement for scaling OSPF to large AS domains, as it results in significant reductions in routing table sizes, smaller link-state databases, and less network traffic to synchronize the router link-state databases. On the other hand, address aggregation also implies loss of information about the length of the shortest path from the ABR to each subnet; typically, an ABR associates each advertised aggregate with a single *weight* that is common for all subnets covered by the aggregate. This, in turn, can lead to the selection of significantly suboptimal OSPF routing paths between source-destination subnet pairs that span different areas.

In this paper, we address the important practical problem of configuring OSPF aggregates to minimize the error in OSPF shortest path computations due to subnet aggregation. We first develop an optimal dynamic programming algorithm that, given an upper bound k on the number of aggregates to be advertised by the ABRs and a weight-assignment function for the aggregates, computes the k aggregates that result in the minimum cumulative/maximum error in the shortest path computations for all source-destination subnet pairs. Subsequently, we tackle the problem of assigning weights to OSPF aggregates such that the cumulative/maximum error in the computed shortest paths is minimized. We demonstrate that, while for certain special cases (e.g., unweighted cumulative error) efficient optimal algorithms for the weight-assignment problem can be devised, the general problem itself is $\mathcal{NP}$-hard. Consequently, we have to rely on search heuristics to solve the weight-assignment problem: We propose a randomized search strategy for the general case of weighted cumulative error, and an optimal pseudo-polynomial time algorithm for the maximum error case. To the best of our knowledge, our work is the first to address the algorithmic issues underlying the configuration of OSPF aggregates and to propose efficient configuration algorithms that are *provably optimal* for many practical scenarios.

1 Introduction

Open Shortest Path First (OSPF) is a widely-used protocol for routing within an Autonomous System (AS) domain

*Permanent address: Cornell University, Computer Sciences Dept., Ithaca, NY. The work was done while the author was visiting Bell Laboratories.

in today's Internet [FT00, Hui00, Moy98]. To scale for large AS networks, OSPF implements a two-level hierarchical routing scheme through the deployment of *OSPF areas*. Each OSPF area comprises a collection of subnets interconnected by routers. Detailed information about links and subnets within an OSPF area is flooded throughout the elements connected to the area. As a result, every router knows the exact topology of its enclosing OSPF area – this includes the subnets and the links connecting routers within the area. On the other hand, details of an area's topology are not advertised beyond the area's borders and are thus hidden from other areas in the same AS. Instead, subnet addresses within each area are grouped into *aggregates* and only these aggregates are flooded into the rest of the network (thus making an area's subnets reachable from the remainder of the AS network). This task of advertising aggregate information about subnets in an area is carried out by *Area Border Routers (ABRs)*, that is, routers attached to two or more areas.

OSPF areas and address aggregation are crucial in enabling OSPF to scale for AS domains comprising hundreds or thousands of subnets; specifically, they play an important role in optimizing router and network resource consumption, as explained below.

1. **Router Memory.** For OSPF areas *not* directly connected to a router in the AS, the router's routing tables only need to contain entries corresponding to subnet aggregates rather than individual subnet addresses. In other words, a router stores individual subnet addresses in its routing table only for the OSPF areas that are directly connected to it. This obviously leads to smaller routing table sizes and, thus, lower memory requirements at routers.

2. **Router Processing Cycles.** The link-state database maintained at each router is much smaller, since it only needs to include summary information for subnets belonging to OSPF areas not directly connected to the router. Consequently, the computational cost of the shortest-path calculation decreases substantially.

3. **Network Bandwidth.** For subnets within each OSPF area, only aggregate address information (rather than individual subnet addresses) is flooded into the rest of the AS network. As a result, the volume of OSPF flooding traffic necessary to synchronize the link-state databases of the AS routers is significantly reduced.

Nevertheless, despite its obvious benefits, OSPF address aggregation involves important practical tradeoffs. This is because address aggregation typically results in loss of information which, in turn, can lead to *suboptimal* routing paths. To see this, we need to delve in more depth into how OSPF routing works in the presence of address aggregation. Briefly, each ABR attaches a *weight* to each aggregate that it advertises to the rest of the network. This weight is critical in determining the path used by a router external to the area to reach subnets covered by the aggregate. More specifically, among all the ABRs advertising the aggregate (with possibly different weights), an external router chooses the ABR (say b) that minimizes the sum of the following two quantities: (1) the length of the shortest path from the external router to the border router b; and, (2) the weight advertised by b for the aggregate. Once such an ABR b is chosen, IP packets from the external router to every subnet covered by the aggregate are forwarded along the shortest path from the external router to b and, subsequently, along the shortest path from b to the subnet. However, for certain subnet(s) covered by the aggregate, this path may be significantly suboptimal, since there can be a much shorter path from the external router to the subnet through a different ABR. This is illustrated in the following example.

Example 1.1 Consider the AS network consisting of the four areas 0.0.0.0, 0.0.0.1, 0.0.0.2, and 0.0.0.3 shown in Figure 1. (Area 0.0.0.0 corresponds to the AS's *backbone area* that interconnects the ABRs of the different OSPF areas in the AS.) The boxes in the figure are routers, while the filled black thin rectangles denote subnets. The figure also illustrates the various subnet addresses and the weight of each link connecting a pair of routers. ABR b_1 belongs to area 0.0.0.1, b_2 to area 0.0.0.2, and b_3 and b_4 to area 0.0.0.3. The subnet addresses in area 0.0.0.3 can be aggregated to different degrees. For instance, the aggregate 10.1.0.0/21 covers all the subnets in the area. In contrast,

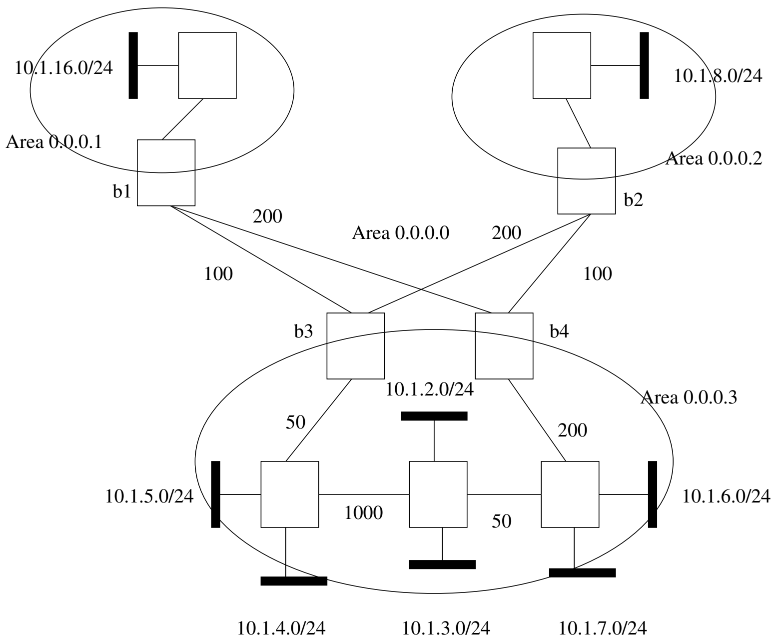


Figure 1: Example of suboptimal routing due to address aggregation.

10.1.4.0/22 covers subnets 10.1.4.0/24, 10.1.5.0/24, 10.1.6.0/24 and 10.1.7.0/24, while 10.1.2.0/23 covers subnets 10.1.2.0/24 and 10.1.3.0/24.

Suppose one of the aggregate addresses advertised by the ABRs of area 0.0.0.3 is 10.1.4.0/22. Suppose further that each ABR assigns a weight to the aggregate that equals the distance of the furthest component subnet in the aggregate from the router (as suggested, for example, by Moy [Moy98]). Thus, router b_3 advertises 10.1.4.0/22 with a weight of 1100 (distance of subnet 10.1.6.0/24 from b_3), while router b_4 advertises 10.1.4.0/22 with a weight of 1250 (distance of subnet 10.1.4.0/24 from b_4). Thus, external router b_1 belonging to area 0.0.0.1 forwards all packets to subnets in 10.1.4.0/22 through border router b_3, since the shortest path to the aggregate through b_3 has a length of $100 + 1100 = 1200$, while the shortest path through b_4 has length $200 + 1250 = 1450$. Note, however, that the path from b_1 to subnets 10.1.6.0/24 and 10.1.7.0/24 passing through router b_3 has length 1200 and is suboptimal since the shortest path from b_1 to both subnets is through border router b_4 and its length is only 400. Thus, even in this simple scenario, address aggregation results in an error of $1200 - 400 = 800$ in the optimal route (i.e., shortest path) computation between b_1 and each of the subnets 10.1.6.0/24 and 10.1.7.0/24.

Further, note that considering different weight assignments for the aggregates does not alleviate the problem. The root of the problem is that a single border router is selected by b_1 for reaching all subnets in 10.1.4.0/22. If b_4 is chosen instead of b_3, then the paths from b_1 to subnets 10.1.4.0/24 and 10.1.5.0/24 through b_4 become much longer (their length is $200 + 1250 = 1450$) compared to the shortest paths to the subnets that pass through b_3 (whose length is $100 + 50 = 150$). ∎

The primary reason for suboptimal paths being selected when subnets are aggregated is that a single weight is used by each ABR for all the subnets covered by the aggregate; obviously, a single weight may be incapable of accurately capturing the distance of the border router to every covered subnet. This problem is exacerbated when the aggregated subnets are spread across the area with some subnets being in close proximity to distinct border routers in the area. This was precisely the case in Example 1.1, where aggregate 10.1.4.0/22 spans two subnets: 10.1.5.0/24 (close to b_3) and 10.1.6.0/24 (close to b_4), the distance between which is greater than 1000. The end result is that the single weight advertised for aggregate 10.1.4.0/22 is not representative of the true distance between any border router (either b_3 or b_4) and the two subnets in 10.1.4.0/22.

One possible way to reduce the error in suboptimal OSPF routing paths in the presence of aggregation is to avoid aggregating distant subnets that are close to multiple border routers. Thus, in Example 1.1, instead of advertising the single aggregate 10.1.4.0/22, one can choose to advertise two aggregates 10.1.4.0/23 (with weights 50 and 1250 at b_3 and b_4, respectively) and 10.1.6.0/23 (with weights 1100 and 200 at b_3 and b_4, respectively). This clearly reduces the error in the selected paths to zero, since the assigned weights capture the ABRs' distances to the aggregated subnets *exactly*. Thus, there is an important tradeoff between the number of aggregates advertised (and, consequently, the size of the routing tables) and the error in the selected shortest paths. This tradeoff is further complicated by the fact that the aggregates advertised by OSPF border routers do not have to be disjoint – it is entirely possible for one advertised aggregate to be *completely contained* in another. In such a scenario, the *longest match property* of IP routing causes the more specific aggregate to take precedence for route computation to subnets within the aggregate. Also, configuring the *weight assignments* used by ABRs for address aggregates is another important mechanism for controlling the quality of the OSPF routing paths in the presence of aggregation. In the following example, we illustrate how, by carefully selecting the address aggregates as well as the associated weights, all the subnets in area 0.0.0.3 (Figure 1) can be advertised using only two (overlapping) aggregates while incurring zero error in shortest-path computations.

Example 1.2 Consider the AS depicted in Figure 1. One way to ensure that the error in the selected paths to area 0.0.0.3's subnets is zero is to have the ABRs advertise the following three aggregates: 10.1.4.0/23, 10.1.2.0/23 and 10.1.6.0/23. The reason for this is that it is possible to choose weights for each aggregate at each ABR such that the weight equals the exact distance between the border router and every subnet covered by the aggregate. For instance, for aggregate 10.1.2.0/23, weights of 1050 and 250 at ABRs b_3 and b_4, respectively, reflect the exact distances of the border routers to subnets in it.

Achieving zero error with only two aggregates is more challenging. Note that all the subnets in area 0.0.0.3 can be covered by the single aggregate 10.1.0.0/21; however, this aggregate by itself, cannot result in zero error. Another possibility is to consider the two disjoint aggregates 10.1.4.0/22 and 10.1.2.0/23 which cover all the subnets. However, as we saw earlier, since subnets 10.1.4.0/23 and 10.1.6.0/23 covered by 10.1.4.0/22 are closer to different routers (and distant from each other), this cannot result in zero error either. Thus, the key to optimizing the error is to bundle 10.1.4.0/23 into one aggregate, and 10.1.2.0/23 and 10.1.6.0/23 into the other. It turns out that this can be achieved by advertising the following two aggregates: 10.1.0.0/21 and 10.1.4.0/23. The longest match characteristic of IP routing causes the latter aggregate to be used for routing to subnets in 10.1.4.0/23 and the former to be used to route to subnets in 10.1.2.0/23 and 10.1.6.0/23.

