IP Lookup By Binary Search On Prefix Length [*]

Kun Suk Kim & Sartaj Sahni
Department of Computer and Information Science and Engineering
University of Florida, Gainesville, FL 32611
{kskim, sahni}@cise.ufl.edu

Abstract

Waldvogel et al. [9] have proposed a collection of hash tables (CHT) organization for an IP router table. Each hash table in the CHT contains prefixes of the same length together with markers for longer-length prefixes. IP lookup can be done with $O(\log l_{dist})$ hash-table searches, where l_{dist} is the number of distinct prefix-lengths (also equal to the number of hash tables in the CHT). Srinivasan and Varghese [8] have proposed the use of controlled prefix-expansion to reduce the value of l_{dist}. The details of their algorithm to reduce the number of lengths are given in [7]. The complexity of this algorithm is $O(nW^2)$, where n is the number of prefixes, and W is the length of the longest prefix. The algorithm of [7] does not minimize the storage required by the prefixes and markers for the resulting set of prefixes. We develop an algorithm that minimizes storage requirement but takes $O(nW^3 + kW^4)$ time, where k is the desired number of distinct lengths. Also, we propose improvements to the heuristic of [7].

Keywords: Packet routing, router tables, longest-prefix matching, controlled prefix expansion, binary search on length, dynamic programming.

1 Introduction

An Internet router table is a set of tuples of the form (p, a), where p is a binary string whose length is at most W ($W = 32$ for IPv4 destination addresses and $W = 128$ for IPv6), and a is an output link (or next hop). When a packet with destination address A arrives at a router, we are to find the pair (p, a) in the router table for which p is a longest matching prefix of A (i.e., p is a prefix of A and there is no longer prefix q of A such that (q, b) is in the table). Once this pair is determined, the packet is sent to ouput link a. The speed at which the router can route packets is limited by the time it takes to perform this table lookup for each packet.

Several solutions for the IP lookup problem (i.e., finding the longest matching-prefix) have been proposed. These solutions are surveyed in [3, 6]. In this paper, we focus on the collection of hash tables (CHT) scheme of Waldvogel et al. [9].

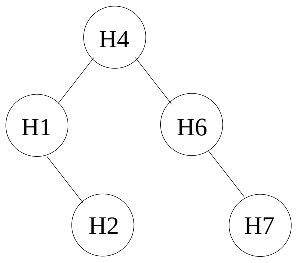
[*]This work was supported, in part, by the National Science Foundation under grant CCR-9912395.

Let P be the set of prefixes in a router table, and let P_i be the subset of P comprised of prefixes whose length is i. In the scheme of Waldvogel et al. [9], we maintain a hash table H_i for every P_i that is not empty. H_i includes the prefixes of P_i as well as markers for prefixes in $\cup_{i<j\leq W} P_j$. Each marker m in H_i is i-bits long and $m.lmp$ is the longest matching-prefix for m. Consider the prefix set $P = \{P1, ..., P6\}$ of Figure 1(a). The prefixes of P have 5 distinct lengths 1, 2, 4, 6, and 7. So, the CHT of [9] will comprise H_1, H_2, H_4, H_6, and H_7. Given a destination address d, the longest matching-prefix, $lmp(d)$ is found by seraching the H_is using a binary search. Suppose that the binary search follows a path as determined by the binary tree of Figure 1(b). That is, if the first 4 bits of d correspond to a prefix in H_4, this prefix becomes the longest matching-prefix found so far and the search continues to H_6; if the first four bits of d correspond to a marker m in H_4, then $m.lmp$ becomes the longest-matching prefix found so far and the search continues to H_6; otherwise, the search continues to H_1. The quest for $lmp(d)$ examines at most 3 hash tables in our example. When the number of distinct lengths is l_{dist}, the number of hash tables examined is $O(\log l_{dist})$.

For the described search to work correctly, H_4 must have markers for P_5 and P_6; H_1 for P_2; and H_6 for P_6. H_1, for example, will include $P1$ and $P2$ plus the marker $1*$ for $P3$ (actually, since $P2 = 1*$, the marker isn't needed); while H_4 will include $P4$ plus the marker 1001* for $P5$ and $P6$. The lmp value for the marker 1001* is $P3$.

P1 =0*
P2 =1*
P3 =10*
P4 =1000*
P5 =100100*
P6 =1001001*

(a) Prefixes



(b) Tree for binary search

00* *(P1a)*
01* *(P1b)*
10* *(P3)*
11* *(P2a)*
1000* *(P4)*
1001000* *(P5a)*
1001001* *(P6)*

(c) Expanded prefixes

Figure 1: Controlled prefix expansion

Srinivasan and Varghese [8] have proposed the use of controlled prefix-expansion to reduce the number of distinct lengths and hence the number of hash tables in the CHT. By reducing the number of hash tables in the CHT, the worst-case number of hash tables searched in the quest for $lmp(d)$ may be reduced.

Prefix expansion [8] replaces a prefix of length u with 2^{v-u} prefixes of length v, $v > u$. The new prefixes are obtained by appending all possible bit sequences of length $v-u$ to the prefix being expanded. So, for example, the prefix 1* may be expanded to the length 2 prefixes 10* and 11* or to the length 3 prefixes 100*, 101*, 110*, and 111*. In case an expanded prefix is already present in the prefix set of the router table, it is dominated by the existing prefix (the expanded prefix 10* represents a shorter original prefix 1* that cannot be used to match destination addresses that begin with 10 when longest-prefix matching is used) and so is discarded. So, if we expand $P2 = 1*$ in our collection of Figure 1(a) to length 2, the expaned prefix $P2b = 10*$ is dominated by $P3 = 10*$. Figure 1(c) shows the prefixes that result when the length 1 prefixes of Figure 1(a) are expanded to length 2 and the length 6 prefix is expanded to length 7. You may verify that $lmp(d)$ is the same for all d regardless of whether we use the prefix set of Figure 1(a) or (c) (when the latter set is used, we need to map back to the original prefix from which an expanded prefix came). Since, the prefixes of Figure 1(c) have only 3 distinct lengths, the corresponding CHT has only 3 hash tables and may be searched for $lmp(d)$ with at most 2 hash-table searches. Hence, the CHT scheme results in faster lookup when the prefixes of Figure 1(c) are used than when those of Figure 1(a) are used. [8, 4, 5] use prefix expansion to improve the lookup performance of trie-representations of router tables.

When reducing the number of distinct lengths from u to v, the choice of the target v lengths affects the number of markers and prefixes that have to be stored in the resulting CHT but not the number of hash tables, which is always v. Although the number of target lengths may be determined from the expected number of packets to be processed per second and the performance characteristics of the computer to be used for this purpose, the target lengths are determined so as to minimize the storage requirements of the CHT. Consequently, Srinivasan and Varghese [8] formulated the following optimization problem.

Exact Collection of Hash Tables Optimization Problem (ECHT)

Given a set P of n prefixes and a target number of distinct lengths k, determine target lengths $l_1, ..., l_k$ such that the storage required by the prefixes and markers for the prefix set $expansion(P)$ obtained from P by prefix expansion to the determined target lengths is minimum.

When P and k are not implicit, we use the notation $ECHT(P, k)$. For simplicity, Srinivasan [7] assumes that the strorage required by the prefixes and markers for the prefix set $expansion(P)$ equals the number of prefixes and markers. We make the same assumption in this paper. Srinivasan [7] provides an $O(nW^2)$-time heuristic for ECHT. We first show, in Section 2, that the heuristic of Srinivasan [7] may be implemented so that its complexity is $O(nW + kW^2)$ on practical prefix-sets. Then, in Section 3, we provide an $O(nW^3 + kW^4)$-time algorithm for ECHT. In Section 4, we formulate an alternative version ACHT of the ECHT problem. In this alternative version, we are to find at most k distinct target lengths to minimize storage rather than exactly k target lengths. The ACHT problem also may be solved in $O(nW^3 + kW^4)$ time. In Section 5, we propose a reduction in the search range used by the heuristic of [7]. The proposed range reduction reduces the run time by more than 50% exclusive of the preprocessing time. The reduced-range heuristic generates the same results on our benchmark prefix data-sets as are generated by the full-range heuristic of [7]. A more accurate cost estimator than is used in the heuristic of [7] is proposed in Section 6. Experimental results highlighting the relative performance of the various algorithms and heuristics for ECHT and ACHT are presented in Section 7.

