# Speeding Up Multi-level Route Analysis Through Improved Multi-LCS Algorithm

Pei Tu[1], Xiapu Luo[2,3], Weigang Wu[1](✉), and Yajuan Tang[4]

[1] Department of Computer Science, Sun Yat-Sen University, Guangzhou, China
tuwantpkyj@hotmail.com, wuweig@mail.sysu.edu.cn

[2] Department of Computing, The Hong Kong Polytechnic University, Kowloon, Hong Kong
csxluo@comp.polyu.edu.hk

[3] The Hong Kong Polytechnic University Shenzhen Research Institute, Shenzhen, China

[4] Department of Electronic and Information Engineering, Shantou University, Shantou, China
yjtang@stu.edu.cn

**Abstract.** Although the multi-level route analysis (e.g., AS, subnet, IP levels) is very useful to many applications (e.g. profiling route changes, designing efficient route-tracing algorithms, etc.), few research investigates how to conduct such analysis efficiently. Regarding routes as sequences, current approaches only handle two routes at a time and they just apply algorithms designed for general sequence comparison. In this paper, we propose and implement a new approach named `Fast-rtd` that contrasts multiple routes simultaneously and exploits the unique features of Internet routes to decrease the computational complexity in terms of time and memory. Our extensive evaluations on real traceroute data demonstrate the efficiency of `Fast-rtd`, such as more than 45% memory reduction, 3% to 15% pruning rate increase, and up to 25% speed improvement.

**Keywords:** Multi-level route analysis · Multiple LCS · BGP

## 1 Introduction

Identifying the common and/or the different portions among a set of routes in multiple levels (e.g., AS, subnet, IP levels) is a primitive of route analysis [1]. Such a primitive is very useful to many applications, such as profiling route changes and their impacts [2–6], measuring route asymmetry and diversity [7–10], and designing efficient route-tracing solutions [11], to name a few. Although the primitive's basic idea is straightforward, it is non-trivial to efficiently realize this primitive because of the tremendous volume of data. For example, the Ark project collects 500 million traceroutes in each probing unit and more than 10 billion traceroutes have been recorded. [12]. Moreover, the majority of existing systems process each level independently, thus resulting in redundant processing and high demand of

© Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2014
V.C.M. Leung et al. (Eds.): TridentCom 2014, LNICST 137, pp. 327–337, 2014.
DOI: 10.1007/978-3-319-13326-3_31

resources [1]. Our previous system, `rtd`, improves the analysis efficiency by integrating all levels recursively [1].

However, we identify another two deficiencies in existing approaches including `rtd`. First, existing methods only handle two routes at a time. Applications may need to process multiple routes at the same time, such as locating the invariant portions of the routes collected during a period of time, determining the common IP/Subnet/ASes among a set of routes, etc. Although it is possible to first analyze *each* pair of routes and then synthesize the result, such approach is inefficient because the number of comparisons may increase exponentially. Second, existing approaches usually regard routes as sequences and then apply algorithms designed for general sequences to process routes. In other words, they do not exploit the unique features of Internet routes to optimize the processing. Note that routes are special sequences. For example, each IP address will appear in a correct route once to avoid loop. In this paper, we propose a novel approach named `Fast-rtd` for multi-level route analysis, which can overcome the above two limitations and achieve higher efficiency. We make three contributions:

1. We identify the limitations of existing multi-level route analysis approaches, including the inefficiency of processing multiple routes and the lack of optimization by exploiting the unique features of Internet routes.
2. We propose a new approach named `Fast-rtd` that supports contrasting multiple routes at the same time and exploits the unique features of Internet routes to further decrease the computational complexity in terms of time and memory.
3. We implement the new approach in around 800 lines of C++ codes and conduct extensive evaluations on its performance using real traceroute data. The results show that `Fast-rtd` can achieve more than 45% memory reduction, 3% to 15% pruning rate increase, and up to 25% speed improvement.