One question still remains: what weights should assigned to each aggregate? While this is straightforward for the aggregate 10.1.4.0/23 (since the two subnets 10.1.4.0/24 and 10.1.5.0/24 covered by it are at the same distance from any given border router), it is somewhat less obvious for the aggregate 10.1.0.0/21. Simply setting the weight equal to the distance of the ABR to the most distant covered subnet (see, e.g., [Moy98]) may not result in the least error. To see this, suppose b_3 and b_4 advertise 10.1.0.0/21 with weights 1100 and 1250, respectively (i.e., the maximum distance to a subnet contained in the aggregate). This causes b_1 to select b_3 for subnets in 10.1.0.0/21, and the resulting cumulative error in the selected paths from b_1 to all subnets in area 0.0.0.3 is $2 * (150 - 150) + 2 * (1150 - 450) + 2 * (1200 - 400) = 3000$. On the other hand, a lower cumulative error can be achieved if b_3 and b_4 advertise 10.1.0.0/21 with weights 730 and 570, respectively (i.e., the average distance to the subnets contained in the aggregate). In this case, b_1 selects border router b_4 to access subnets in 10.1.0.0/21, resulting in a lower cumulative error of $2 * (1450 - 150) + 2 * (450 - 450) + 2 * (400 - 400) = 2600$ (assuming again that only 10.1.0.0/21 is advertised for area 0.0.0.3).

Further, configuring border routers b_3 and b_4 to advertise aggregates 10.1.0.0/21 and 10.1.4.0/23, b_3 with weights 730 and 50, and b_4 with weights 570 and 1250, causes the cumulative error for b_1 to reduce to zero. This is because b_1 selects ABR b_3 for subnets in 10.1.4.0/23 and ABR b_4 for the remaining subnets (that is, subnets in 10.1.0.0/21 but not contained in 10.1.4.0/23). Thus, the selected paths after aggregation are indeed the shortest paths from b_1 to the subnets in area 0.0.0.3. ∎

Our Contributions. In this paper, we address the important practical problem of configuring OSPF aggregates to minimize the error in OSPF shortest path computations due to address aggregation. From our discussion above, we can see that OSPF aggregate configuration involves two key subproblems: (1) selecting the aggregates to advertise at each ABR, and (2) assigning weights to each advertised aggregate at each ABR. We address each of these two problems separately. We first develop an *optimal* dynamic programming algorithm that, given an upper bound k on the number of aggregates to be advertised by the ABRs and a weight-assignment function for the aggregates, computes the k aggregates that result in the minimum cumulative/maximum error in the OSPF shortest path computations for all source-destination subnet pairs. This problem is obviously relevant when there is a limit on the number of aggregates that can be advertised within an AS in order to bound the routing table sizes, number of entries in the link-state database, or the amount of network traffic due to OSPF advertisements. The objective then is to choose the k aggregates to advertise such that the selected paths are as close to the shortest paths as possible (where "closeness" is measured in terms of either the total or the maximum over all source-destination subnet pairs in the AS). Furthermore, our proposed algorithm can be easily extended to optimally solve the dual OSPF configuration problem, where the goal is to compute the minimum number of aggregates so that the (cumulative or maximum) error in selected paths is less than a certain user-specified threshold.

We then address our second subproblem of selecting weights for OSPF aggregates at each ABR such that the deviation of selected paths from the shortest paths is minimized. More specifically, we attack the following problem: Given an address aggregate x, determine an assignment of weights to x at each ABR in its area such that the (cumulative or maximum) error in the selected paths between source-destination subnet pairs is minimized. We demonstrate that, while for certain restricted (but interesting) cases the above problem can be solved in polynomial time, the general problem itself is $\mathcal{NP}$-hard. Consequently, we have to rely on search heuristics to solve the weight-assignment problem. We also propose a randomized search strategy for the general case of weighted cumulative error, and an optimal pseudo-polynomial time algorithm for the maximum error case.

Combining our algorithmic results for the two subproblems provides us with an effective, integrated solution for configuring (in many cases, optimally) OSPF aggregates in an AS domain. The idea is to first apply our weight-assignment algorithms to determine "good" weights for all candidate aggregates at each ABR, and then employ our dynamic programming algorithm to select the optimal subset of aggregates to advertise (given these weight assignments). To the best of our knowledge, our work is the first to address the algorithmic issues underlying the configuration of OSPF aggregates and to propose efficient configuration algorithms that are *provably optimal* for many practical scenarios.

2 System Model and Problem Formulation

System Model and Notation. We model the network as an undirected graph. Nodes in the graph correspond to either routers or subnets. Edges in the graph connect routers with other routers or subnets. A link exists between two routers if the two routers can directly exchange IP packets without going through an intermediate router (that is, the two routers are either connected to the same subnet or are connected by a point-to-point link). A link exists between a subnet and a router if the subnet is connected to the router. Each link has an associated weight, which is the OSPF weight assigned to the link (that is, the interface to which the link is connected to). For simplicity, we assume that the link is assigned the same weight at both ends – our algorithms, however, are applicable even if link weights are not symmetric. Table 1 describes the notation employed in the paper.

The set of subnets $\mathcal{S}$ in the network are partitioned into disjoint areas. The set of areas is denoted by $\mathcal{R}$ and the set of subnets in area $R_i \in \mathcal{R}$ is denoted by S_i. A router is said to be attached to area R_i if it is directly connected

Symbol	Description
s, t	Source, destination subnets
$\mathcal{R}$	Set of OSPF areas
R_i	Generic term for OSPF area
$\mathcal{S}$	Set of all subnets in autonomous system domain
S_i	Set of subnets in area R_i
$\mathcal{B}$	Set of all ABRs
B_i	Set of ABRs for area R_i
b, c	Generic letters for an ABR
$\mathcal{A}$	Set of aggregates eligible for advertising
A_i	Set of aggregates eligible for advertising in area R_i
x, y	Generic advertised address aggregates
X, Y	Generic sets of advertised address aggregates
$D(s, t)$	Degree of importance of the source-destination subnet pair (s, t)
W_X	Weight assignment function for aggregates in X
$lsp(s, t)$	Length of shortest path between subnets s and t
$lsp(s, t, X, W_X)$	Length of shortest path between subnets s and t when aggregates in X are advertised with weights in W_X

Table 1: Notation used in the paper.

to a subnet in S_i. A router that is attached to two or more areas is called an *area border router* (ABR). We denote by B_i the set of ABRs attached to area R_i. In addition to area R_i (and possibly other areas), every ABR in B_i is also attached to a special *backbone* area. The backbone area serves to connect the subnets in the various other areas. We denote by $lsp(s, t)$ the length of the shortest path between s and t, where s and t can be subnets or routers within the AS. Note that, if s and t belong to the same area R_i, then the shortest path between s and t is defined to be over links in area R_i. If, instead, s and t belong to distinct areas (say, R_i and R_j respectively), then the shortest path between s and t involves two ABRs $b \in B_i$ and $c \in B_j$, and consists of three path segments: the first is the shortest path between s and b involving links in R_i, the second is the shortest path between b and c over links in the backbone area and the final segment is the shortest path between c and t all of whose links belong to area R_j. Note that lsp can be defined in a similar fashion if either of the subnets s and t above are replaced by routers.

In OSPF, information relating to links and subnets in an area are flooded throughout the area. Consequently, routers attached to area R_i have detailed knowledge of R_i's topology. As a result, IP packets originating in any subnet s belonging to area R_i destined to a subnet t in the same area are forwarded along the shortest path between s and t. However, in order to ensure smaller routing table sizes and reduce network traffic overhead, detailed information about individual subnets within an area are typically not be advertised beyond the area's borders. Instead, area R_i's ABRs will typically be configured to advertise a set of aggregates X that cover subnets in S_i and a separate weight for each aggregate in X. We denote by $W_X(x, b)$, the weight assigned to an aggregate $x \in X$ by ABR $b \in B_i$. Each ABR in B_i floods in the entire backbone area, every aggregate $x \in X$ along with the weight assigned to x by it – this causes ABRs belonging to every other area to receive x. An ABR $c \in B_j$, in turn, floods x into area R_j with an adjusted weight equal to $\min_{b \in B_i}\{lsp(b, c) + W_X(x, b)\}$. Thus, a subnet s in S_j, in order to reach aggregate x that covers subnets in S_i selects a path passing through ABR $b \in B_i$ for which $lsp(s, b) + W_X(x, b)$ is minimum.

Due to the longest match property of IP routing, the most specific aggregate covering a subnet determines the path to the subnet. We say that an aggregate x is more specific than an aggregate y if x is contained in y, which we denote by $x \in y$. Thus, for a subnet t in S_i, if x is the most specific aggregate in X that covers t, then a subnet s in S_j in order to reach t, selects the path comprising of the shortest path from s to b and then from b to t, where $b \in B_i$ is the ABR for which $lsp(s, b) + W_X(x, b)$ is minimum. We denote the length of this selected path from s to t for the set of advertised aggregates X and weight assignment W_X by $lsp(s, t, X, W_X)$. Thus, $lsp(s, t, X, W_X) = lsp(s, b) + lsp(b, t)$, and the error in the selected path is simply $lsp(s, t, X, W_X) - lsp(s, t)$. When s and t belong to the same area, we define $lsp(s,t,X,W_X)$ to be equal to $lsp(s,t)$. Note that $lsp(s,t,X,W_X) = \infty$ if X does not contain an aggregate that covers t (the implication here is that t is unreachable from s).

Problem Statement. We address the problem of computing the set of aggregates X advertised across all the areas in an AS and the weight assignment function W_X such that the error in the selected paths is minimized. Clearly, we need to impose certain restrictions on X and W_X in order to ensure the reachability of remote subnets in a different area. First, we require that X be *complete*, that is, every subnet in $\mathcal{S}$ be covered by some aggregate in X. The next two restrictions serve to ensure that an ABR cannot advertise an aggregate covering a subnet in S_i unless it belongs to B_i. We say that an aggregate x is *eligible* if all the subnets in $\mathcal{S}$ covered by it belong to a single area. Thus, in the network of Figure 1, aggregate 10.1.0.0/21 is eligible since it only covers subnets in Area 0.0.0.3; however, aggregate 10.1.0.0/20 is not since it covers subnets 10.1.8.0/24 and 10.1.4.0/24 that belong to different areas. Let $\mathcal{A}$ denote the set of all eligible aggregates such that every aggregate in $\mathcal{A}$ covers at least one subnet in $\mathcal{S}$. Note that $\mathcal{S} \subseteq \mathcal{A}$. Further, let $A_i \subseteq \mathcal{A}$ denote the set of eligible aggregates that cover subnets in S_i. We require that the set of advertised aggregates $X \subseteq \mathcal{A}$. We also require that only ABRs in B_i advertise aggregates in A_i. One way to model this is by requiring that $W_X(x,b) = \infty$ if $x \in A_i$ and $b \notin B_i$.

We are now in a position to state the two basic problems tackled in the remainder of this paper.

1. **Aggregate Selection Problem:** Given a k and a weight assignment function $W_{\mathcal{A}}$, compute a complete set $X \subseteq \mathcal{A}$ containing at most k aggregates such that $\sum_{s,t\in\mathcal{S}}(lsp(s,t,X,W_{\mathcal{A}}) - lsp(s,t))$ is minimum.

2. **Weight Selection Problem:** For an aggregate $x \in A_i$, compute a weight assignment function $W_{\{x\}}$ such that $\sum_{s,t\in\mathcal{S},t\in x}(lsp(s,t,\{x\},W_{\{x\}}) - lsp(s,t))$ is minimum.

Note that, in both problem statements above every source-destination subnet pair is assigned the same degree of importance. In other words, in the final error, the error in the selected path between every subnet pair is treated equally, that is, given an equal degree of importance. However, this is somewhat restrictive since minimizing the error in the selected paths for certain source-destination subnets may be more important. This may happen, for instance, for subnet pairs between which there is a disproportionately high volume of traffic, or subnet pairs carrying high-priority or delay-sensitive traffic like voice. Thus, we can consider a *degree of importance* function D which for a pair of subnets s,t returns a real value $D(s,t)$ that reflects the importance of minimizing the error in the selected path between subnets s and t. Note that $D(s,t)$ can be any arbitrary function of the volume/priority of traffic flowing between subnets s and t. Subnet pairs for whom the error in the selected path does not matter (either due to very low traffic volume or due to low priority data traffic) can be assigned low values for $D(s,t)$ or even zero. The generalized aggregate and weight selection problems incorporating the degrees of importance are then as follows.

1. **Generalized Aggregate Selection Problem:** Given a k and a weight assignment function $W_{\mathcal{A}}$, compute a complete set $X \subseteq \mathcal{A}$ containing at most k aggregates such that $\sum_{s,t\in\mathcal{S}} D(s,t) * (lsp(s,t,X,W_{\mathcal{A}}) - lsp(s,t))$ is minimum.