2 The Heuristic of Srinivasan [7]

The ECHT heuristic of Srinivasan [7] uses the following definitions:

1. $ExpansionCost[i, j]$

   This is the number of distinct prefixes that result when all prefixes $P_i \in P$ whose length, $|P_i|$, is such that $i \leq |P_i| < j$ are expanded to length j. For example, when $P = \{0*, 1*, 01*, 100*\}$, $ExpansionCost[1, 3] = 8$ (note that 0* and 1* contribute 4 prefixes each; 01* contributes none because its expanded prefixes are included in the expanded prefixes of 0*).

2. $Entries[j]$

   This is the maximum number of markers in H_j (should j be a target length) plus the number of prefixes in P whose length is j. Srinivasan [7] uses "maximum number of markers" in the definition of $Entries[j]$ rather than the exact number of markers because of the reported difficulty in computing this latter quantity.

3. $T[j, r]$

   This is an upper bound on the storage required by the optimal solution to $ECHT(Q, r)$, where $Q \subseteq P$ comprises all prefixes of P whose length is at most j; the optimal solution to $ECHT(Q, r)$ is required to contain markers, as necessary, for prefixes of P whose length exceeds j.

Srinivasan [7] provides the following dynamic programming recurrence for $T[j, r]$.

$$T[j, r] = Entries[j] + \min_{m \in \{r-1 \ldots j-1\}} \{T[m, r-1] + ExpansionCost[m+1, j]\} \quad (1)$$

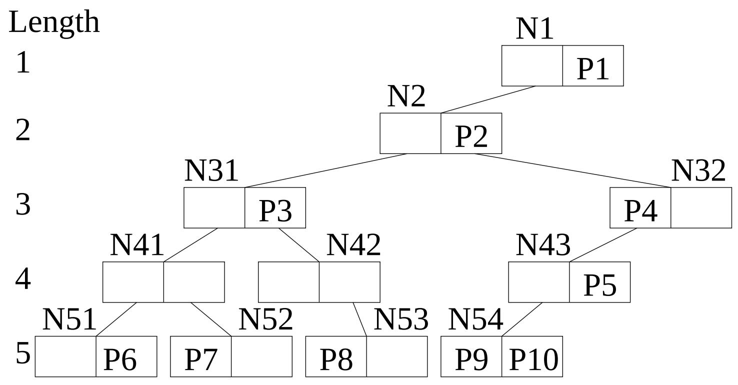
$$T[j, 1] = Entries[j] + ExpansionCost[1, j] \tag{2}$$

We may verify the correctness of Equations 1 and 2. When $r = 1$, there is only 1 target length and this length is no more than j. When Q has a prefix whose length is j, then j must be the target length. In this case, the number of expanded prefixes is at most $ExpansionCost[1, j]$ plus the number of prefixes whose length is j. So, the number of prefixes and markers is at most $Entries[j] + ExpansionCost[1, j]$. When Q has no prefix whose length is j, the optimal target length is the largest l, $l < j$ such that Q has a prefix whose length is l. In this case, $Entries[l] + ExpansionCost[1, l] \le Entries[j] + ExpansionCost[1, j]$ is an upper bound on the number of prefixes and markers.

To compute $ExpansionCost$ and $Entries$, a 1-bit trie [1] is used. Figure 2 shows a prefix set and its corresponding 1-bit trie. Notice that nodes at level i (the root is at level 0) of the 1-bit trie store prefixes whose length is $i + 1$. Srinivasan [7] states how $ExpansionCost[i, j], 1 \le i < j \le W$ may be computed in $O(nW^2)$ time using a 1-bit trie for P. Sahni and Kim [4] have observed that, for practical prefix sets, the 1-bit trie has $O(n)$ nodes. So, by performing a postorder traversal of the 1-bit trie, $ExpansionCost[i, j], 1 \le i < j \le W$ may be computed in $O(nW)$ time (note that $n > W$). Details of this process are provided in Section 3.1 where we show how a closely related function may be computed.

Prefixes
P1=1*
P2=01*
P3=001*
P4=010*
P5=0101*
P6=00001*
P7=00010*
P8=00110*
P9=01000*
P10=01001*

(a) A prefix set



(b) Corresponding 1-bit trie

Figure 2: Prefixes and corresponding 1-bit trie

For $Entries[j]$, Srinivasan [7] proposes counting the number of prefixes stored in level $j - 1$ of the 1-bit trie and the number of (non-null) pointers to nodes at level j (the number of pointers actually equals the number of nodes). The former gives the number of prefixes whose length is j and the latter gives the maximum number of markers needed for the longer-length prefixes.

Suppose that m and j are target lengths and that no l, $m < l < j$ is a target length. The actual number of prefixes and markers in H_j may be considerably less than $Entries[j] + ExpansionCost[m+1, j]$ for the following reasons.

1. An expanded prefix counted in $ExpansionCost[m+1, j]$ may be indentical to a prefix in P whose length is j.

2. Some of the prefixes in P whose length is more than j may not need to leave a marker in H_j because their length is not on any binary search (sub)path that is preceded by the length j. For example, for the binary search described by Figure 1(b), H_1 needs markers only for prefixes in H_2; not for those in H_4, H_6, and H_7. However, $Entries[1]$ accounts for markers needed by prefixes in H_2 as well as those in H_4, H_6, and H_7.

3. $Entries[j]$ doesn't account for the fact that a marker may be identical to a prefix in which case the storage count for the marker and the prefix together should be 1 and not 2. For example, in Figure 2(b), the marker corresponding to the non-null pointer to node N42 is identical to the prefix P3 and that for the non-null pointer to N43 is identical to P4. So, we can safely reduce the value of $Entries[3]$ from 5 to 3. Note also that if the target lengths for the example of Figure 2(a) are 1, 3, and 5, then the number of prefixes and markers in H_3 is 4. However, $ExpansionCost[2, 3] + Entries[3] = 2 + 5 = 7$.

Exclusive of the time needed to compute $ExpansionCost$ and $Entries$, the complexity of computing $T[W, k]$ and the target k lengths using Equations 1 and 2 is $O(kW^2)$ [7]. So, the overall complexity is $O(nW^2)$ (note that $n \geq k$). As noted above, we may reduce the time required to compute $ExpansionCost$ on practical prefix-sets by performing a postorder traversal of the 1-bit trie. Hence, for practical prefix-sets, the overall run time is $O(nW + kW^2)$.

3 An Optimal-Storage Algorithm

As noted in Section 2, the algorithm of Srinivasan [7] is only a heuristic for ECHT. Since $T[W, k]$ is only an upper bound on the cost of an optimal solution for $ECHT(P, k)$, there is no assurance that the determined target lengths actually result in an optimal or close-to-optimal solution to $ECHT(P, k)$. In this section, we develop an algorithm to determine the storage cost of an optimal solution to $ECHT(P, k)$. The algorithm is easily extended to determine the target lengths that yield this optimal storage cost.

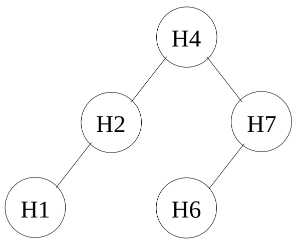


Figure 3: Alternative binary tree for binary search

Like the heuristic of Srinivasan [7], our algorithm uses dynamic programming. However, we modify the definition of expansion cost and introduce an accurate way to count the number of markers.

Although the heuristic of Srinivasan [7] is insensitive to the shape of the binary tree that describes the binary search, the optimal-storage algorithm cannot be insensitive to this shape. To see this, notice that the binary tree of Figure 1(b) corresponds to the traditional way to program a binary search. In this, if *low* and *up* define the current search range, then the next comparison is made at $mid = \lfloor (low + up)/2 \rfloor$. If instead, we were to make the next comparison at $mid = \lceil (low + up)/2 \rceil$, the search is described by the binary tree of Figure 3. When a binary search is performed according to this tree, only H_4 need have markers. The markers in H_4 are the same regardless of whether we use $mid = \lfloor (low + up)/2 \rfloor$ or $mid = \lceil (low + up)/2 \rceil$. By using the latter definition of *mid*, we eliminate markers from all remaining hash tables. In our development of the optimal-storage algorithm, we assume that $mid = \lceil (low + up)/2 \rceil$ is used. Our development is easily altered for the case when $mid = \lfloor (low + up)/2 \rfloor$ is used.