The remainder of this paper is organized as follows. Section 2 introduces the multi-level route analysis. We detail the algorithm in Section 3 and evaluate it in Section 4. After introducing related work in Section 5, we conclude the paper in Section 6.

## 2 Multi-level Route Analysis

Following [1], we define a legitimate route $R$ as an ordered sequence of nodes $r_1 r_2 \ldots r_{|R|}$, where $r_i \neq r_j$ $(i, j \in \{1, 2, \ldots, |R|\})$ to avoid routing loops. Each node $r_i$, $i \in \{1, 2, \ldots, |R|\}$, is an IP address having $n$ levels of labels, and its $t$-th level of label is denoted as $L_t(r_i)$. The levels construct an ordered set $\mathcal{L} = \{L_1, ..L_n\}$ with a transitive relation $\succ$ that have the following properties:

1. $L_t \succ L_{t+1} \Rightarrow$ if $L_t(r_i) \neq L_t(r_j)$ then $L_{t+1}(r_i) \neq L_{t+1}(r_j)$.
2. $L_1 \succ L_2 \cdots \succ L_n$.

Note that $L_{t+1}(r_i) \neq L_{t+1}(r_j)$ does not imply $L_t(r_i) \neq L_t(r_j)$, $\forall i, j \in \{1, 2, \ldots, |R|\}$. Following [1–4,7–10], we use $AS(r_i)/SN(r_i)/IP(r_i)$ to denote the AS/subnet/IP-level label of $r_i$ and define $\mathcal{L} = \{$AS-level, subnet-level, IP-level$\}$ with

$$AS - level \succ subnet - level \succ IP - level \tag{1}$$

The goal of a multi-level route analysis is two-fold. First, it identifies the common portions among a set of routes $R_k$ ($k = 1, \ldots, M$, $M \geq 2$) on different levels. Then, based on the common portions, it outputs the difference among these routes on different levels. Instead of conducting the comparison on each level independently, our previous work (i.e., `rtd`) integrates the analysis of all levels recursively [1]. Note that `rtd` compares two routes at a time using LCS algorithms for general sequence. Although we can use it to analyze each pair of routes and then synthesize the result, it has much larger computation complexity than the algorithms designed for locating LCS of multiple sequences. For example, for a set of $M$ routes, `rtd` will conduct $\frac{M(M-1)}{2}$ comparisons. In this paper, we propose `Fast-rtd` to extend `rtd`'s functionality from comparing two routes to multiple routes by using an advanced multi-string LCS algorithm and further improve the performance in terms of time and memory by exploiting Internet routes' features. We will use an example shown in Fig. 1 to introduce the basic idea of multi-level route analysis and then detail `Fast-rtd` in Section 3.

As shown in Fig. 1, we compare two routes $R_1$ ={IP1, IP2, IP3, IP4, IP5, IP6, IP7, IP8, IP9, IP10, IP11, IP12, IP13, IP14, IP15} and $R_2$ ={IP1, IP2, IP3, IP6, IPa, IPb, IPc, IPd, IPe, IP12, IPf, IP14, IP15}. `rtd` starts the comparison from the AS level. Since $R_1$ and $R_2$ differs in the second AS (i.e., $AS2$ and $AS5$), we know that subnets/IPs belonging to $AS2$ in $R_1$ and those belonging to $AS5$ in $R_2$ are different according to Eqn.1. Therefore, `rtd` will compare subnets in the same ASes (e.g., $AS1$, $AS3$, and $AS4$). Taking $AS1$ as an example, `rtd` will contrast $AS1$'s subnets in $R_1$ (i.e., {SN1, SN2, SN3}) and that in $R_2$ (i.e., {SN1, SN2, SN3}). Since they are the same, `rtd` will compare the IPs in $R_1$ (i.e.,{IP1, IP2, IP3, IP4, IP5, IP6}),and those in $R_2$ (i.e., {IP1, IP2, IP3, IP6}) and identify the common IPs (i.e., {IP1, IP2, IP3, IP6} ) and the difference (i.e., {IP4, IP5}). After that, `rtd` will conduct the same analysis to $AS3$ and $AS4$. As shown in Fig. 1, elements in red box are the differences between $R_1$ and $R_2$.