2. **Generalized Weight Selection Problem:** For an aggregate $x \in A_i$, compute a weight assignment function $W_{\{x\}}$ such that $\sum_{s,t\in\mathcal{S},t\in x} D(s,t) * (lsp(s,t,\{x\},W_{\{x\}}) - lsp(s,t))$ is minimum.

In the all problems outlined above, our goal is to minimize the (weighted) cumulative error across all source-destination subnet pairs. An alternative statement of the OSPF configuration problem (also considered in this paper) aims to minimize the *maximum* error across all the source-destination subnets. The corresponding aggregate and weight selection problems can be formulated in a similar fashion, except that instead of minimizing $\sum_{s,t\in\mathcal{S}}()$, the objective is to minimize $\max_{s,t\in\mathcal{S}}()$.

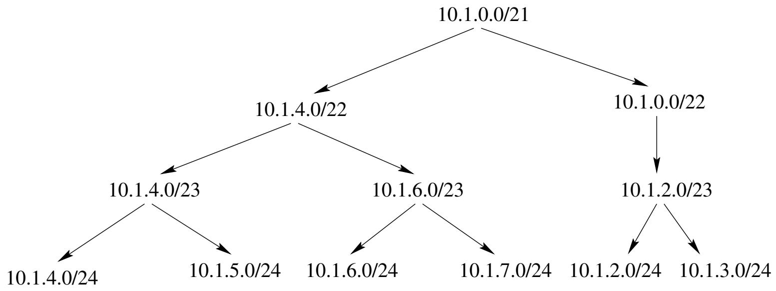


Figure 2: Aggregate tree for eligible aggregates covering subnets in area 0.0.0.3.

3 Generalized Aggregate Selection Problem

In this section, we present a dynamic programming algorithm for the generalized aggregate selection problem. Our algorithm exploits the fact that the containment structure of aggregates in $\mathcal{A}$ is a set of trees (termed *aggregate trees*). We define the notion of error for each aggregate tree when certain aggregates in it are selected, and demonstrate that the cumulative error in the shortest-path computation when a subset X of aggregates is advertised is equal to the sum of the corresponding aggregate-tree errors when X is selected. We present our dynamic programming algorithm for selecting k aggregates in a single aggregate tree that minimize the tree's error in Section 3.2. Section 3.3 then presents the algorithm for combining the results for the collection of aggregate trees to derive the final k aggregates that yield the minimum overall error. Finally, in Section 3.5, we show how our algorithms can be extended for minimizing the maximum error in the OSPF routing paths.

3.1 Aggregate Trees

The containment relationship among the eligible aggregates in $\mathcal{A}$ naturally form a set of trees. For aggregates $x, y \in \mathcal{A}$, an edge from x to y exists if x covers y and every other aggregate $z \in \mathcal{A}$ that covers y also covers x. Figure 2 illustrates the aggregate tree for the eligible aggregates that cover subnets in Area 0.0.0.3 (from Figure 1). Observe that since $\mathcal{A}$ contains aggregates 10.1.4.0/22, 10.1.6.0/23 and 10.1.7.0/24, there is an edge from 10.1.6.0/23 to 10.1.7.0/24; however, there is no edge from 10.1.4.0/22 to 10.1.7.0/24 in the tree. Also, the internal nodes in the aggregate tree have either one or two children, but no more than two children. For instance, in Figure 2, 10.1.0.0/22 has only one child since 10.1.0.0/23 does belong to $\mathcal{A}$ (subnet 10.1.1.0/24 does not exist in the network). Note that each leaf of an aggregate tree is a subnet in $\mathcal{S}$. Further, the root $r(T)$ of tree T is basically an aggregate that is not covered by any other eligible aggregate.

We next define the error of a tree T when a set of aggregates in it have been selected. Suppose x is an aggregate in the tree T, y is the most specific selected aggregate covering x in the tree, and X is the set of selected aggregates. Then the error of the subtree rooted at x is recursively defined as follows:

$$E(x,y,X,W_{\mathcal{A}}) = \begin{cases} \sum_{(x,u)\in T} E(u,y,X,W_{\mathcal{A}}) & \text{if } x \text{ has children, and } x \notin X \\ \sum_{(x,u)\in T} E(u,x,X,W_{\mathcal{A}}) & \text{if } x \text{ has children, and } x \in X \\ \sum_{s\in\mathcal{S}} D(s,x) * (lsp(s,x,\{y\},W_{\mathcal{A}}) - lsp(s,x)) & \text{if } x \text{ is a leaf and } x \notin X \\ \sum_{s\in\mathcal{S}} D(s,x) * (lsp(s,x,\{x\},W_{\mathcal{A}}) - lsp(s,x)) & \text{if } x \text{ is a leaf and } x \in X \end{cases}$$

The error for an entire tree T with the set of selected aggregates X is then simply $E(r(T),\epsilon,X,W_{\mathcal{A}})$ (ϵ denotes the empty aggregate that does not cover any other aggregate). Note that each recursive invocation of E on x's children propagates x as the most specific selected aggregate if $x \in X$. Consequently, whenever E is invoked for the subtree rooted at an aggregate x, y is always the most specific selected aggregate covering x. As a result, the error of a tree is simply the sum of the errors of all the leaf subnets in it, where the error of a subnet t is

$\sum_{s \in \mathcal{S}} D(s,t) * (lsp(s,t,\{y\},W_{\mathcal{A}}) - lsp(s,t))$, where y is the most specific aggregate in X that covers t. Thus, since every subnet in $\mathcal{S}$ is contained in one of the trees, the sum of errors of all the trees is essentially the cumulative error in the selected paths, which is the metric we are interested in minimizing.

Thus, we have reduced the aggregate selection problem to the problem of computing a set X containing at most k aggregates such that the sum of the errors of all the trees is minimum. We break this into two subproblems which we address in the following two subsections. First, we present our dynamic programming algorithm to compute the optimal subset of aggregates for a single aggregate tree. Then, we show how to select a combination of aggregates from different trees (i.e., OSPF areas) that minimize the overall error for the entire collection of aggregate trees.

3.2 Computing Optimal Aggregates for a Single Tree

We begin by presenting below, a set of recursive equations for computing a tight lower bound on the error of a tree assuming that at most k arbitrary aggregates in the tree can be selected. The equations form the basis of our dynamic programming algorithm and can be used to compute the k best aggregates to select in order to minimize the error of the tree. Suppose that x is an aggregate in the tree T and y is the most specific aggregate in the tree covering x that has already been selected. Then, the minimum error of the subtree rooted at x if we are allowed to choose at most k aggregates in the subtree, is as follows.

$$minE(x,y,k,W_{\mathcal{A}}) = \begin{cases} E(x,y,\emptyset,W_{\mathcal{A}}) & \text{if } k=0 \\ \min\{minE(u,y,k,W_{\mathcal{A}}), minE(u,x,k-1,W_{\mathcal{A}})\} & \\ & \text{if } k>0 \text{ and } x \text{ has a single child } u \\ \min\{\min_{0 \le i \le k}\{minE(u,y,i,W_{\mathcal{A}}) + minE(v,y,k-i,W_{\mathcal{A}})\}, & \\ \qquad \min_{0 \le i \le k-1}\{minE(u,x,i,W_{\mathcal{A}}) + minE(v,x,k-1-i,W_{\mathcal{A}})\}\} & \\ & \text{if } k>0 \text{ and } x \text{ has children } u,v \\ \min\{E(x,y,\emptyset,W_{\mathcal{A}}), E(x,y,\{x\},W_{\mathcal{A}})\} & \text{if } k>0 \text{ and } x \text{ is a leaf} \end{cases}$$

The intuition underlying the above set of equations is that if $k=0$, then since no aggregates in the subtree can be selected, the minimum error is simply the error of the subtree when no aggregates in it are chosen. In case $k>0$ and x has children, and if X is the set of aggregates in the subtree rooted at x that if selected result in the minimum error, then the following hold for X: (1) either $x \in X$ or $x \notin X$, and (2) of the remaining aggregates in X, i are in the subtree rooted at its left child and the remaining $k-i$ or $k-i-1$ aggregates (depending on whether $x \in X$) are in the subtree rooted at its right child. Thus, since the error of the subtree with x as root is simply the sum of the errors of its left and right subtrees, we can compute the minimum error for (the subtree rooted at) x by first computing the minimum error of its left and right subtrees for $0 \le i \le k$ selected aggregates and for the cases when x is either selected or not selected, and then choosing the combination with the smallest error. Finally, if $k>0$ and x is a leaf, then there are only two possible alternatives for selecting aggregates in x's subtree: either to select x or not to select x. The minimum error for these two cases then yields the desired minimum error. In the following, we formally prove that $minE$ is indeed the minimum error of the subtree rooted at x if at most k aggregates can be selected in it.

Theorem 3.1 $minE(x,y,k,W_{\mathcal{A}})$ is equal to the minimum of $E(x,y,X,W_{\mathcal{A}})$, where X is any arbitrary set containing at most k aggregates in the subtree rooted at x. ∎

Proof: The proof is by induction on the number of aggregates in the subtree rooted at x.
Basis: Suppose that there is only one aggregate in the subtree rooted at x, that is, x is a leaf. In case $k=0$, then the minimum error of the subtree is simply $E(x,y,\emptyset,W_{\mathcal{A}})$ since none of the aggregates in the subtree can be selected. On the other hand, if $k>0$, then the two possibilities are that either $x \in X$ or $x \notin X$ and the minimum error is the minimum of the error for these two cases.

Induction Step: Suppose that X is the set containing at most k aggregates in the subtree rooted at x that minimizes the error of the subtree. We show that $minE(x, y, k, W_{\mathcal{A}})$ is equal to $E(x, y, X, W_{\mathcal{A}})$ for one case (other cases can be handled in a similar fashion). The case we consider is when $x \in X$ and has two children u and v with U and V denoting the aggregates selected in the subtrees rooted at u and v, respectively. Note that if $|U| = i$, then $|V| \le k - i - 1$. From the definition of error, $E(x, y, X, W_{\mathcal{A}}) = E(u, x, U, W_{\mathcal{A}}) + E(v, x, V, W_{\mathcal{A}})$. Thus, since X minimizes the error of the subtree rooted at x, U and V must be the sets containing at most i and $k - i - 1$ aggregates, respectively, and that minimizes the error of subtrees rooted at U and V, respectively. Due to the induction hypothesis, $minE(u, x, i, W_{\mathcal{A}}) = E(u, x, U, W_{\mathcal{A}})$ and $minE(v, x, k - i - 1, W_{\mathcal{A}}) = E(v, x, V, W_{\mathcal{A}})$. Thus, since $minE(x, y, k, W_{\mathcal{A}}) \le minE(u, x, i, W_{\mathcal{A}}) + minE(v, x, k - i - 1, W_{\mathcal{A}})$, we can conclude that $minE(x, y, k, W_{\mathcal{A}}) \le E(x, y, X, W_{\mathcal{A}})$. Note that $minE(x, y, k, W_{\mathcal{A}})$ cannot be less than $E(x, y, X, W_{\mathcal{A}})$ since this would lead to a contradiction since X would not minimize the error of the subtree rooted at x. ∎

From Theorem 3.1, it follows that $minE(r(T), \epsilon, k, W_{\mathcal{A}})$ returns the minimum possible error for a tree T when at most k aggregates in the tree can be selected. Procedure COMPUTEMINERROR in Figure 3 uses dynamic programming to compute the k aggregates that result in the minimum possible error for the subtree rooted at x and y is the most specific aggregate covering x that has already been selected. The procedure is invoked with arguments that include the root aggregate of the tree $r(T)$, ϵ and k. The key ideas are similar to those described earlier for the computation of $minE$, the minimum error for the tree. For instance, if an aggregate x has children, then procedure COMPUTEMINERROR(x, y, l) recursively invokes itself for each of its children for the cases when x is selected and when x is not selected. Furthermore, in the case that x has two children, the procedure is invoked for each child for all the possibilities for the number of aggregates in each child subtree.

The only difference is that in addition to the minimum error, the procedure also computes the aggregates that are responsible for minimizing the tree error. Thus, every invocation of procedure COMPUTEMINERROR, in addition to returning the minimum error for the subtree rooted at x, also returns the set of aggregates in the subtree that cause the error to be minimum. This set is derived by taking the union of the optimal aggregates for the subtrees rooted at its children, and adding $\{x\}$ to it if selecting x is required for minimizing the error (Steps 11, 20, and 35). Note also that in order to improve computational efficiency, the optimal aggregates and the minimum error for the subtree rooted at x with y as the most specific aggregate and at most l selected aggregates are stored in subTree[x, y, l].aggregates and subTree[x, y, l].error, respectively. The first invocation of COMPUTEMINERROR(x, y, l) causes the body of the procedure to be executed, but subsequent invocations simply return the previously computed and stored values.