3.1 Expansion Cost

Define $EC(i, j)$, $1 \leq i \leq j \leq W$, to be the number of distinct prefixes that result when all prefixes $P_i \in P$ whose length, $|P_i|$, is such that $i \leq |P_i| \leq j$ are expanded to length j. Note that $EC(i, i)$ is the number of prefixes in P whose length is i.

We may compute EC by traversing the 1-bit trie for P in a postorder fashion. Each node x at level $i - 1$ of the trie maintains a local set of $EC(i, j)$ values, $LEC(i, j)$, which is the expansion cost measured relative to the prefixes in the subtree of which x is root. Some of the cases for the computation of $x.LEC(i, j)$ are given below.

1. $x.LEC(i, i)$ equals the number of prefixes stored in node x. For example, for node N1 of Figure 2(a), $LEC(1, 1) = 1$ and for node N54, $LEC(5, 5) = 2$. For the remaining cases, assume $i < j$.

2. If x has a prefix in its left data field (e.g., the prefix in the left data field of node N32 is P4) and also has one in its right data field, then $x.LEC(i, j) = 2^{j-i+1}$.

3. If x has no prefixes (e.g., nodes N41 and N42) and x has non-null left and right subtrees, then $x.LEC(i,j) = x.leftChild.LEC(i+1,j) + x.rightChild(i+1,j)$.

4. If x has a right prefix and a non-null left subtree, then $x.LEC(i,j) = x.leftChild.LEC(i+1,j) + 2^{j-i}$.

The remaining cases are similar to those given above. One may verify that $EC(i,j)$ is just the sum of the $LEC(i,j)$ values taken over all nodes at level $i-1$ of the trie. Figure 4 gives the LEC and EC values for the example of Figure 2. In this figure, $LEC51$, for example, refers to the LEC values for node N51.

LEC51[5,j]	j = 5
	1

LEC52[5,j]	j = 5
	1

LEC53[5,j]	j = 5
	1

LEC54[5,j]	j = 5
	2

LEC41[4,j]	j = 4	5
	0	2

LEC42[4,j]	j = 4	5
	0	1

LEC43[4,j]	j = 4	5
	1	4

LEC31[3,j]	j = 3	4	5
	1	2	6

LEC32[3,j]	j = 3	4	5
	1	2	4

LEC2[2,j]	j = 2	3	4	5
	1	3	6	14

LEC1[1,j]	j = 1	2	3	4	5
	1	3	7	14	30

(a) LEC values

EC[i,j]	j = 1	2	3	4	5
i = 1	1	3	7	14	30
2		1	3	6	14
3			2	4	10
4				1	7
5					5

(b) EC values

Figure 4: LEC and EC values for Figure 2

Since a 1-bit trie for n prefixes may have $O(nW)$ nodes, we may compute all EC values in $O(nW^2)$ time by computing the LEC values as above and summing up the computed LEC values. A postorder traversal suffices for this. As noted in [4], the 1-bit tries for practical prefix sets have $O(n)$ nodes. Hence, in practice, the EC values take only $O(nW)$ time to compute.

3.2 Number of Markers

Define $MC(i,j,m)$, $1 \le i \le j \le m \le W$, to be the number of markers in H_j under the following assumptions.

1. The prefix set comprises only those prefixes of P whose length is at most m.

2. The target lengths include $i-1$ (for notational convenience, we assume that 0 is a trivial target length for which H_0 is always empty) and j but no length between $i-1$ and j. Hence, prefixes

whose length is i, $i+1$, $\cdots$, or j are expanded to length j. Only prefixes whose length is between $j+1$ and m may leave a marker in H_j.

For $MC(2,4,5)$ (Figure 2), P6 through P10 may leave markers in H_4. The candidate markers are obtained by considering only the first four bits of each of these prefixes. Hence, the candidate markers are 0000*, 0001*, 0011*, and 0100*. However, since the next smaller target length is 1, P2, P3, and P4 will leave a prefix in H_4. The prefixes in H_4 are 0100*, 0101*, 0110*, 0111*, 0010*, and 0011*. So, of the candidate markers, only 0000* and 0001* are different from the prefixes in H_4. Therefore, the marker count $MC(2,4,5)$ is 2.

We may compute all $MC(i,j,m)$ values in $O(nW^3)$ time ($O(nW^2)$ for practical prefix sets) using a local function LMC in each node of the 1-bit trie and a postorder traversal. The method is very similar to that described in Section 3.1 for the computation of all EC values. Figure 5 shows the LMC and MC values for our example of Figure 2.

LMC51[5,j,k]	k = 5
j = 5	0

LMC52[5,j,k]	k = 5
j = 5	0

LMC53[5,j,k]	k = 5
j = 5	0

LMC54[5,j,k]	k = 5
j = 5	0

LMC41[4,j,k]	k = 4	5
j = 4	0	2
5		0

LMC42[4,j,k]	k = 4	5
j = 4	0	1
5		0

LMC43[4,j,k]	k = 4	5
j = 4	0	1
5		0

LMC31[3,j,k]	k = 3	4	5
j = 3	0	0	1
4		0	2
5			0

LMC32[3,j,k]	k = 3	4	5
j = 3	0	0	0
4		0	0
5			0

LMC1[1,j,k]	k = 1	2	3	4	5
j = 1	0	1	1	1	1
2		0	1	1	1
3			0	0	1
4				0	2
5					0

LMC2[2,j,k]	k = 2	3	4	5
j = 2	0	1	1	1
3		0	0	1
4			0	2
5				0

(a) LMC values

MC[1,j,k]	k = 1	2	3	4	5
j = 1	0	1	1	1	1
2		0	1	1	1
3			0	0	1
4				0	2
5					0

MC[2,j,k]	k = 2	3	4	5
j = 2	0	1	1	1
3		0	0	1
4			0	2
5				0

MC[3,j,k]	k = 3	4	5
j = 3	0	0	1
4		0	2
5			0

MC[4,j,k]	k = 4	5
j = 4	0	4
5		0

MC[5,j,k]	k = 5
j = 5	0

(b) MC values

Figure 5: LMC and MC values for Figure 2

3.3 Algorithm for ECHT

Let $Opt(i,j,r)$ by the storage requirement of the optimal solution to $ECHT(P,r)$ under the following restrictions.

1. Only prefixes of P whose length is between i and j are considered.

2. Exactly r target lengths are used.

3. j is one of the target lengths (even if there is no prefix whose length is j).

Let l_{max}, $l_{max} \leq W$, be the length of the longest prefix in P. We see that $Opt(1, l_{max}, k)$ is the storage requirement of the optimal solution to $ECHT(P, k)$.

When $r = 1$, there is exactly one target length, j. So, all prefixes must be expanded to this length and there are no markers. Therefore,

$$Opt(i, j, 1) = EC(i, j), i \leq j \tag{3}$$

When $r = 2$, one of the target lengths is j and the other, say m, lies between i and $j - 1$. Because we assume $mid = \lceil (low + up)/2 \rceil$, the first search is made in H_j and the second in H_m. Consequently, neither H_j nor H_m has any markers. H_j (H_m) includes prefixes resulting from the expansion of prefixes of P whose length is between $m + 1$ and j (i and m). So,

$$Opt(i, j, 2) \quad = \quad \min_{i \leq m < j} \{EC(i, m) + EC(m + 1, j)\}, i < j \tag{4}$$

Consider the case $r > 2$. Let the r target lengths be $l_1 < l_2 < \cdots < l_r$. Suppose that the $mid = \lceil (1+r)/2 \rceil$th target length is v. Let $u-1$ be the largest target length such that $u-1 < v$. The first search of the binary search is done in H_v. The number of prefixes and markers in H_v is $EC(u, v) + MC(u, v, j)$. Additionally, the $mid - 1 = \lfloor (r-1)/2 \rfloor$ target lengths that are less than v define an optimal $(mid - 1)$-target-length solution for prefixes whose length is between i and $u-1$ subject to the constraint that $u-1$ is a target length (notice that there are no markers in this solution for prefixes whose length exceeds $u - 1$) and the $r - m = \lfloor (r-1)/2 \rfloor$ target lengths greater than v define an optimal $(r - m)$-target-length solution for prefixes whose length is between $v + 1$ and j subject to the constraint that j is a target length. Hence, we obtain the following recurrence for $Opt(i, j, r)$.