3.3 Combining the Aggregates for Set of Trees

Suppose there are m aggregate trees $T_1, T_2, \ldots, T_m$. Further, let $T_i[j]$.[error, aggregates] denote the minimum error and the set of at most j aggregates in T_i responsible for minimizing T_i's error. Then, $X_i[j]$.[error, aggregates], the minimum error for the set of trees $T_1, \ldots, T_i$ and the j aggregates that minimize their cumulative error can be computed using the result of the following theorem.

Theorem 3.2 For the set of trees $T_1, \ldots, T_m$,

$$X_i[j].\text{error} = \begin{cases} T_i[j].\text{error} & \text{if } i = 1 \\ \min_{0 \le l \le j}\{X_{i-1}[l].\text{error} + T_i[j-l].\text{error}\} & \text{otherwise} \end{cases}$$ ∎

Proof: The proof is by induction on i.
Basis: When $i = 1$, the statement of the theorem clearly holds since $X_1[j].\text{error} = T_1[j].\text{error}$.
Induction Step: Suppose for the trees $T_1, \ldots, T_i$, the cumulative error is minimum for j aggregates when l aggregates are selected from $T_1, \ldots, T_{i-1}$ and the remaining $j - l$ aggregates are selected from T_i. Note that $X_i[j].\text{error} \le X_{i-1}[l].\text{error} + T_i[j-l].\text{error}$. Thus, since due to the induction hypothesis, $X_{i-1}[l]$.error is equal to the minimum

```
procedure COMPUTEMINERROR(Aggregate x, Aggregate y, integer l)
if subTree[x, y, l].computed = true
    return [subTree[x, y, l].error, subTree[x, y, l].aggregates]
minError := minError1 := minError2 := ∞
if x is a leaf {
    minError1 := Σ_{s∈S} D(s, x) ∗ (lsp(s, x, {y}, W_A) − lsp(s, x))
    if l > 0
        minError2 := Σ_{s∈S} D(s, x) ∗ (lsp(s, x, {x}, W_A) − lsp(s, x))
    if minError1 ≤ minError2
        [subTree[x, y, l].error, subTree[x, y, l].aggregates] := [minError1, ∅]
    else
        [subTree[x, y, l].error, subTree[x, y, l].aggregates] := [minError2, {x}]
}
if x has a single child u {
    [minError1, aggregates1] := COMPUTEMINERROR(u, y, l)
    if l > 0
        [minError2, aggregates2] := COMPUTEMINERROR(u, x, l − 1)
    if minError1 ≤ minError2
        [subTree[x, y, l].error, subTree[x, y, l].aggregates] := [minError1, aggregates1]
    else
        [subTree[x, y, l].error, subTree[x, y, l].aggregates] := [minError2, aggregates2 ∪ {x}]
}
if x has children u and v {
    for i := 0 to l {
        [minError1, aggregates1] := COMPUTEMINERROR(u, y, i)
        [minError2, aggregates2] := COMPUTEMINERROR(v, y, l − i)
        if minError1 + minError2 < minError
            minError := minError1 + minError2
            aggregates := aggregates1 ∪ aggregates2
    }
    for i := 0 to l − 1 {
        [minError1, aggregates1] := COMPUTEMINERROR(u, x, i)
        [minError2, aggregates2] := COMPUTEMINERROR(v, x, l − i − 1)
        if minError1 + minError2 < minError
            minError := minError1 + minError2
            aggregates := aggregates1 ∪ aggregates2 ∪{x}
    }
    [subTree[x, y, l].error, subTree[x, y, l].aggregates] := [minError, aggregates]
}
subTree[x, y, l].computed := true
return [subTree[x, y, l].error, subTree[x, y, l].aggregates]
```

Figure 3: Dynamic programming algorithm for computing the aggregates that minimize tree error.

procedure COMBINEMINERROR()
1. **for** $i = 1$ to m
2.   **for** $j = 0$ to k {
3. $T_i[j]$.[error, aggregates] := COMPUTEMINERROR(r(T_i), ϵ, j)
4. $X_i[j]$.[error, aggregates] := $[\infty, \emptyset]$
5.   }
6. **for** $j = 0$ to k
7. $X_1[j]$.[error, aggregates] := $T_1[j]$.[error, aggregates]
8. **for** $i = 2$ to m
9.   **for** $j = 0$ to k
10.   **for** $l = 0$ to j
11.   **if** ($X_{i-1}[l]$.error + $T_i[j-l]$.error < $X_i[j]$.error) {
12. $X_i[j]$.error = $X_{i-1}[l]$.error + $T_i[j-l]$.error
13. $X_i[j]$.aggregates = $X_{i-1}[l]$.aggregates $\cup$ $T_i[j-l]$.aggregates
14.   }

Figure 4: Combining aggregates for a set of aggregate trees.

error for the first $i-1$ trees when l aggregates are selected from them, it follows that $X_i[j]$.error is less than or equal to the minimum error for the first i trees when j aggregates are selected. In fact, since $X_i[j]$.error is set to the minimum of $X_{i-1}[l]$.error + $T_i[j-l]$.error for $0 \leq l \leq j$, due to the induction hypothesis, it must be equal to the minimum error for the first i trees when j aggregates are selected. ∎

Procedure COMBINEMINERROR in Figure 4 computes in $X_m[k]$.[error, aggregates] the minimum cumulative error and the k aggregates that minimize the error for the trees $T_1, \ldots, T_m$. After computing the error and aggregates for each individual tree in Steps 1–5, in each iteration of Steps 8–14, the $X_i[j]$s are computed for increasing values of i based on the individual tree errors and the $X_{i-1}[j]$s computed in the previous iteration (as stated in Theorem 3.2). For each $X_i[j]$, the aggregates are computed by taking the union of the aggregates for the $X_{i-l}[l]$ and $T_i[j-l]$ that result in the minimum error for $X_i[j]$.

3.4 Time and Space Complexity

Suppose that d is the maximum depth of an aggregate tree, the number of aggregates in $\mathcal{A}$ is N and the number of subnets in $\mathcal{S}$ is n. Note that for 32-bit IP addresses, $d \leq 32$. Then the time complexity of the procedure COMPUTEMINERROR can be shown to be $O(n^3 + Ndk^2)$. The reason for this is that $lsp(s,t)$, the shortest path between subnets s and t needs to be computed for all subnet pairs. The time complexity of this step is $O(n^3)$. Also, for each subnet x and every aggregate y covering it, one can precompute and store $\sum_{s \in \mathcal{S}} D(s,x) * (lsp(s,x,\{y\},W_{\mathcal{A}}) - lsp(s,x))$, thus enabling this information to be accessed in constant time. Further, the body of COMPUTEMINERROR is executed at most once for each combination of x, y and l. For a specific x, there are at most dk different possibilities for y and l for which the body of the procedure is executed. This is because y has to be an ancestor of x in the tree and $l \leq k$. Each execution of the body of COMPUTEMINERROR makes at most $2l+1$ recursive calls, and thus, since there are N possible aggregates, the total number of times COMPUTEMINERROR is invoked is $O(Ndk^2)$. As a result, the time complexity of procedure COMPUTEMINERROR is $O(n^3 + Ndk^2)$. Further, the space complexity of the procedure is $O(n^2 + Ndk)$, $O(n^2)$ to store the shortest path and error information for subnets, and $O(Ndk)$ to store the error and aggregate values for each of the Ndk possible combinations of values for x, y and l.

It is fairly straightforward to observe that the three **for** loops spanning Steps 8–14 of procedure COMBINEMINERROR execute $O(mk^2)$ steps. Thus, the overall time complexity of the procedure is is $O(n^3 + Ndk^2 + mk^2)$, where the first two terms are the time complexity of computing the aggregates that minimize the error for the m trees. Note that even though COMBINEMINERROR makes independent successive invocations to COMPUTEMINERROR($r(T_i)$,

$\epsilon, j)$ for $j = 0, \ldots, k$, the results computed in subTree during an invocation are shared between the invocations. The space complexity of procedure COMBINEMINERROR is simply $O(mk)$ to store the X_i and T_i arrays.

3.5 Minimizing Maximum Error

Note that instead of minimizing the cumulative error over source destination subnet pairs, our algorithms can be adapted to minimize the maximum error over source destination pairs. In order to do this, we simply need to redefine the error of a tree to be the maximum error of the leaf subnets in it (instead of the sum of errors). Thus, the recursive definition of the error of the subtree rooted at x given that y is the most specific selected aggregate covering x in the tree and X is the set of selected aggregates, is as follows:

$$E(x, y, X, W_{\mathcal{A}}) = \begin{cases} \max_{(x,u)\in T} E(u, y, X, W_{\mathcal{A}}) & \text{if } x \text{ has children, and } x \notin X \\ \max_{(x,u)\in T} E(u, x, X, W_{\mathcal{A}}) & \text{if } x \text{ has children, and } x \in X \\ \max_{s\in\mathcal{S}} D(s, x) * (lsp(s, x, \{y\}, W_{\mathcal{A}}) - lsp(s, x)) & \text{if } x \text{ is a leaf and } x \notin X \\ \max_{s\in\mathcal{S}} D(s, x) * (lsp(s, x, \{x\}, W_{\mathcal{A}}) - lsp(s, x)) & \text{if } x \text{ is a leaf and } x \in X \end{cases}$$

Further, the minimum error of the subtree rooted at x if at most k aggregates in the subtree can be chosen (given that y is the most specific aggregate in the tree covering x and that has already been selected), is as follows:

$$minE(x, y, k, W_{\mathcal{A}}) = \begin{cases} E(x, y, \emptyset, W_{\mathcal{A}}) & \text{if } k = 0 \\ \min\{minE(u, y, k, W_{\mathcal{A}}), minE(u, x, k-1, W_{\mathcal{A}})\} & \\ & \text{if } k > 0 \text{ and } x \text{ has a single child } u \\ \min\{\min_{0\le i\le k}\{\max\{minE(u, y, i, W_{\mathcal{A}}), minE(v, y, k-i, W_{\mathcal{A}})\}\}, & \\ \qquad \min_{0\le i\le k-1}\{\max\{minE(u, x, i, W_{\mathcal{A}}), minE(v, x, k-1-i, W_{\mathcal{A}})\}\}\} & \\ & \text{if } k > 0 \text{ and } x \text{ has children } u, v \\ \min\{E(x, y, \emptyset, W_{\mathcal{A}}), E(x, y, \{x\}, W_{\mathcal{A}})\} & \text{if } k > 0 \text{ and } x \text{ is a leaf} \end{cases}$$

Note that unlike the cumulative error case, where we were interested in the distributing the aggregates amongst the subtrees of x rooted at children u and v so that the sum of the errors of the subtrees was minimized, for the maximum error case, we are interested in minimizing the maximum of the errors of the two subtrees (since the error of the subtree rooted at x is the maximum of the errors of its two child subtrees). Thus, the following modifications need to be made to procedure COMPUTEMINERROR to compute the k aggregates that minimize the (maximum) error for the tree: (1) replace $\sum_{s\in\mathcal{S}} D(s, x) * (lsp(s, x, \{y\}, W_{\mathcal{A}}) - lsp(s, x))$ in Steps 5 and 7 by $\max_{s\in\mathcal{S}} D(s, x) * (lsp(s, x, \{y\}, W_{\mathcal{A}}) - lsp(s, x))$, and (2) replace minError1 $+$ minError2 in Steps 26, 27, 33, and 34 with $\max\{$minError1, minError2$\}$. Similarly, the following simple modification to procedure COMBINEMINERROR enables it to compute the minimum error of a set of trees for the maximum error case: replace $X_{i-1}[l]$.error + $T_i[j-l]$.error in Steps 11 and 12 by $\max\{X_{i-1}[l]$.error, $T_i[j-l]$.error$\}$.

4 Weight Selection Problem

In the previous section, for a given weight assignment function $W_{\mathcal{A}}$, we proposed algorithms for computing the optimal set of aggregates X for which the error in the selected paths is minimized. However, the final error and set of optimal aggregates X are very sensitive to the weight that a border router advertises for each aggregate. Thus, the weight assignment problem is important for ensuring that selected paths are of high quality, and is the subject of this section.

Recall that the weight assignment problem is to compute a weight assignment function $W_{\{x\}}$ for a single aggregate $x \in A_i$ such that the error in the selected paths from all subnets to destination subnets covered by x is minimized. The weight assignment function $W_{\{x\}}$ assigns a weight to x at each ABR $b \in B_i$. Note that we are

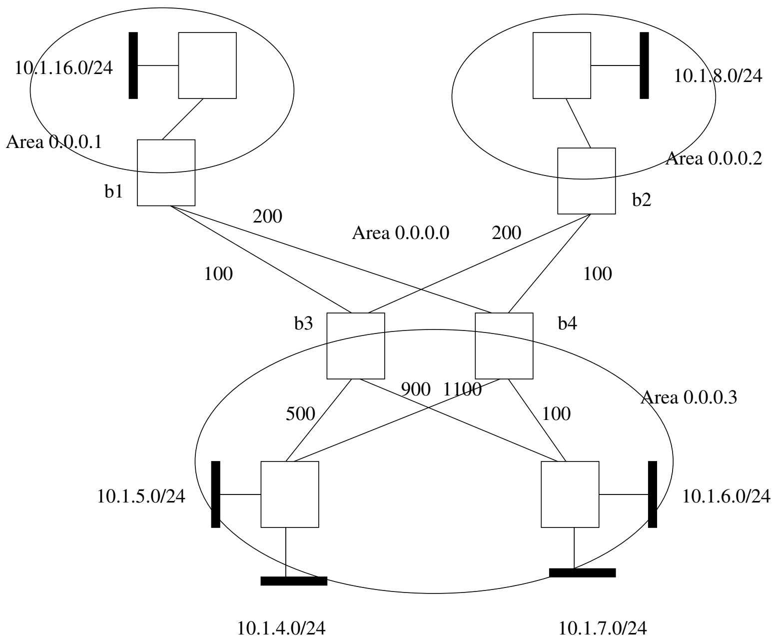


Figure 5: Example of an AS where choosing maximum-distance weights does not minimize the maximum error.

interested in computing the optimal weights for x under the assumption that no other aggregates covering subnets in x are concurrently being advertised. Also, since the aggregate x for whom we wish to compute weights is fixed, we drop the subscript $\{x\}$ for W – thus we will use $W(b)$ to denote the weight assigned to x by ABR $b \in B_i$.

Intuitively, since $W(b)$ is supposed to represent the distance between b and subnets covered by x, two possible logical choices for $W(b)$ are the following:

1. $\max_{t \in x}\{lsp(b,t)\}$.
2. $\frac{1}{|x|}\sum_{t \in x} lsp(b,t)$.

The first choice, recommended in [Moy98], is simply the maximum distance of a subnet in aggregate x from the border router b, while the latter is the average distance of subnets in x from b. Note that since both choices are oblivious of the source subnets (not covered by x) and the error to be minimized, as illustrated in the examples below, for most cases, neither choice optimizes our objective error function. In the following two examples, we show that choosing $W(b)$ to be $\max_{t \in x}\{lsp(b,t)\}$ minimizes neither the cumulative error nor the maximum error.

Example 4.1 Consider the network in Figure 1. Suppose we are interested in computing weights for the aggregate 10.1.0.0/21 that covers all the subnets in Area 0.0.0.3. If each border router chose the maximum distance to a subnet in 10.1.0.0/21 as the weight for it, b_3 would assign 10.1.0.0/21 a weight of 1100 (distance of b_3 from 10.1.6.0/24) and b_4 would assign to 10.1.0.0/21 a weight of 1250 (distance between b_4 and 10.1.6.0/24). Consequently, both subnets 10.1.16.0/24 and 10.1.8.0/24 select the path through ABR b_3 to access the subnets in 10.1.0.0/21 which has a cumulative error of $2*0+2*700+2*800 = 3000$ (for 10.1.16.0/24) and $2*0+2*900+2*1000 = 4800$ (for 10.1.8.0/24). In contrast, assigning weights 1000 and 500 to 10.1.0.0/21 at ABRs b_3 and b_4, respectively, causes the selected paths to be through b_4 which results in much smaller cumulative errors of $2*1300+2*0+2*0 = 2600$ (for 10.1.16.0/24) and $2*1100+2*0+2*0 = 2200$ (for 10.1.8.0/24). ∎

Example 4.2 Consider the network in Figure 5. Suppose we are interested in computing weights for the aggregate 10.1.0.0/21 that covers all the subnets in Area 0.0.0.3. If each border router chose the maximum distance to a subnet in 10.1.0.0/21 as the weight for it, b_3 would assign 10.1.0.0/21 a weight of 900 (distance of b_3 from 10.1.6.0/24) and b_4 would assign to 10.1.0.0/21 a weight of 1100 (distance between b_4 and 10.1.4.0/24). Consequently, both subnets 10.1.16.0/24 and 10.1.8.0/24 select the path through ABR b_3 to access the subnets in 10.1.0.0/21 which has a maximum error of $\max\{0, 700\} = 700$ (for 10.1.16.0/24) and $\max\{0, 900\} = 900$ (for 10.1.8.0/24). In contrast, assigning weights 1000 and 500 to 10.1.0.0/21 at ABRs b_3 and b_4, respectively, causes the selected paths to be through b_4 which results in lower values for maximum error – $\max\{700, 0\} = 700$ (for 10.1.16.0/24) and $\max\{500, 0\} = 500$ (for 10.1.8.0/24). ∎

Choosing $W(b) = \frac{1}{|x|}\sum_{t\in x} lsp(b,t)$ yields somewhat better results because intuitively this is more representative of the distance between b and subnets in x than $\max_{t\in x}\{lsp(b,t)\}$. As a matter of fact, setting $W(b)$ to be the average distance of b to subnets in x can be shown to minimize the cumulative error for the weight selection problem. However, it does not minimize the maximum error as illustrated by the example below.

Example 4.3 Consider the network in Figure 1. Suppose we are interested in computing weights for the aggregate 10.1.0.0/21 that covers all the subnets in Area 0.0.0.3. If each border router chose the average distance to a subnet in 10.1.0.0/21 as the weight for it, b_3 would assign 10.1.0.0/21 a weight of 730 and b_4 would assign to 10.1.0.0/21 a weight of 570. Consequently, both subnets 10.1.16.0/24 and 10.1.8.0/24 select the path through ABR b_4 to access the subnets in 10.1.0.0/21 which has a maximum error of 1300 for 10.1.16.0/24 and 1100 for 10.1.8.0/24. In contrast, assigning weights 500 and 1000 to 10.1.0.0/21 at ABRs b_3 and b_4, respectively, causes the selected paths to be through b_3 which results in lower values for maximum error – 700 for 10.1.16.0/24 and 1000 for 10.1.8.0/24. ∎

In the following subsections, we first show that selecting $W(b) = \frac{1}{|x|}\sum_{t\in x} lsp(b,t)$ results in the minimum cumulative error and is a solution to the weight selection problem. However, the generalized weight selection problem that involves minimizing the product of the cumulative error of selected paths and their degrees of importance is an $\mathcal{NP}$-hard problem [GJ79]. Consequently, we present search-based heuristics to solve the generalized weight selection problem and a pseudo-polynomial time algorithm to solve the weight selection problem when the objective is to minimize the maximum error. Finally, we show that efficient algorithms can be devised for the generalized weight selection problem when B_i contains only 2 ABRs.

4.1 Problem Formulation

In this subsection, we simplify some of the notation and introduce some new terminology that we need in order to address the weight selection problem which is: for an aggregate $x \in A_i$, compute a weight assignment function W such that $\sum_{s\in\mathcal{S},t\in x}(lsp(s,t,\{x\},W) - lsp(s,t))$ is minimum. For each source s, the selected paths to subnets covered by x is through the ABR $b \in B_i$ for which $lsp(s,b) + W(b)$ is minimum (among all the ABRs). We denote the ABR selected for source s by $B(s,W)$. Note that for $t \in x$, $lsp(s,t,\{x\},W) = lsp(s,B(s,W)) + lsp(B(s,W),t)$. Further, suppose $e(s,b)$ denotes the error in the selected paths to subnets in x if ABR b is selected for source s. Thus, $e(s,b) = \sum_{t\in x} lsp(s,b) + lsp(b,t) - lsp(s,t)$. Then, $e(s,B(s,W)) = \sum_{t\in x} lsp(s,t,\{x\},W) - lsp(s,t)$, and thus the weight selection problem becomes that of computing a weight assignment W such that $\sum_{s\in\mathcal{S}} e(s,B(s,W))$ is minimum.

The above problem formulation is for minimizing the cumulative error. If we wish to minimize the maximum error, then $e(s,b) = \max_{t\in x}\{lsp(s,b) + lsp(b,t) - lsp(s,t)\}$ and the weight assignment W must be such that $\max_{s\in\mathcal{S}} e(s,B(s,W))$ is minimum.

4.2 Weight Selection Problem (Cumulative Error)

For the cumulative error case, it can be shown that choosing $W(b)$ to be the average distance of b to subnets in x minimizes the cumulative error in the selected paths between sources and destination subnets in x.

Theorem 4.1 The weight assignment function W which assigns a weight $W(b) = \frac{1}{|x|}\sum_{t\in x} lsp(b,t)$ to ABR b results in the minimum value for $\sum_{s\in\mathcal{S}} e(s, B(s,W))$. ∎

Proof: Suppose the weight assignment function W assigns a weight $W(b) = \frac{1}{|x|}\sum_{t\in x} lsp(b,t)$ to ABR b. For each source s, $lsp(s, B(s,W)) + W(B(s,W))$ is less than or equal to $lsp(s,b) + W(b)$ for all ABRs b. Thus, first expanding W and then multiplying by constant $|x|$ and subtracting constant $\sum_{t\in x} lsp(s,t)$ from both sides, we get $\sum_{t\in x} lsp(s, B(s,W)) + lsp(B(s,W),t) - lsp(s,t) \le \sum_{t\in x} lsp(s,b) + lsp(b,t) - lsp(s,t)$ for all ABRs b. As a result, for every source s, $e(s, B(s,W)) \le e(s,b)$ for all ABRs b and thus, $\sum_{s\in\mathcal{S}} e(s, B(s,W))$ is minimum for the weight assignment W. ∎

4.3 Generalized Weight Selection Problem (Cumulative Error)

For the cumulative error case, $e(s,b) = \sum_{t\in x} lsp(s,b) + lsp(b,t) - lsp(s,t)$ is closely related to the criterion for selecting an ABR b for s which is that $lsp(s,b) + W(b)$ is minimum (note that $\sum_{t\in x} lsp(s,t)$ is a constant). However, for the generalized cumulative error case, $e(s,b) = \sum_{t\in x} D(s,t) * (lsp(s,b) + lsp(b,t) - lsp(s,t))$ and thus $e(s,b)$ can be any arbitrary value based on the value of $D(s,t)$. This fact that $e(s,b)$ can be any arbitrary value makes the problem of computing a weight assignment function W that minimizes $\sum_{s\in\mathcal{S}} e(s, B(s,W))$ intractable, as the following theorem demonstrates. The proof of the theorem involves a rather complex reduction from 3-SAT and, in the interest of space, has been moved to Appendix A.1.

Theorem 4.2 For arbitrary values of $e(s,b)$ and constant E, determining if there exists a weight assignment function W for which $\sum_{s\in\mathcal{S}} e(s, B(s,W)) \le E$ is $\mathcal{NP}$-hard. ∎

A simple iterative greedy search heuristic can be used to compute a weight assignment W that results in a low value for the cumulative error. The basic idea is to start with a set $\mathcal{W}$ of random weight assignments. Then, in each subsequent iteration, for each $W \in \mathcal{W}$, from a number of candidate modifications, the one that minimizes the cumulative error is greedily chosen and applied to W. For a $W \in \mathcal{W}$, each candidate modification consists of adjusting the weight $W(b)$ for a single ABR $b \in B_i$. Thus, for each weight assignment W, we are interested in computing the ABR b and a weight w such that setting $W(b) = w$ (and leaving the weights for other ABRs unchanged) results in the smallest value for the cumulative error $\sum_s e(s, B(s,W))$. For n sources and m ABRs, this can be computed in $O(mn(\log n + m))$ time as follows. First, we compute for each ABR b, the weight w such that setting $W(b) = w$ minimizes the error. Then, we choose from all the (ABR, weight) pairs (b,w) the one that results in the minimum error.

In order to compute the optimal weight w for an ABR b, we first compute for every source s, the ABR c in $B_i - \{b\}$ for which $lsp(s,c) + W(c)$ is minimum (this can be achieved in $O(mn)$ steps). For the source s, let $v(s) = lsp(s,c) + W(c) - lsp(s,b)$ and $e(s) = e(s,c)$. Suppose that $v(s_1), \ldots, v(s_n)$ are the values for the sources in sorted (increasing) order (sorting the n values takes $O(n\log n)$ steps). Also, let $v(s_{n+1}) = \infty$. Then, choosing a value for $W(b)$, such that $v(s_{j-1}) < W(b) \le v(s_j)$ causes the cumulative error to be $\sum_{l=1}^{j-1} e(s_l) + \sum_{l=j}^{n} e(s_l, b)$ (since ABR b is selected for sources $s_j, \ldots, s_n$ when $W(b) \le v(s_j)$). Thus, in a single pass over the sequence $v(s_1), \ldots, v(s_n)$, the optimal weight $v(s_j)$ for $W(b)$ which minimizes the cumulative error $\sum_{l=1}^{j-1} e(s_l) + \sum_{l=j}^{n} e(s_l, b)$ can be computed. Thus, the optimal weight for each ABR can be computed in $O(n(\log n + m))$ steps and for all ABRs, in $O(mn(\log n + m))$ time.