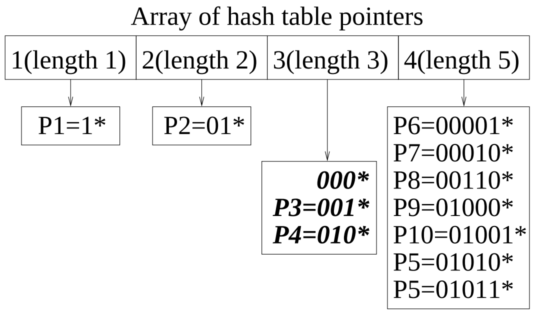
$$\begin{aligned} Opt(i, j, r) \quad &= \min_{i+\lceil (r-1)/2 \rceil \leq u \leq v \leq j - \lfloor (r-1)/2 \rfloor} \{Opt(i, u - 1, \lceil (r-1)/2 \rceil) \\ &+ \; Opt(v + 1, j, \lfloor (r-1)/2 \rfloor) + EC(u, v) + MC(u, v, j)\}, 3 \leq r \leq j - i + 1 \end{aligned} \tag{5}$$

Using Equations 3–5 to compute $Opt(1, 5, 4)$ for the example of Figure 2, we get

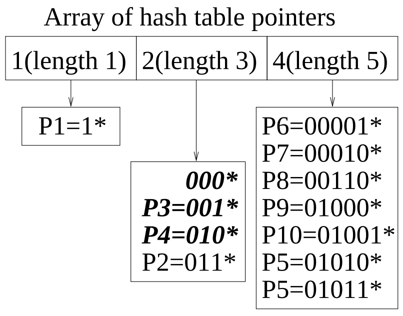
$$Opt(1, 5, 4) \quad = \quad \min_{3 \leq u \leq v \leq 4} \{Opt(1, u - 1, 2) + Opt(v + 1, 5, 1) + EC(u, v) + MC(u, v, j)\}$$

$$
\begin{aligned}
&= \min\{Opt(1,2,2) + Opt(4,5,1) + EC(3,3) + MC(3,3,5),\\
&\quad Opt(1,2,2) + Opt(5,5,1) + EC(3,4) + MC(3,4,5),\\
&\quad Opt(1,3,2) + Opt(5,5,1) + EC(4,4) + MC(4,4,5)\}\\
&= \min\{EC(1,1) + EC(2,2) + EC(4,5) + EC(3,3) + MC(3,3,5),\\
&\quad EC(1,1) + EC(2,2) + EC(5,5) + EC(3,4) + MC(3,4,5),\\
&\quad \min\{EC(1,1) + EC(2,3), EC(1,2) + EC(3,3)\} + EC(5,5) + EC(4,4) + MC(4,4,5)\}\\
&= \min\{1+1+7+2+1, 1+1+5+4+2, \min\{1+3, 3+2\} + 5 + 1 + 4\}\\
&= \min\{12, 13, 14\} = 12.
\end{aligned}
$$

From the above computations, we see that the optimal expansion lengths are 1, 2, 3, and 5. Figure 6(a) shows the CHT structure that results when these four target lengths are used. The three markers are shown in boldface, two of these markers are also prefixes (P3 and P4). The storage cost is 12.



(a) 4 target lengths



(b) 3 target lengths

Figure 6: Optimal-storage CHTs for Figure 2

Complexity

To solve Equations 3–5 for $Opt(1, l_{max}, k)$, we need to compute $O(kW^2)$ $Opt(i,j,r)$ values. Each of these values may be computed in $O(W^2)$ time from earlier computed Opt values. Hence, exclusive of the time needed to compute EC and MC, the time to compute $Opt(1, l_{max}, k)$ is $O(kW^4)$. Adding in the time to compute EC and MC, we get $O(nW^3 + kW^4)$ as the overall time needed to solve the ECHT problem. Of course, on practical data sets, the time is $O(nW^2 + kW^4)$.

4 An Alternative Formulation

In the $ECHT(P, k)$ problem, we are to find exactly k target lengths for P that minimize the number of (expanded) prefixes and markers (i.e., minimize storage cost). Although k is determined by constraints on required lookup performance, the determined k is only an upper bound on the number of target lengths because using a smaller number of target lengths improves lookup performance. The $ECHT$ problem is formulated with the premise that using a smaller k will lead to increased storage cost, and so in the interest of conserving storage/memory while meeting the lookup performance requirement, we use the maximum permissible number of target lengths. However, this premise is not true. As an example, consider the prefix set $P = \{P1, P2, P3\} = \{0*, 00*, 010*\}$. The solution for $ECHT(P, 2)$ uses the target lengths 2 and 3; $P1$ expands to 00* and 01* but the 00* expansion is dominated by $P2$ and is discarded; no markers are stored in either H_2 or H_3; and the storage cost is 3. The solution for $ECHT(P, 3)$, on the other hand, uses the target lengths 1, 2, and 3; no prefixes are expanded; H_2 needs a marker 01* for $P3$; and the total storage cost is 4!

With this in mind, we formulate the $ACHT(P, k)$ problem in which we are to find at most k target lengths for P that minimize the storage cost. In case of a tie, the solution with the smaller number of target lengths is preferred, because this solution has a reduced average lookup for the preceding example, the solution to $ECHT(P, 3)$ is $\{1, 2, 3\}$, whereas the solution to $ACHT(P, 3)$ is $\{2, 3\}$. For the example of Figure 2, the solution to $ECHT(P, 4)$ is $\{1, 2, 3, 5\}$ resulting in a storage cost of 12; the solution to $ACHT(P, 4)$ is $\{1, 3, 5\}$ resulting in a storage cost that is also 12 (see Figure 6(b)).

The $ACHT$ problem may be solved in the same asymptotic time as needed for the $ECHT$ problem by first computing $Opt(i, j, r)$, $1 \le i < j \le l_{max}$, $1 \le r \le k$ and then finding the r for which $Opt(1, l_{max}, r)$ is minimum, $1 \le r \le k$.

5 A Reduced-Range Heuristic

We first adapt the $ECHT$ heuristic of Srinivasan [7] to the $ACHT$ problem. For this purpose, we define the function C, which is the $ACHT$ analog of T. To get the definition of C, simply replace $ECHT(Q, r)$ by $ACHT(Q, r)$ in the definition of T. Also, we use the same definitions for $ExpandedCost$ (now abbreviated to $ECost$) and $Entries$ as used in [7] (see Section 2). It is easy to see that $C(j, r) \le C(j, r-1), r > 1$.

A simple dynamic programming recurrence for C is:

$$C(j,r) = \mathit{Entries}(j) + \min_{m \in \{0 \ldots j-1\}} \{C(m, r-1) + \mathit{ECost}(m+1, j)\}, j > 0, r > 1 \tag{6}$$

$$C(0,r) = 0, C(j,1) = \mathit{Entries}(j) + \mathit{ECost}(1,j), j > 0 \tag{7}$$

To see the correctness of Equations 6 and 7, note that when $j > 0$, there must be at least one target length. If $r = 1$, then there is exactly one target length. This target length is at most j (the target length is j when there is at least one prefix of this length) and so $\mathit{Entries}(j) + \mathit{ECost}(1,j)$ is an upper bound on the storage cost. If $r > 1$, let m and s, $m < s$, be the two largest target lengths in the solution to $ACHT(P, r)$. m could be at any of the lengths 0 through $j - 1$; $m = 0$ would mean that there is only 1 target length. Hence the storage cost is bounded by $\mathit{Entries}(j) + C(m, r-1) + \mathit{ECost}(m+1, j)$. Since we do not know the value of m, we may minimize over all choices for m. $C(0, r) = 0$ is a boundary condition.

We may obtain an alternative recurrence for $C(j,r)$ in which the range of m on the right side is $r-1 \ldots j-1$ rather than $0 \ldots j-1$. First, we obtain the following dynamic programming recurrence for C:

$$C(j,r) = \min\{C(j, r-1), T[j,r]\}, r > 1 \tag{8}$$

$$C(j,1) = \mathit{Entries}(j) + \mathit{ECost}(1,j) \tag{9}$$

The rationale for Equation 8 is that the best CHT that uses at most r target lengths either uses at most $r - 1$ target lengths or uses exactly r target lengths. When at most $r - 1$ target lengths are used, the cost is bounded by $C(j, r-1)$, and when exactly r target lengths are used, the cost is bounded by $T[j,r]$, which is defined by Equation 1.