To recap, the greedy heuristic, in each iteration, modifies each weight assignment $W \in \mathcal{W}$ by setting $W(b) = w$, where ABR b and weight w result in the minimum error for W and are computed as described above. It terminates the search computation either after a fixed number of iterations or if the improvement in cumulative error during an iteration due to modifying every $W \in \mathcal{W}$ drops below an error threshold ϵ.

4.4 Generalized Weight Selection Problem (Maximum Error)

Recall that if we are interested in minimizing the maximum error, then $e(s, b) = \max_{t \in x}\{lsp(s, b) + lsp(b, t) - lsp(s, t)\}$ and the weight assignment W must be such that $\max_{s \in \mathcal{S}} e(s, B(s, W))$ is minimum. Thus, we can employ an algorithm similar to the greedy search heuristic described earlier to compute a weight assignment function that minimizes the maximum error (instead of the cumulative error).

However, if we assume that weight assignments W and shortest path distances lsp to be non-negative integers, then we can devise a more efficient pseudo-polynomial time algorithm for computing the weight assignment that minimizes the maximum error. Suppose we could devise a procedure P that computes a weight assignment W (if there exists one) such that $\max_{s \in \mathcal{S}} e(s, B(s, W)) \leq E$ for some constant E. Then, a simple procedure for computing the weight assignment that minimizes the maximum error is as follows: (1) Sort the errors $e(s, b)$ between (source, ABR) pairs – let $E_1, \ldots, E_r$ be the errors in order of increasing value, (2) Repeatedly invoke the procedure P for increasing values of i, until P returns a weight assignment W for which $\max_{s \in \mathcal{S}} e(s, B(s, W)) \leq E_i$. Thus, E_i is the smallest value for which a weight assignment exists and represents the minimum possible value for the maximum error. Further, W is the weight assignment that minimizes the maximum error. Note that instead of considering each E_i sequentially, one can also use a binary search procedure to compute the minimum value for the maximum error more efficiently.

Thus, the crucial task for us is to develop the procedure P that computes a weight assignment W (if there exists one) such that $\max_{s \in \mathcal{S}} e(s, B(s, W)) \leq E$ for some constant E. We show that the problem of computing a W such that the maximum error is at most E is equivalent to solving a set of inequalities involving the $W(b)$s as variables. For a source s, let $R(s)$ denote the set of ABRs $b \in B_i$ for which $e(s, b) \leq E$ – thus, for the remaining ABRs $b \in B_i - R(s)$, $e(s, b) > E$. Consequently, since the error for each source can be at most E, the computed W must be such that one of the ABRs in $R(s)$ is selected for s. For this, we require W to satisfy the following set of inequalities:

$$\min_{b \in R(s)}\{W(b) + lsp(s, b)\} \leq W(c) + lsp(s, c) \qquad \text{for all } c \in B_i - R(s)$$

Thus, for each source s, we obtain the set of inequalities described above[1]. Note that the W's in the equations are variables and the lsp's are constants. Also, for each ABR b, W also needs to satisfy the constraint $0 \leq W(b)$. Suppose Q denotes the set of inequalities over all the sources and ABRs. It is straightforward to observe that for a W the maximum error is E if and only if W is a solution for the set of equations Q. Thus, we simply need to focus on computing a W that satisfies the inequalities in Q. Observe that if for a source s, the set $R(s)$ is empty, then there does not exist a W for which the set of inequalities Q is satisfiable. The reason for this is that for the source, we obtain inequalities of the form $\min\{\} \leq W(c) + lsp(s, c)$ which cannot be satisfied since $\min\{\} = \infty$. Also, no equations are generated for a source s if $R(s) = B_i$ (that is, the error for the source s is at most E irrespective of the chosen ABR).

Procedure COMPUTEWEIGHTSMAX in Figure 6 is an iterative pseudo-polynomial time algorithm for computing a W that satisfies Q. In the procedure, W_{new} and W_{old} store the weight assignment values prior to and after each iteration. In each iteration, a new weight assignment W_{new} is computed after substituting the previous weight assignment W_{old} for the W's only on the LHS of each inequality in Q (Steps 3–4). (We use LHS and RHS to denote the left and right hand-side of an inequality, respectively.) Note that each inequality in Q' has the form $C \leq W(c) + lsp(s, c)$, where C and $lsp(s, c)$ are constants and $W(c)$ is a variable. Also, lsp_{max} is the maximum value for $lsp(s, b)$ for a (source, ABR) pair.

[1] We assume that if for two ABRs b and c, $W(b) + lsp(s, b) = W(c) + lsp(s, c)$, then the ABR with a smaller error is selected for s. In case this assumption does not hold, the following stronger inequality can be employed.

$$\min_{b \in R(s)}\{W(b) + lsp(s, b)\} + 1 \leq W(c) + lsp(s, c) \qquad \text{for all } c \in B_i - R(s)$$

procedure COMPUTEWEIGHTSMAX(Q)

1. **for each** $b \in B_i$, set $W_{old}(b) := 0$
2. **while** $(\sum_{b \in B_i} W_{old}(b) \leq (\frac{|B_i|*(|B_i|-1)}{2}) * lsp_{max})$ {
3. Let Q' be a new set of inequalities that result when the value $W_{old}(b)$ is substituted for each variable $W(b)$ only on the LHS of each inequality in Q
4. **for each** $b \in B_i$, set $W_{new}(b)$ to the smallest possible value such that each inequality in Q' is satisfied when $W_{new}(b)$ is substituted for variable $W(b)$ in Q'
5. **if** $W_{new} = W_{old}$
6. **return** W_{new}
7. **else**
8. $W_{old} := W_{new}$
9. }
10. **return** "there does not exist a weight assignment W"

Figure 6: Algorithm for computing weights for 2 ABRs.

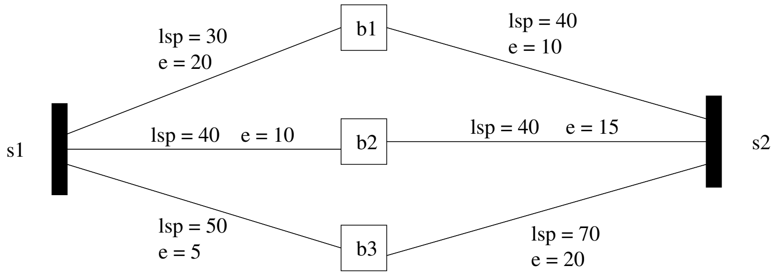


Figure 7: Example network for tracing execution of procedure COMPUTEWEIGHTSMAX.

Example 4.4 Consider sources s_1 and s_2, and ABRs b_1, b_2 and b_3 shown in Figure 7. Let the functions lsp and e for the (source, ABR) pairs be as depicted in the figure. For $E = 10$, $R(s_1) = \{b_2, b_3\}$ and $R(s_2) = \{b_1\}$. Thus, for source s_1, Q contains the following inequality:

$$\min\{W(b_2) + 40, W(b_3) + 50\} \leq W(b_1) + 30 \tag{1}$$

And for source s_2, Q contains the following two inequalities:

$$W(b_1) + 40 \leq W(b_2) + 40 \tag{2}$$

$$W(b_1) + 40 \leq W(b_3) + 70 \tag{3}$$

Note that since weight assignments cannot be negative, even though we don't explicitly state this, Q and Q' always contains the following three constraints: $0 \leq W(b_1)$, $0 \leq W(b_2)$ and $0 \leq W(b_3)$.

We now trace the execution of COMPUTEWEIGHTSMAX for the above set of inequalities in Q. Initially, $W_{old}(b_i)$ is set to 0 for all 3 ABRs. In the first iteration, substituting 0 for all the $W(b_i)$ variables on the LHS of Inequalities (1), (2) and (3) results in the following set of equations Q' (Step 3 of the procedure): $\min\{10, 20\} \leq W(b_1)$ (due to Inequality 1), $0 \leq W(b_2)$ (due to Inequality 2) and $-30 \leq W(b_3)$ (due to Inequality 3). As a result, $W_{new}(b_1)$ is set to 10, while W_{new} for b_2 and b_3 continue to be 0. At the beginning of the second iteration, thus, $W_{old}(b_1) = 10$. Consequently, after substitution of $W_{old}(b_i)$ for the $W(b_i)$ variables on the LHS of inequalities in Q, Q' contains equations $\min\{10, 20\} \leq W(b_1)$, $10 \leq W(b_2)$ and $-20 \leq W(b_3)$. This causes $W_{old}(b_1)$ and $W_{old}(b_2)$ to be set to 10 and 10, respectively, at the end of the iteration (in Step 8). Similarly, it can be shown that at the end of the third iteration, $W_{old}(b_1)$ and $W_{old}(b_2)$ are set to 20 and 10, respectively, and during the 4th iteration, $W_{old}(b_1)$

and $W_{old}(b_2)$ are set to 20 and 20, respectively. In the fifth and final iteration, equations in Q' after substitution are $20 \leq W(b_1)$, $20 \leq W(b_2)$ and $-10 \leq W(b_3)$, causing W_{new} to be equal to W_{old}. Thus, in the final weight assignment returned by COMPUTEWEIGHTSMAX, $W(b_1) = 20$, $W(b_2) = 20$ and $W(b_3) = 0$. ∎

In the following, we show that COMPUTEWEIGHTSMAX returns a W that is a solution to Q if and only if Q is satisfiable. In order to show this, in the following lemmas, we show that (1) for any W that satisfies Q, $W_{old}(b) \leq W(b)$, and (2) if Q is satisfiable, then there exists a W that is a solution to Q for which $\sum_{b \in B_i} W(b) \leq (\frac{|B_i|*(|B_i|-1)}{2}) * lsp_{max}$. Thus, since W_{old} does not decrease between successive iterations and the procedure terminates only when a W is found or $\sum_{b \in B_i} W_{old}(b)$ becomes greater than $(\frac{|B_i|*(|B_i|-1)}{2}) * lsp_{max}$, COMPUTEWEIGHTSMAX computes W correctly. In the proofs, W^j_{old} denote the value of W_{old} at the end of the j^{th} iteration. Also, Q'_j denotes the set of inequalities in Q when W^j_{old} is substituted for all variables on the LHS of each inequality in Q.

Lemma 4.3 For every ABR $b \in B_i$, $W^{j+1}_{old}(b) \geq W^j_{old}(b)$. ∎

Proof: The proof is by induction on j.
Basis: Clearly the lemma holds for $j = 0$. At the end of the 0^{th} iteration (that is, initially), $W^0_{old}(b) = 0$ for every ABR $b \in B_i$. Since Q contains the inequality $0 \leq W(b)$ for every ABR b, $W_{new}(b) \geq 0$ and thus $W^1_{old}(b) \geq 0$.
Induction Step: Consider a $j > 0$. Since due to the induction hypothesis, $W^j_{old}(b) \geq W^{j-1}_{old}(b)$, it follows that the LHS of each inequality in Q'_{j-1} is less than or equal to the LHS of the corresponding inequality in Q'_j. Thus, since $W^j_{old}(b)/W^{j+1}_{old}(b)$ is the smallest possible value for $W(b)$ for which the inequalities in Q'_{j-1}/Q'_j with $W(b)$ on the RHS are satisfied, it follows that $W^{j+1}_{old}(b) \geq W^j_{old}(b)$. ∎

Lemma 4.4 For every weight assignment W' that is a solution to Q, $W'(b) \geq W_{old}(b)$, for every ABR $b \in B_i$. ∎

Proof: We use induction to show that the lemma holds at the end of each iteration j.
Basis: Clearly the lemma holds for $j = 0$. At the end of the 0^{th} iteration (that is, initially), $W^0_{old}(b) = 0$ for every ABR $b \in B_i$. Thus, since $W'(b) \geq 0$, the lemma holds at the end of the 0^{th} iteration.
Induction Step: Consider a $j > 0$. At the start of the j^{th} iteration, due to the induction hypothesis, we have that $W'(b) \geq W^{j-1}_{old}(b)$. Suppose that Q' denotes the set of inequalities in Q when all variables on the LHS of every inequality in Q are substituted with W'. Due to the induction hypothesis, it follows that the LHS of each inequality in Q'_{j-1} is less than or equal to the LHS of the corresponding inequality in Q'. Thus, since $W^j_{old}(b)/W'(b)$ is the smallest possible value for $W(b)$ for which the inequalities in Q'_{j-1}/Q' with $W(b)$ on the RHS are satisfied, it follows that $W'(b) \geq W^j_{old}(b)$. ∎

Lemma 4.5 If Q is satisfiable, then there exists a weight assignment W' that is a solution to Q and for which $\sum_{b \in B_i} W'(b) \leq (\frac{|B_i|*(|B_i|-1)}{2}) * lsp_{max}$. ∎

Proof: Let W' be a weight assignment that is a solution for Q with the smallest value for $\sum_{b \in B_i} W'(b)$ and further suppose that $\sum_{b \in B_i} W'(b) > (\frac{|B_i|*(|B_i|-1)}{2}) * lsp_{max}$. Suppose v is the smallest value for $W'(b)$ among all the ABRs. Then, v must be 0 since the weight assignment W'' where $W''(b) = W'(b) - v$ is also a solution to Q and $\sum_{b \in B_i} W''(b) < \sum_{b \in B_i} W'(b)$ – which leads to a contradiction. Without loss of generality, let $W'(b_1), \ldots, W'(b_{|B_i|})$ be the weights for ABRs sorted in increasing order. Thus, $W'(b_1) = 0$. Since $\sum_{b \in B_i} W'(b) > (\frac{|B_i|*(|B_i|-1)}{2}) * lsp_{max}$, it must be the case that for a pair of consecutive ABRs b_l and b_{l+1}, $W'(b_{l+1}) - W'(b_l) > lsp_{max}$. We show that W'' is a solution for Q, where $W''(b_j) = W'(b_j)$, for $1 \leq j \leq l$ and $W''(b_j) = W'(b_j) - 1$, for $l+1 \leq j \leq |B_i|$. However, this leads to a contradiction since $\sum_{b \in B_i} W''(b) < \sum_{b \in B_i} W'(b)$.

In order to show that Q is satisfiable for W'', we need to consider the following two cases for each inequality.

1. Variable $W(b_j)$ is on the RHS of the inequality ($1 \leq j \leq l$). In this case, the value of the RHS of the inequality is identical for both W' and W'', while the value of the LHS for W'' is less than or equal to the LHS for W'. Thus, if the inequality was satisfiable for W', it is also satisfiable for W''.

2. Variable $W(b_j)$ is on the RHS of the inequality ($l + 1 \leq j \leq |B_i|$). In this case, the value of the RHS of the inequality decreases by 1 for W'' compared to W'. If the value of the LHS of the inequality also decreases by 1 for W'' compared to W', then since the inequality is satisfiable for W', it is also satisfiable for W''.

   The other case we need to consider is when the value of the LHS of the inequality is the same for W' and W''. This corresponds to the case when the minimum value of the LHS (for both W' and W'') is due to the term $W(b_k) + lsp(s, b_k)$ for some $1 \leq k \leq l$. Since the inequality holds for W', $W'(b_k) + lsp(s, b_k) \leq W'(b_j) + lsp(s, b_j)$. Further, since $k \leq l < j$, $W'(b_j) - W'(b_k) > lsp_{max}$. Rearranging terms, $W'(b_j) > W'(b_k) + lsp_{max}$, or alternately, $W'(b_j) > W'(b_k) + lsp(s, b_k)$ (since $lsp(s, b_k) \leq lsp_{max}$). Thus, $W'(b_j) + lsp(s, b_j) > W'(b_k) + lsp(s, b_k)$ and $W'(b_j) + lsp(s, b_j) - 1 \geq W'(b_k) + lsp(s, b_k)$. However, since $W''(b_j) = W'(b_j) - 1$ and $W''(b_k) = W'(b_k)$, the inequality holds for W''.

Thus, W'' is a solution for Q. However, this leads to a contradiction since $\sum_{b \in B_i} W''(b) < \sum_{b \in B_i} W'(b)$, and thus it must be the case that $\sum_{b \in B_i} W'(b) \leq (\frac{|B_i| * (|B_i| - 1)}{2}) * lsp_{max}$. ▮

Clearly, if a weight assignment W is returned by procedure ComputeWeightsMax, then this is a solution to Q. The reason for this is that for the returned W, $W = W_{new} = W_{old}$ – thus, when the value for $W(b)$ returned by the procedure is substituted for the occurrence of variable $W(b)$ in each inequality, every inequality is satisfied. However, we also need to show that our procedure finds a solution for Q if one exists.

Theorem 4.6 If Q is satisfiable, then procedure ComputeWeightsMax returns a weight assignment W that is a solution to Q. ▮

Proof: Suppose Q is satisfiable. Due to Lemma 4.5, it follows that there exists a W that is a solution to Q such that $\sum_{b \in B_i} W(b) \leq (\frac{|B_i| * (|B_i| - 1)}{2}) * lsp_{max}$. Further, due to Lemma 4.4, $W_{old}(b) \leq W(b)$ and so for W_{old}, $\sum_{b \in B_i} W_{old}(b) \leq (\frac{|B_i| * (|B_i| - 1)}{2}) * lsp_{max}$. Also, due to Lemma 4.3, $\sum_{b \in B_i} W_{old}(b)$ at the end of an iteration is greater than or equal to it's value at the end of the previous iteration. Thus, since $\sum_{b \in B_i} W_{old}(b)$ cannot exceed $(\frac{|B_i| * (|B_i| - 1)}{2}) * lsp_{max}$, at some point during the execution of the procedure, the value of W_{old} between two consecutive iterations does not change. This weight assignment W is returned by the procedure and is a solution to Q. The reason for this is that for the returned W, $W = W_{new} = W_{old}$ – thus, when the value for $W(b)$ returned by the procedure is substituted for the occurrence of variable $W(b)$ in each inequality, every inequality is satisfied. ▮

The worst-case time complexity of the overall procedure to compute a W that minimizes the maximum error (by repeatedly invoking ComputeWeightsMax for error values in $E_1, \ldots, E_r$, $r = mn$) can be shown to be $O(m^4 \log(mn) lsp_{max})$, where $m = |B_i|$ is the number of ABRs and n is the number of sources. The $\log(mn)$ term is due to the binary search over all the errors to determine the minimum error for which a weight assignment W can be computed by ComputeWeightsMax. In the worst case, ComputeWeightsMax performs $O(m^2 lsp_{max})$ iterations since the procedure terminates when the sum of the weights become $(\frac{|B_i| * (|B_i| - 1)}{2}) * lsp_{max}$ and the sum increases by at least 1 in each iteration. Finally, the time complexity for computing W_{new} in each iteration is $O(m^2)$ since in the worst case, the number of inequalities in Q is m^2 (one inequality for every pair of W variables, one on the LHS and the other on the RHS of the inequality).

4.5 Generalized Weight Selection Problem (2 ABRs)

For the special case when B_i contains only 2 ABRs, b_1 and b_2, it is possible to devise an efficient algorithm whose time complexity is $O(n \log n)$ where n is the number of sources. In this subsection, due to space constraints, we

```
procedure ComputeWeightsTwoABR()
Set V_opt := v(s_1), E := E_opt := Σ_{s∈S} e(s, b_1)
for j := 1 to n {
    E := E + e(s_j, b_2) − e(s_j, b_1)
    if E < E_opt
        V_opt := v(s_{j+1}), E_opt := E
}
return V_opt
```

Figure 8: Algorithm for computing weights for 2 ABRs.

only address the problem of computing a weight assignment W such that the cumulative error $\sum_{s \in \mathcal{S}} e(s, B(s, W))$ is minimized; however, it is straightforward to modify our proposed algorithm to minimize the maximum error.

The error for a source s is $e(s, b_1)$ if $W(b_1) + lsp(s, b_1) \leq W(b_2) + lsp(s, b_2)$ and $e(s, b_2)$ otherwise. For source s, suppose that $v(s) = lsp(s, b_2) - lsp(s, b_1)$ and (without loss of generality) let $v(s_1), \ldots, v(s_n)$ denote the values in the sorted order. Also, let s_0 and s_{n+1} be two dummy sources with zero error to both ABRs, $v(s_0) = -\infty$ and $v(s_{n+1}) = \infty$. Then, for $0 \leq j \leq n$, if $v(s_j) < W(b_1) - W(b_2) \leq v(s_{j+1})$, the cumulative error is $\sum_{l=j+1}^{n} e(s_l, b_1) + \sum_{l=1}^{j} e(s_l, b_2)$. Thus, the problem of computing a W reduces to that of computing the $v(s_j), v(s_{j+1})$ pair for which the cumulative error is the smallest.

Procedure ComputeWeightsTwoABR in Figure 8 computes in variable V_{opt} the upper value of the $v(s_j), v(s_{j+1})$ pair with the minimum cumulative error (that is stored in E_{opt}). Variable E in the procedure is used to keep track of the cumulative error for the $v(s_j), v(s_{j+1})$ pair that is currently under consideration in the for loop. Thus, choosing values for $W(b_1)$ and $W(b_2)$ such that $W(b_1) - W(b_2) = V_{opt}$ yields the desired weight assignment function W that minimizes the cumulative error.

5 Concluding Remarks

Address aggregation within OSPF areas is critical for scalability since it can result in significant reductions in routing table sizes, smaller link-state databases and less network traffic to synchronize the router link-state databases. However, address aggregation can also lead to the selection of *suboptimal* OSPF routing paths between source-destination subnet pairs that span different areas. In this paper, we addressed the important practical problem of configuring OSPF aggregates at *Area Border Routers* (ABRs) to minimize the error in OSPF shortest path computations due to subnet aggregation. We first developed an optimal dynamic programming algorithm that, given an upper bound k on the number of aggregates to be advertised by the ABRs and a weight-assignment function for the aggregates, computes the k aggregates that result in the minimum cumulative/maximum error in the shortest path computations for all source-destination subnet pairs. Subsequently, we tackled the problem of assigning weights to OSPF aggregates such that the cumulative/maximum error in the computed shortest paths is minimized. We showed that, while for certain special cases (e.g., unweighted cumulative error) efficient optimal algorithms for the weight-assignment problem can be devised, the general problem itself is $\mathcal{NP}$-hard. We proposed a randomized search strategy for the general case of weighted cumulative error, and an optimal pseudo-polynomial time algorithm for the maximum error case. To the best of our knowledge, our work is the first to carry out a systematic study of the algorithmic issues underlying the configuration of OSPF aggregates and to propose efficient configuration algorithms that are *provably optimal* for many practical scenarios.

Acknowledgements: We would like to thank Mihalis Yannakakis for pointing out to us the optimality of the weight assignment that assigns to each aggregate the average of the weights in subnets covered by it.

References

[FT00] Bernard Fortz and Mikkel Thorup. "Internet Traffic Engineering by Optimizing OSPF Weights". In *Proceedings of IEEE INFOCOM'2000*, Tel Aviv, Israel, March 2000.

[GJ79] M.R. Garey and D.S. Johnson. *"Computers and Intractability: A Guide to the Theory of NP-Completeness"*. W.H. Freeman, 1979.

[Hui00] C. Huitema. *"Routing in the Internet, 2nd Edition"*. Prentice-Hall, Inc., 2000.

[Moy98] J. T. Moy. *"OSPF Anatomy of an Internet Routing Protocol"*. Addison-Wesley Professional Computing Series, 1998.

– This appendix contains material that can be read at the discretion of the reviewer and has been included only for the purpose of completeness. –

A Proofs of Theoretical Results

A.1 Proof of Theorem 4.2

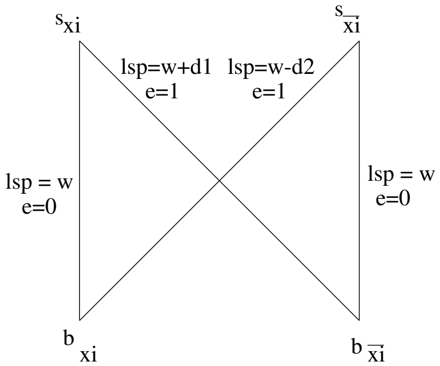


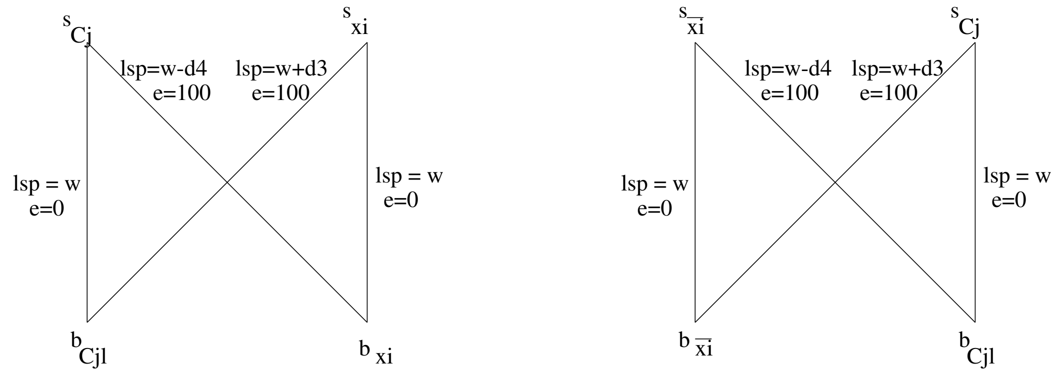
Figure 9: Values of lsp and e for each Literal

Proof: We show that the problem of determining the existence of a weight assignment is $\mathcal{NP}$-hard by reducing 3-SAT to it. In 3-SAT we are given a boolean formula of the form $C_1 \wedge \cdots \wedge C_n$ over literals $x_1, \ldots, x_m$. Each conjunct C_j consists of a disjunction of 3 terms, where each term is either a literal x_i or its negation $\bar{x}_i$. The 3-SAT problem is to determine if there exists an assignment of truth values to each literal x_i (true or false) such that every conjunct becomes true.