Let $U(j,r)$ be as defined below:

$$U(j,r) = \mathit{Entries}(j) + \min_{m \in \{r-1 \ldots j-1\}} \{C(m, r-1) + \mathit{ECost}(m+1, j)\}, j > 0, r > 1$$

From Equations 1 and 8 we obtain:

$$C(j,r) = \min\{C(j, r-1), U(j,r)\}, r > 1 \tag{10}$$

To see the correctness of Equation 10, note that for all j and r such that $r \leq j, T(j,r) \geq C(j,r)$. Furthermore,

$$\begin{aligned} Entries(j) &+ \min_{m \in \{r-1 \ldots j-1\}} \{T[m, r-1] + ECost(m+1, j)\} \\ &\geq Entries(j) + \min_{m \in \{r-1 \ldots j-1\}} \{C(m, r-1) + ECost(m+1, j)\} \\ &= U(j,r) \end{aligned} \tag{11}$$

Therefore, when $C(j, r-1) \leq U(j,r)$, Equations 8 and 10 compute the same value for $C(j,r)$. When $C(j, r-1) > U(j,r)$, it appears from Equation 11 that Equation 10 may compute a smaller $C(j,r)$ than is computed by Equation 8. However, this is impossible, because

$$\begin{aligned} C(j,r) &= Entries(j) + \min_{m \in \{0 \ldots j-1\}} \{C(m, r-1) + ECost(m+1, j)\} \\ &\leq Entries(j) + \min_{m \in \{r-1 \ldots j-1\}} \{C(m, r-1) + ECost(m+1, j)\} \end{aligned}$$

Therefore, the $C(j,r)$s computed by Equations 8 and 10 are equal.

In the remainder of this section, we use the reduced ranges $r-1 \ldots j-1$ for C. Heuristically, the range for m (in Equation 6) may be restricted to a range that is (often) considerably smaller than $r-1 \ldots j-1$. The narrower range we wish to use is $\max\{M(j-1, r), M(j, r-1), r-1\} \ldots j-1$, where $M(j,r), r > 1$, is the smallest m that minimizes

$$C(m, r-1) + ECost(m+1, j)$$

in Equation 6 (we assume that in case of a tie, the smaller value of m is selected). Although the use of this narrower range could result in different results from what we get using the range $r-1 \ldots j-1$, on our benchmark prefix sets, this doesn't happen. In the remainder of this section, we derive a condition Z on the 1-bit trie that, if satisfied, guarantees that the use of the narrower range yields the same results as when the range $r-1 \ldots j-1$ is used.

Let P be the set of prefixes represented by the 1-bit trie. Let $exp(i,j)$, $i \leq j$, be the set of distinct prefixes obtained by expanding the prefixes of P whose length is between i and $j-1$ to length j. Note that $exp(i,i) = \emptyset$ and that $|exp(i,j)| = ECost(i,j)$. We say that $exp(i,j)$ covers a length j prefix p of P iff $p \in exp(i,j)$. Let $n(i,j)$ be the number of length j prefixes in P that are not covered by a prefix of $exp(i,j)$. Note that $n(i,i)$ is the number of length i prefixes in P.

The condition Z that ensures that the use of the narrower range produces the same C values as when the range $r-1 \ldots r-1$ is used is

$$Z = ECost(a,j) - ECost(b,j) \geq 2(n(b,j) - n(a,j))$$

where, $0 < a < b \leq j$.

Lemma 1 *For every 1-bit trie, (a) $ECost(i,j+1) \geq 2ECost(i,j), 0 < i \leq j$, and (b) $ECost(i,j) \geq ECost(i+1,j), 0 < i < j$.*

Proof (a)

$$\begin{aligned} ECost(i,j+1) &= |exp(i,j+1)| \\ &= 2[|exp(i,j)| + n(i,j)] \\ &= 2ECost(i,j) + 2n(i,j) \\ &\geq 2ECost(i,j) \end{aligned}$$

(b) Since, $exp(i+1,j) \subseteq exp(i,j)$, $ECost(i,j) = |exp(i,j)| \geq |exp(i+1,j)| = ECost(i+1,j)$. ■

Lemma 2 $\forall(j > 0, i < j)[Entries(j) + ECost(i,j) \leq Entries(j+1) + ECost(i,j+1)]$.

Proof By definition, $Entries(j)$ = number of prefixes of length j plus the number of nodes at level j of the trie (this latter number equals the number of pointers from level $j-1$ to level j). Since each length j prefix expands to 2 length $j+1$ prefixes, the first term in the sum for $Entries(j)$ is at most $ECost(i,j+1)/2$. Since the subtree rooted at each level j node contains at least one prefix, the second term in the sum for $Entries(j)$ is at most $Entries(j+1)$. So,

$$Entries(j) \leq ECost(i,j+1)/2 + Entries(j+1)$$

From Lemma 1(a), $ECost(i,j) \leq ECost(i,j+1)/2$. So, $Entries(j) + ECost(i,j) \leq ECost(i,j+1) + Entries(j+1)$. ■

Lemma 3 $\forall(j > 0, r > 0)[C[j,r] \leq C[j+1,r]]$.

Proof First, consider the case when $r = 1$. From Equation 7, we get $C(j,1) = Entries(j) + ECost(1,j)$ and $C(j+1,1) = Entries(j+1) + ECost(1,j+1)$. From Lemma 2, $Entries(j) + ECost(1,j) \leq Entries(j+1) + ECost(1,j+1)$. Hence, $C(j,1) \leq C(j+1,1)$.

Next, consider the case $r > 1$. From the definition of $M(j,r)$, it follows that

$$C(j+1,r) = \mathit{Entries}(j+1) + C(b,r-1) + \mathit{ECost}(b+1,j+1),$$

where $0 \le b = M(j+1,r) \le j$. When $b < j$, using Equation 6 and Lemma 1, we get

$$\begin{aligned} C(j,r) &\le \mathit{Entries}(j) + C(b,r-1) + \mathit{ECost}(b+1,j) \\ &\le \mathit{Entries}(j+1) + C(b,r-1) + \mathit{ECost}(b+1,j+1) \\ &= C(j+1,r). \end{aligned}$$

When $b = j$,

$$C(j+1,r) = \mathit{Entries}(j+1) + C(j,r-1) + \mathit{ECost}(j+1,j+1) \ge C(j,r-1).$$

■

The remaining lemmas of this section assume that Z is true.

Lemma 4 $\mathit{ECost}(a,j+1) - \mathit{ECost}(b,j+1) \ge \mathit{ECost}(a,j) - \mathit{ECost}(b,j)]$, $0 < a < b \le j$.

Proof From the definition of $n(i,j)$, it follows that

$$\begin{aligned} \mathit{ECost}(a,j) - \mathit{ECost}(b,j) &= \sum_{l=a}^{j-1} n(a,l) * 2^{j-l} - \sum_{l=b}^{j-1} n(b,l) * 2^{j-l} \\ &= \sum_{l=a}^{b-1} n(a,l) * 2^{j-l} - \sum_{l=b}^{j-1} [n(b,l) - n(a,l)] * 2^{j-l} \end{aligned}$$

Hence,

$$\begin{aligned} \mathit{ECost}(a,j+1) - \mathit{ECost}(b,j+1) &= \sum_{l=a}^{b-1} n(a,l) * 2^{j+1-l} - \sum_{l=b}^{j} [n(b,l) - n(a,l)] * 2^{j+1-l} \\ &= 2[\sum_{l=a}^{b-1} n(a,l) * 2^{j-l} - \sum_{l=b}^{j} [n(b,l) - n(a,l)] * 2^{j-l}] \\ &= 2[\sum_{l=a}^{b-1} n(a,l) * 2^{j-l} - \sum_{l=b}^{j-1} [n(b,l) - n(a,l)] * 2^{j-l}] - 2[n(b,j) - n(a,j)] \\ &= 2[\mathit{ECost}(a,j) - \mathit{ECost}(b,j)] - 2[n(b,j) - n(a,j)] \end{aligned}$$

Hence, when condition Z is true, $\mathit{ECost}(a,j+1) - \mathit{ECost}(b,j+1) \ge \mathit{ECost}(a,j) - \mathit{ECost}(b,j)$. ■

Lemma 5 $\forall (j > 0, r > 1)[M(j+1,r) \ge M(j,r)]$.

Proof Let $M(j,r) = a$ and $M(j+1,r) = b$. Suppose $b < a$. Then,

$$\begin{aligned} C(j,r) &= \mathit{Entries}(j) + C(a,r-1) + \mathit{ECost}(a+1,j) \\ &< \mathit{Entries}(j) + C(b,r-1) + \mathit{ECost}(b+1,j) \end{aligned}$$

since, otherwise, $M(j,r) = b$.

$$\begin{aligned} C(j+1,r) &= \mathit{Entries}(j+1) + C(b,r-1) + \mathit{ECost}[b+1,j+1] \\ &\leq \mathit{Entries}(j+1) + C(a,r-1) + \mathit{ECost}[a+1,j+1] \end{aligned}$$

Therefore,

$$\mathit{ECost}(a+1,j) + \mathit{ECost}(b+1,j+1) < \mathit{ECost}(b+1,j) + \mathit{ECost}(a+1,j+1)$$

Hence,

$$\mathit{ECost}(b+1,j+1) - \mathit{ECost}(a+1,j+1) < \mathit{ECost}(b+1,j) - \mathit{ECost}(a+1,j)$$

From Lemma 4, this is not possible when condition Z is true. So, when condition Z is true, $b \geq a$. ■

The next few lemmas use the function Δ, which is defined as $\Delta(j,r) = C(j,r-1) - C(j,r)$. Since, $C(j,r) \leq C(j,r-1), \Delta(j,r) \geq 0$ for all $j > 0$ and all $r > 1$.

Lemma 6 $\forall(j>0)[\Delta(j,2) \leq \Delta(j+1,2)]$.

Proof If $C(j,2) = C(j,1)$, there is nothing to prove as $\Delta(j+1,2) \geq 0$. The only other possibility is $C(j,2) < C(j,1)$ (i.e., $\Delta(j,2) > 0$). In this case, the solution for $ACHT(j,2)$ uses exactly 2 target lengths. From the recurrence for C (Equations 6 and 7), it follows that $C(j,1) = \mathit{Entries}(j) + \mathit{ECost}(1,j)$, and

$$\begin{aligned} C(j,2) &= \mathit{Entries}(j) + C(a,1) + \mathit{ECost}(a+1,j) \\ &= \mathit{Entries}(j) + \mathit{Entries}(a) + \mathit{ECost}(1,a) + \mathit{ECost}(a+1,j), \end{aligned}$$

for some a, $0 < a < j$. Therefore,

$$\begin{aligned} \Delta(j,2) &= C(j,1) - C(j,2) \\ &= \mathit{ECost}(1,j) - \mathit{Entries}(a) - \mathit{ECost}(1,a) - \mathit{ECost}(a+1,j). \end{aligned}$$

From Equations 6 and 7, it follows that

$$\begin{aligned} C(j+1,2) &\leq \mathit{Entries}(j+1) + C(a,1) + \mathit{ECost}(a+1,j+1) \\ &= \mathit{Entries}(j+1) + \mathit{Entries}(a) + \mathit{ECost}(1,a) + \mathit{ECost}(a+1,j+1) \end{aligned}$$

Hence,

$$\Delta(j+1,2) \geq ECost(1,j+1) - Entries(a) - ECost(1,a) - ECost(a+1,j+1).$$

Therefore,

$$\begin{aligned}\Delta(j+1,2) - \Delta(j,2) &\geq ECost(1,j+1) - Entries(a) - ECost(1,a) - ECost(a+1,j+1) \\ &- ECost(1,j) + Entries(a) + ECost(1,a) + ECost(a+1,j) \\ &= [ECost(1,j+1) - ECost(a+1,j+1)] - [ECost(1,j) - ECost(a+1,j)]\end{aligned}$$

From this and Lemma 4, it follows that when Z is true, $\Delta(j+1,2) - \Delta(j,2) \geq 0$. ■

Lemma 7 $\forall(j>0,k>2)[(\Delta(j,k-1) \leq \Delta(j+1,k-1)] \Longrightarrow \forall(j>0,k>2)[(\Delta(j,k) \leq \Delta(j+1,k)]$.

Proof Assume that $\forall(j>0,k>2)[(\Delta(j,k-1) \leq \Delta(j+1,k-1)]$. We shall show that $\forall(j>0,k>2)[(\Delta(j,k) \leq \Delta(j+1,k)]$. Let $M(j,k)=b$ and $M(j+1,k-1)=c$.

Case 1: $c \geq b$.

$$\begin{aligned}\Delta(j,k) &= C(j,k-1) - C(j,k) \\ &= C(j,k-1) - Entries(j) - C(b,k-1) - ECost(b+1,j) \\ &\leq Entries(j) + C(b,k-2) + ECost(b+1,j) \\ &\quad -Entries(j) - C(b,k-1) - ECost(b+1,j) \\ &= \Delta(b,k-1).\end{aligned}$$

Also,

$$\begin{aligned}\Delta(j+1,k) &= C(j+1,k-1) - C(j+1,k) \\ &\geq Entries(j+1) + C(c,k-2) + ECost(c+1,j+1) \\ &\quad -Entries(j+1) - C(c,k-1) - ECost(c+1,j+1) \\ &= \Delta(c,k-1).\end{aligned}$$

Since $c \geq b$, $\Delta(b,k-1) \leq \Delta(c,k-1)$. Therefore,

$$\Delta(j+1,k) \geq \Delta(c,k-1) \geq \Delta(b,k-1) \geq \Delta(j,k).$$

Case 2: $c < b$.

Let $M[j+1,k]=a$, $M[j,k]=b$, $M[j+1,k-1]=c$, and $M[j,k-1]=d$. From Lemma 5, $a \geq b$ and $c \geq d$. Since $b > c$, $a \geq b > c \geq d$. Also,

$$\begin{aligned} \Delta(j,k) &= C(j,k-1) - C(j,k) \\ &= [Entries(j) + C(d,k-2) + ECost(d+1,j)] \\ &\quad -[Entries(j) + C(b,k-1) + ECost(b+1,j)] \end{aligned}$$

and

$$\begin{aligned} \Delta(j+1,k) &= C(j+1,k-1) - C(j+1,k) \\ &= [Entries(j+1) + C(c,k-2) + ECost(c+1,j+1)] \\ &\quad -[Entries(j+1) + C(a,k-1) + ECost(a+1,j+1)]. \end{aligned}$$

Therefore,

$$\begin{aligned} \Delta\Delta &= \Delta(j+1,k) - \Delta(j,k) \\ &= [C(c,k-2) + ECost(c+1,j+1)] - [C(d,k-2) + ECost(d+1,j)] \\ &\quad +[C(b,k-1) + ECost(b+1,j)] - [C(a,k-1) + ECost(a+1,j+1)]. \end{aligned} \tag{12}$$

Since $j > b > c \geq d = M[j,k-1]$,

$$\begin{aligned} Entries(j) &+ C(c,k-2) + ECost(c+1,j) \\ &\geq Entries(j) + C(d,k-2) + ECost(d+1,j) \end{aligned} \tag{13}$$

Furthermore, since $M(j+1,k) = a \geq b$,

$$\begin{aligned} Entries(j+1) &+ C(b,k-1) + ECost(b+1,j+1) \\ &\geq Entries(j+1) + C(a,k-1) + ECost(a+1,j+1) \end{aligned} \tag{14}$$

Substituting Equations 13 and 14 into Equation 12, we get

$$\begin{aligned} \Delta\Delta &\geq [ECost(c+1,j+1) - ECost(b+1,j+1)] \\ &\quad -[ECost(c+1,j) - ECost(b+1,j)]. \end{aligned}$$

Lemma 4 and $c < b$ imply that when Z is true, $ECost(c+1,j+1) - ECost(b+1,j+1) \geq ECost(c+1,j) - ECost(b+1,j)$. Therefore, $\Delta\Delta \geq 0$. ■

Lemma 8 $\forall(j > 0, k \geq 2)[\Delta(j,k) \leq \Delta(j+1,k)]$.

Proof Follows from Lemmas 6 and 7. ■

Lemma 9 *Let* $k > 2$. $\forall(j > 0)[\Delta(j,k-1) \leq \Delta(j+1,k-1)] \Longrightarrow \forall(j > 0)[M(j,k) \geq M(j,k-1)]$.