Given an instance of the 3-SAT problem, we construct an instance of the weight assignment problem as follows. For each literal x_i, we introduce two sources s_{x_i} and $s_{\bar{x}_i}$ and two ABRs b_{x_i} and $b_{\bar{x}_i}$. Further, we assign values to lsp and e between the sources and ABRs as depicted in Figure 9. Here w, d_1 and d_2 are constants and $d_1 < d_2$. Thus, $lsp(s_{x_i}, b_{x_i}) = lsp(s_{\bar{x}_i}, b_{\bar{x}_i}) = w$, $lsp(s_{x_i}, b_{\bar{x}_i}) = w + d_1$ and $lsp(s_{\bar{x}_i}, b_{x_i}) = w - d_2$. Also, $e(s_{x_i}, b_{x_i}) = e(s_{\bar{x}_i}, b_{\bar{x}_i}) = 0$, and $e(s_{x_i}, b_{\bar{x}_i}) = e(s_{\bar{x}_i}, b_{x_i}) = 1$.

Note that the above choice of values for lsp and e ensures that for any weight assignment W, only one of $B(s_{x_i}, W) = b_{x_i}$ or $B(s_{\bar{x}_i}, W) = b_{\bar{x}_i}$ is possible, but not both. This is because if for an assignment W, b_{x_i} is to be chosen for source s_{x_i} and $b_{\bar{x}_i}$ is to be chosen for source $s_{\bar{x}_i}$, then the following constraints need to hold:

$$W(b_{x_i}) + w \leq W(b_{\bar{x}_i}) + w + d_1 \text{ or alternately, } W(b_{x_i}) - W(b_{\bar{x}_i}) \leq d_1$$



(a) $C_{jl} = x_i$ (b) $C_{jl} = \bar{x_i}$

Figure 10: Values of lsp and e for each Conjunct

and

$$W(b_{\bar{x_i}}) + w \leq W(b_{x_i}) + w - d_2 \text{ or alternately, } W(b_{x_i}) - W(b_{\bar{x_i}}) \geq d_2$$

This is clearly impossible since $d_1 < d_2$.

In addition, for each conjunct C_j, we introduce one source s_{C_j} and 3 ABRs $b_{C_{j1}}$, $b_{C_{j2}}$ and $b_{C_{j3}}$. Depending on whether $C_{jl} = x_i$ or $C_{jl} = \bar{x_i}$, we assign values to lsp and e as shown in Figure 10. Thus, if $C_{jl} = x_i$, then $lsp(s_{x_i}, b_{C_{jl}}) = w + d_3$, $lsp(s_{C_j}, b_{C_{jl}}) = w$ and $lsp(s_{C_j}, b_{x_i}) = w - d_4$. Also, $e(s_{x_i}, b_{C_{jl}}) = e(s_{C_j}, b_{x_i}) = 100$ and $e(s_{C_j}, b_{C_{jl}}) = 0$. Note that $d_3 < d_4$. Thus, using an argument similar to the one used earlier, we can show that in any assignment W, it cannot be the case that both $B(s_{x_i}, W) = b_{x_i}$ and $B(s_{C_j}, W) = b_{C_{jl}}$.

Finally, note that for the remaining source and ABR pairs, lsp is set to ∞ and e is set to 100. For example, for distinct literals x_i, x_j, $lsp(s_{x_i}, b_{x_j}) = \infty$ and $e(s_{x_i}, b_{x_j}) = 100$. Similarly, if C_j does not contain literal x_i, then $lsp(s_{C_j}, b_{x_i}) = \infty$ and $e(s_{C_j}, b_{x_i}) = 100$. Also, the d_i's are related as follows: $d_1 < d_3 - 1$ and $d_2 > 2 * d_4 + 1$.

We are now in a position to show the following: there exists a weight assignment W such that $\sum_{s \in \mathcal{S}} e(s, B(s, W)) \leq m$ if and only if there exists an assignment of truth values to the literals such that the 3-SAT formula is satisfiable (note that m is the number of literals).

only if: Suppose W is the weight assignment function such that $\sum_{s \in \mathcal{S}} e(s, B(s, W)) \leq m$. As discussed earlier, since $d_1 < d_2$, for each literal x_i, both $B(s_{x_i}, W) = b_{x_i}$ and $B(s_{\bar{x_i}}, W) = b_{\bar{x_i}}$ are not possible. Thus, the two sources corresponding to each literal x_i contribute at least 1 to $\sum_s e$. Further, since there are m literals and $\sum_s e \leq m$, it follows that for the two ABRs selected for s_{x_i} and $s_{\bar{x_i}}$, the sum of the errors must be 1. Thus, either (1) $B(s_{x_i}, W) = b_{x_i}$ and $B(s_{\bar{x_i}}, W) = b_{x_i}$, or (2) $B(s_{x_i}, W) = b_{\bar{x_i}}$ and $B(s_{\bar{x_i}}, W) = b_{\bar{x_i}}$. Also, since $\sum_s e \leq m$, it must be the case that for each source s_{C_j}, one of the ABRs $b_{C_{j1}}$, $b_{C_{j2}}$ or $b_{C_{j2}}$ must be chosen (since these are the only ABRs for whom the error is 0). Suppose $b_{C_{jl}}$ is chosen. Then, if $C_{jl} = x_i$, it cannot be the case that $B(s_{x_i}, W) = b_{x_i}$ and if $C_{jl} = \bar{x_i}$, it cannot be the case that $B(s_{\bar{x_i}}, W) = b_{\bar{x_i}}$ (since $d_3 < d_4$).

Consider the following assignment of truth values to each literal x_i: if $B(s_{x_i}, W) = b_{x_i}$, then x_i is assigned false and if $B(s_{\bar{x_i}}, W) = b_{\bar{x_i}}$, then x_i is assigned true. This assignment of truth values is consistent since as shown earlier, both $B(s_{x_i}, W) = b_{x_i}$ and $B(s_{\bar{x_i}}, W) = b_{\bar{x_i}}$ are not possible. Further, we can show that every conjunct C_j is true under the assignment. Suppose $b_{C_{jl}}$ is the ABR chosen for source s_{C_j}. Then, if $C_{jl} = x_i$, it must be the case that $B(s_{\bar{x_i}}, W) = b_{\bar{x_i}}$ and so x_i is assigned true. Similarly, if $C_{jl} = \bar{x_i}$, it must be the case that $B(s_{x_i}, W) = b_{x_i}$ and so x_i is assigned false.

if: Suppose there exists an assignment of truth values to literals $x_1, \ldots, x_m$ such that every conjunct is satisfiable. We show that there exists an assignment of weights such that $\sum_{s \in \mathcal{S}} e(s, B(s, W)) \leq m$. Specifically, we show that there is an assignment of weights such that the following ABRs are selected for the two sources s_{x_i} and $s_{\bar{x}_i}$ corresponding to each literal x_i and the source s_{C_j} for each clause C_j.

1. If x_i is true: $B(s_{x_i}, W) = b_{\bar{x}_i}$ and $B(s_{\bar{x}_i}, W) = b_{\bar{x}_i}$.

2. If x_i is false: $B(s_{x_i}, W) = b_{x_i}$ and $B(s_{\bar{x}_i}, W) = b_{x_i}$.

3. If C_{jl} is true: $B(s_{C_j}, W) = b_{C_{jl}}$.

Clearly, since for the first two cases, the sum of the errors due to the choice of ABRs is 1, and $e(s_{C_j}, b_{C_{jl}}) = 0$, the above-mentioned selection ensures that $\sum_{s \in \mathcal{S}} e(s, B(s, W)) \leq m$. (Note that for each conjunct C_j at least one C_{jl} is true.) In the following, we show that there exists an assignment of weights to the various ABRs that ensures the selection of ABRs for sources described above. Note that for a source s, we do not need to concern ourselves with ABRs b for whom $lsp(s, b) = \infty$ since $B(s, W)$ can never be b.

We first consider the weights for the two ABRs b_{x_i} and $b_{\bar{x}_i}$ corresponding to literal x_i. If x_i is true, then we set $W(b_{x_i}) = \infty$ and $W(b_{\bar{x}_i}) = 1$. This ensures that ABR b_{x_i} is not selected for any source. On the other hand, if x_i is false, then we set $W(b_{x_i}) = d_4 + 1$ and $W(b_{\bar{x}_i}) = \infty$. This ensures that ABR $b_{\bar{x}_i}$ is not selected for any source. Next, we consider the weights for the three ABRs $b_{C_{jl}}$ corresponding to conjunct C_j. If C_{jl} is true, then $W(b_{C_{jl}}) = 0$; else, $W(b_{C_{jl}}) = \infty$.

In the following, we show that the above assignment of weights results in the desired selection of ABRs for sources.

1. x_i is true: For source $s_{\bar{x}_i}$, b_{x_i} cannot be selected since $W(b_{x_i}) = \infty$. Similarly, if for some conjunct C_j, $C_{jl} = \bar{x}_i$, then since $W(C_{jl}) = \infty$, $b_{C_{jl}}$ cannot be selected for $s_{\bar{x}_i}$ either. Thus, $B(s_{\bar{x}_i}, W) = b_{\bar{x}_i}$.

   For source s_{x_i}, b_{x_i} cannot be selected since $W(b_{x_i}) = \infty$. However, if for some conjunct C_j, $C_{jl} = x_i$, then since $W(b_{C_{jl}}) = 0$, if ABR $b_{\bar{x}_i}$ is to be selected over ABR $b_{C_{jl}}$ for s_{x_i}, then we need to show that the following inequality holds:

   $$W(b_{\bar{x}_i}) + w + d_1 < W(b_{C_{jl}}) + w + d_3$$

   Since $W(b_{\bar{x}_i}) = 1$, $W(b_{C_{jl}}) = 0$ and $d_1 < d_3 - 1$, the above equation is true and thus $B(s_{x_i}, W) = b_{\bar{x}_i}$.

2. x_i is false: For source s_{x_i}, $b_{\bar{x}_i}$ cannot be selected since $W(b_{\bar{x}_i}) = \infty$. Similarly, if for some conjunct C_j, $C_{jl} = x_i$, then since $W(C_{jl}) = \infty$, $b_{C_{jl}}$ cannot be selected for s_{x_i} either. Thus, $B(s_{x_i}, W) = b_{x_i}$.

   For source $s_{\bar{x}_i}$, $b_{\bar{x}_i}$ cannot be selected since $W(b_{\bar{x}_i}) = \infty$. However, if for some conjunct C_j, $C_{jl} = \bar{x}_i$, then since $W(b_{C_{jl}}) = 0$, if ABR b_{x_i} is to be selected over ABR $b_{C_{jl}}$ for $s_{\bar{x}_i}$, then we need to show that the following inequality holds:

   $$W(b_{x_i}) + w - d_2 < W(b_{C_{jl}}) + w - d_4$$

   Since $W(b_{x_i}) = d_4 + 1$, $W(b_{C_{jl}}) = 0$ and $2 * d_4 + 1 < d_2$, the above equation is true and thus $B(s_{\bar{x}_i}, W) = b_{x_i}$.

3. C_{jl} is true: In order to show that ABR $b_{C_{jl}}$ is selected for source s_{C_j}, we need to ensure that neither ABR b_{x_i} nor $b_{\bar{x}_i}$ corresponding to literal x_i can be selected for s_{C_j}. Thus, we need the following two equations to hold:

   $$W(b_{C_{jl}}) + w < W(b_{x_i}) + w - d_4$$

   and

   $$W(b_{C_{jl}}) + w < W(b_{\bar{x}_i}) + w + d_3$$

   Since $W(b_{C_{jl}}) = 0$, $W(b_{x_i}) \geq d_4 + 1$ and $W(b_{\bar{x}_i}) \geq 1$, the above equations are true and thus $B(s_{C_j}, W) = b_{C_{jl}}$.

Thus, there exists an assignment of weights to ABRs such that $\sum_{s \in \mathcal{S}} e(s, B(s, W)) \leq m$. ∎