Proof Assume that $\forall(j > 0)[\Delta(j,k-1) \leq \Delta(j+1,k-1)]$. Suppose that $M(j,k-1) = a$, $M(j,k) = b$, and $b < a$ for some $j, j > 0$. From Equation 6, we get

$$\begin{aligned} C(j,k) &= Entries(j) + C(b,k-1) + ECost(b+1,j) \\ &\leq Entries(j) + C(a,k-1) + ECost(a+1,j) \end{aligned}$$

and

$$\begin{aligned} C(j,k-1) &= Entries(j) + C(a,k-2) + ECost(a+1,j) \\ &< Entries(j) + C(b,k-2) + ECost(b+1,j). \end{aligned}$$

Hence,

$$C(b,k-1) + C(a,k-2) < C(a,k-1) + C(b,k-2).$$

Therefore,

$$\Delta(a,k-1) < \Delta(b,k-1).$$

However, $b < a$ and $\forall(j > 0)[\Delta(j,k-1) \leq \Delta(j+1,k-1)]$ imply that $\Delta(b,k-1) \leq \Delta(a,k-1)$. Since our assumption that $b < a$ leads to a contradiction, it must be that there is no $j > 0$ for which $M(j,k-1) = a$, $M(j,k) = b$, and $b < a$. ■

Lemma 10 $\forall(j > 0, k > 2)[M(j,k) \geq M(j,k-1)]$.

Proof Follows from Lemmas 8 and 9. ■

Theorem 1 $\forall(j > 0, k > 2)[M(j,k) \geq max\{M(j-1,k), M(j,k-1), k-1\}]$.

Proof Follows from Lemmas 5 and 10 and the fact that $M(j,k)$ is in the range $r-1 \cdots j-1$. ■

Note 1 *From Lemma 8, it follows that whenever* $\Delta(j,k) > 0$, *then* $\Delta(q,k) > 0$, $\forall q > j$.

Since Theorem 1 holds only when condition Z is true, we must be able to check for this condition in any implementation that attempts to use Theorem 1 to reduce the range for m in Equation 6. Unfortunately, the time required to check condition Z exceeds the anticipated gain from using the narrower range. However, we can provide good reason to expect that condition Z will hold on almost all practical data sets (certainly, the condition holds on the practical data sets available to us). Therefore, we propose the use of the narrower range in practice. Even if the condition fails on some data set, the penalty for using the narrower range would be a suboptimal solution. Since C and T are themselves only upper bounds on the cost of optimal solutions, it isn't clear that much is to be lost by solving for T and C inexactly.

The condition Z is

$$ECost(a,j) - ECost(b,j) \geq 2(n(b,j) - n(a,j))$$

where, $0 < a < b \leq j$. We restate this condition in terms of the number of nodes at level $j-1$ of the 1-bit trie for the prefix set. First, put in all missing nodes so that the total number of nodes at level i of the 1-bit trie is 2^i. Call the new nodes *dummy nodes*. Let the number of dummy nodes added to level $j-1$ be $dum(j)$. A node x, dummy or otherwise, at level $j-1$ is covered by a length s, $s < j$, prefix iff x's ancestor at level $s-1$ contains a prefix. Label a node at level $j-1$ iff it is covered by a prefix whose length is between a and $b-1$ and by no prefix whose length is between b and $j-1$ (equivalently, trace a path toward the root from each node x at level j; if the first prefix encountered is at one of the levels $a-1, \cdots, b-2$, label x). Let $N_i(j)$, $0 \leq i \leq 2$ be the number of labeled nodes at level $j-1$ that contain exactly i prefixes (note that a dummy node has no prefix). We see that

$$ECost(a,j) - ECost(b,j) = 2\sum_{i=0}^{2} N_i(j)$$

Further, let $cov(a,j)$ be the number of length j prefixes covered by prefixes whose length is between a and $j-1$. So,

$$\begin{aligned} n(b,j) - n(a,j) &= n(j,j) - cov(b,j) - [n(j,j) - cov(a,j)] \\ &= cov(a,j) - cov(b,j) \\ &= N_1(j) + 2N_2(j) \end{aligned}$$

Therefore,

$$ECost(a,j) - ECost(b,j) - 2(n(b,j) - n(a,j)) = 2(N_0(j) - N_2(j))$$

So, Z is true iff $N_0(j) \geq N_2(j)$. The 1-bit trie for practical data sets has between $2n$ and $3n$ nodes ([4]). So, for large j, $dum(j)$ is fairly close to 2^{j-1} while the number of nodes that have 2 prefixes

```
**Heuristic** OptimalLengths($W$, $k$)
// $W$ is length of longest prefix.
// $k$ is maximum number of target lengths desired.
// Return $C(W, k)$ and compute $M(*, *)$.
{
    **for** $(j = 1; j <= W; j++)$ {
        $C(j, 1) := Entries(j) + ECost(1, j)$;
        $M(j, 1) := -1$; }
    **for** $(r = 1; r <= k; r++)$
        $C(0, r) := 0$;
    **for** $(r = 2; r \leq k; r++)$
        **for** $(j = r; j <= W; j++)$ {
        // Compute $C(j, r)$
            $minJ := max(M(j-1, r), M(j, r-1), r-1)$;
            $minCost := Entries(j) + C(minJ, r-1)$
                    $+ECost(minJ+1, j)$;
            $minCost := C(j, r-1)$;
            $minL := M(j, r-1)$;
            **for** $(m = minJ + 1; m < j; m++)$ {
                $cost := Entries(j) + C(m, r-1) + ECost(m+1, j)$;
                **if** $(cost < minCost)$ **then**
                    $\{minCost := cost;\ minL := m;\}\}$
            $C[j, r] := minCost;\ M[j, r] := minL;\}$
    **return** $C[W, k]$;
}
```

Figure 7: Algorithm for binary-search hash tables

is at most $n/2$. Since the nodes that comprise $N_0(j)$ are drawn from a much larger pool (the pool is $dum(j)$ plus the empty level $j-1$ nodes of the 1-bit trie) while those that comprise $N_2(j)$ are drawn from a much smaller pool (the pool comprises the nodes at level $j-1$ that have 2 prefixes), we expect $N_0(j) >> N_2(j)$. For small j, almost all nodes (dummy or otherwise) at level $j-1$ are empty. Again, we expect $N_0(j) \geq N_2(j)$.

Theorem 1 leads to Heuristic *OptimalLengths* (Figure 7), which computes $C(W, k)$. The complexity of this algorithm is $O(kW^2)$. Using the M values, the at most k storage-optimal target lengths may be determined in an additional $O(k)$ time. When we add in the time needed to compute the $ECost$ and $Entries$ values, the asymptotic complexity becomes $O(nW^2)$. On practical data sets, the complexity is $O(nW + kW^2)$.

Database	Paix	Pb	MaeWest	Aads	MaeEast
Num of prefixes	85987	35302	30718	27068	22712
Num of nodes in trie	173012	91718	81104	74290	65862

Table 1: Prefix databases obtained from IPMA project [2] on Sep 13, 2000.

6 A More Accurate Cost Estimator

In Section 2, we stated three reasons why the actual number of prefixes and markers in H_j may be considerably less than $Entries[j] + ExpansionCost[m+1, j]$. From the stated reasons, it follows that $EC(i,j) + MCost(i,j)$, where $EC(i,j)$ is as defined in Section 3 and $MCost(i,j), 0 < i \leq j \leq W$ is the number of subtrees rooted at level j of the 1-bit trie that contain prefixes that are not covered by a prefix whose length is between i and j.

Using the more accurate estimator for the number of prefixes and markers, the dynamic programming recurrence for C (Equations 6 and 7) becomes

$$C(j,r) = \min_{m \in \{0 \ldots j-1\}} \{C(m, r-1) + EC(m+1, j) + MCost(m+1, j)\}, j > 0, r > 1 \tag{15}$$

$$C(0,r) = 0, C(j,1) = EC(1,j) + MCost(1,j), j > 0 \tag{16}$$

Since $MCost$ may be computed in the same asymptotic time as needed to compute EC and $ECost$, the asymptotic complexity of the heuristic with the more accurate cost estimator is the same as that of the heuristic of [7].

7 Experimental Results

We programmed our space-optimal algorithm, reduced-range heuristic of Section 5, and the more accurate cost estimator heuristic of Section 6 in C and compared their performance against that of the heuristic of Srinivasan [7]. All algorithms and heuristics were adapted for both the $ECHT$ and the $ACHT$ problems. All codes were compiled using the **gcc** compiler and optimization level **O2**. The codes were run on a SUN Ultra Enterprise 4000/5000 computer. For test data, we used the five IPv4 prefix databases of Table 1. These databases correspond to backbone routers. Notice that the number of nodes in the 1-bit trie for each of our databases is between $2n$ and $3n$, where n is the number of prefixes in the database.

Table 2 shows the memory (i.e., sum of number of prefixes and markers to be stored in the hash tables) required by the solution to $ECHT(P,k)$ for each of our five databases. The k values used by us are 3, 7,

Database	k	T		Actual	Optimal
		S	AC		
Paix	3	321189	305147	299884	299884
	7	167162	158542	144617	124516
	15	167048	158428	118687	107195
Pb	3	143648	138880	133105	133105
	7	75036	72313	62598	53318
	15	74921	72198	50389	44996
MaeWest	3	140516	136194	131042	131042
	7	66251	63585	55006	48404
	15	66157	63491	44592	39620
Aads	3	117908	114452	109430	109430
	7	58732	56600	48436	41864
	15	58630	56498	38949	34705
MaeEast	3	103607	100599	95822	95822
	7	51075	49111	41672	36151
	15	50952	48988	33651	29610

Table 2: Number of prefixes and markers in solution to $ECHT(P, k)$

and 15 (corresponding to a lookup performance of 2, 3, and 4 memory accesses per lookup, respectively). For the two heuristics S (heuristic of [7]) and AC (heuristic of Section 6, we provide both the memory requirement as estimated by the T function as well as the actual requirement of the solution generated by these two heuristics (since the two heuristics consistently generated solutions with the same actual memory requirement, the actual requirement data is provided in a single column). This latter quantity is obtained by counting the number of prefixes and markers for the k lengths determined by the heuristic. As expected, the use of the more accurate cost estimator in heuristic AC results in smaller T values. However, these smaller values do not translate into a reduced actual memory cost. In all cases, the use of the more accurate cost estimator did not affect the selection of the k lengths and the resulting actual number of prefixes and markers was the same using heuristics S and AC. The space-optimal algorithm of Section 3 produces solutions whose memory requirement is up to 15% less than that of the two heuristics. Interestingly, for $k = 3$, the two heuristics generate optimal solutions for all 5 of our databases.

Table 3 gives the memory requirements of the solutions obtained by the two heuristics and the optimal algorithm for $ACHT(P, k)$. For the cases $k = 3$ and 7, the memory requirements of the optimal solutions as well as those of the heuristic solutions for the "at most k lengths" version of our problem are the same as those for the "exactly k lengths version". However, for all 5 of our databases the optimal solution for $ACHT(P, 15)$ is superior to that for $ECHT(P, 15)$. The C value of the heuristic solutions

Database	k	C		Actual	Optimal
		S	AC		
Paix	3	321189	305147	299884	299884
	7	167162	158542	144617	124516
	15	167034	158414	124832	105978
Pb	3	143648	138880	133105	133105
	7	75036	72313	62598	53318
	15	74908	72185	54922	44552
MaeWest	3	140516	136194	131042	131042
	7	66251	63585	55006	48404
	15	66143	63477	44449	39209
Aads	3	117908	114452	109430	109430
	7	58732	56600	48436	41864
	15	58617	56485	42735	34299
MaeEast	3	103607	100599	95822	95822
	7	51075	49111	41672	36151
	15	50937	48973	37076	29255

Table 3: Number of prefixes and markers in solution to $ACHT(P, k)$

Database	S	AC	Optimal
Paix	540	820	4230
Pb	280	440	2460
MaeWest	260	390	2210
Aads	230	350	2049
MaeEast	210	320	1889

Table 4: Preprocessing time in milliseconds

for $ACHT(P, 15)$ are smaller than the T values of the heuristic solutions for $ECHT(P, 15)$. With the exception of MaeWest, however, the actual cost of the heuristic solutions for $ACHT(P, 15)$ are larger than the actual costs of the heuristic solutions for $ECHT(P, 15)$. This is due to the fact that T and C are only upper bounds on actual cost.

Table 4 gives the preprocessing time (i.e., the time to compute $ExpansionCost$, EC, $Entries$, MC, and $MCost$ as needed by the heuristic or optimal algorithm) for our heuristics and optimal algorithm. The preprocessing time for the optimal algorithm is 8 to 9 times that for the heuristic of [7]. The preprocessing time for the more accurate cost-estimator heuristic is about 60% more than that for the heuristic of [7].

Tables 5 and 6 give the times needed to solve the dynamic programming recurrences for our heuristics and our optimal-space algorithm. In these tables, RRS and RRAC refer to the reduced-range versions

Database	k	S	RRS	AC	RRAC	Optimal
Paix	3	40	18	46	21	400
	7	168	68	223	76	13660
	15	344	132	442	168	24650
Pb	3	40	19	63	21	400
	7	172	73	311	97	13720
	15	329	138	333	159	24750
MaeWest	3	41	19	39	25	410
	7	168	69	196	84	14620
	15	335	134	416	146	24670
Aads	3	38	19	39	21	420
	7	164	72	263	109	13650
	15	343	139	421	154	24550
MaeEast	3	39	19	38	22	400
	7	168	74	163	82	13690
	15	346	141	332	158	25060

Table 5: Execution time, in μsec, for $ECHT(P,k)$

of S and AC, respectively. As expected from the analyses of these methods, the preprocessing time is significantly larger than the time needed to solve the dynamic programming recurrences (note that the times in Table 4 are in milliseconds while those in Tables 5 and 6 are in microseconds). Note also that the time for the reduced-range version of each heuristic is less than half that of the original heuristic.

8 Conclusion

We have developed optimal algorithms for the $ECHT(P,k)$ and $ACHT(P,k)$ problems; shown how the dynamic programming recurrence for the heuristic of [7] may be solved in $O(nW + kW^2)$ time on practical data sets (in contrast, the analysis of [7] suggests an $O(nW^2)$ complexity); proposed a reduced-range heuristic as well as a more accurate cost-estimator heuristic. Experimental results show that the reduced-range heuristic reduces the time to solve the dynamic programming recurrences by more than a factor of 2 while yielding the same result as the original full-range heuristic. We are unable to compare the reduction in run time that results from our methods to do the preprocessing versus that proposed in [7], because the code of [7] is unavailable. Although the more accurate cost-estimator heuristic results in solutions with a better cost estimate than those produced by the heuristic of [7], the actual costs of the solutions produced by the two heuristics are the same for our test sets. The optimal-space algorithm produces solutions with a memory requirement up to 15% less than that of the solutions produced by the heuristics. However, the optimal-space algorithm takes between 8 and 9 times as much time

Database	k	S	RRS	AC	RRAC	Optimal
Paix	3	40	18	46	21	440
	7	170	66	165	113	14420
	15	344	127	420	162	26490
Pb	3	39	19	86	27	440
	7	165	69	210	82	14500
	15	342	136	401	146	25800
MaeWest	3	39	19	39	21	430
	7	173	67	231	95	14570
	15	338	129	462	170	26060
Aads	3	40	19	39	28	420
	7	168	69	264	81	14550
	15	343	131	485	149	26240
MaeEast	3	40	19	39	22	430
	7	167	70	162	109	14470
	15	346	134	365	183	26530

Table 6: Execution time, in μsec, for $ACHT(P, k)$

(preprocessing by recurrence solution time) as does the heuristic of [7] and between 5 and 6 times the time taken by the more accurate cost-estimator heuristic.

References

[1] E. Horowitz, S. Sahni, and D. Mehta, Fundamentals of data structures in C++, W.H. Freeman, NY, 1995, 653 pages.

[2] Merit, Ipma statistics, http://nic.merit.edu/ipma, (snapshot on Sep. 13, 2000), 2000.

[3] M. Ruiz-Sanchez, E. Biersack, and W. Dabbous, Survey and taxonomy of IP address lookup algorithms, *IEEE Network*, 2001, 8-23.

[4] S. Sahni and K. Kim, Efficient construction of fixed-stride multibit tries for IP lookup, *Proceedings 8th IEEE Workshop on Future Trends of Distributed Computing Systems*, 2001, 178-184.

[5] S. Sahni and K. Kim, Efficient construction of variable-stride multibit tries for IP lookup, *Proceedings IEEE Symposium on Applications and the Internet (SAINT)*, 2002, 220-227.

[6] S. Sahni, K. Kim, and H. Lu, Data structures for one-dimensional packet classification using most-specific-rule matching, *International Symposium on Parallel Architectures, Algorithms, and Networks (ISPAN)*, 2002, 3-14.

[7] V. Srinivasan, Fast and efficient Internet lookups, *CS Ph.D Dissertation*, Washington University, Aug., 1999.

[8] V. Srinivasan and G. Varghese, Faster IP lookups using controlled prefix expansion, *ACM Transactions on Computer Systems*, Feb:1-40, 1999.

[9] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, Scalable high speed IP routing lookups, *ACM SIGCOMM*, 1997, 25-36.