## 3 Fast-rtd

To extend the comparison from two routes to multiple routes, `Fast-rtd` adopts the `Fast-LCS` algorithm [13], which provides a near-linear solution to the problem of finding the longest common subsequence (LCS) among a set of sequences, which is a NP-hard problem [13,14]. `Fast-rtd` further improves the performance of `Fast-LCS` in terms of speed and memory usage by exploiting routes' features. `Fast-rtd` consists of three steps to be elaborated in the following Sections 3.1 - 3.3. The first two steps are the same as those in the `Fast-LCS` algorithm and our improvements are introduced in the third step.

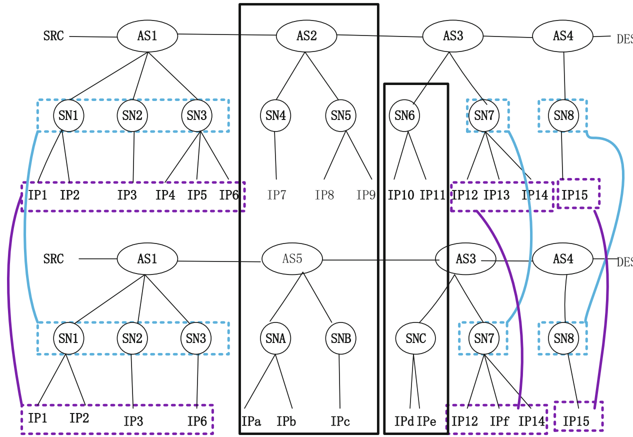



**Fig. 1.** Example of multi-level route analysis

### 3.1 Building Successor Tables

Given a set of routes $R_k$ ($k = 1, \ldots, M$, $M \geq 2$), `Fast-rtd` first constructs a set $R_U$ for containing all unique $r_{k,i}$ ($k = 1, \ldots, M$, $i = 1, \ldots, |R_k|$) and then builds a successor table (denoted as $T_k$) for each route $R_k$ following [13]. Each element in $T_k$ is defined as follows:

$$T_k(i,j) = \begin{cases} \min\{a|a \in S_k(i,j)\}, & S_k(i,j) \neq \phi \\ -, & otherwise \end{cases} \tag{2}$$

Here, $S_k(i,j) = \{a|R_k(a) = R_U(i), a > j)\}$, it stores the positions of $R_U(i)$ in $R_k$, where $i = 1, \ldots, |R_U|$ and $j = 0, \ldots, |R_k|$. Since the same $r$ will not appear twice in a route, each row of $T_k$ has only one integer.

| | $r_1$ | $r_2$ | $r_4$ | $r_3$ | $r_5$ |
|---|---|---|---|---|---|
| $r_1$ | 1 | - | - | - | - |
| $r_2$ | 2 | 2 | - | - | - |
| $r_3$ | 4 | 4 | 4 | 4 | - |
| $r_4$ | 3 | 3 | 3 | - | - |
| $r_5$ | 5 | 5 | 5 | 5 | 5 |

(a) $R_1$'s successor table ($T_1$).

| | $r_2$ | $r_1$ | $r_4$ | $r_3$ | $r_5$ |
|---|---|---|---|---|---|
| $r_1$ | 2 | 2 | - | - | - |
| $r_2$ | 1 | - | - | - | - |
| $r_3$ | 4 | 4 | 4 | 4 | - |
| $r_4$ | 3 | 3 | 3 | - | - |
| $r_5$ | 5 | 5 | 5 | 5 | 5 |

(b) $R_2$'s successor table ($T_2$).

| | $r_2$ | $r_1$ | $r_3$ | $r_5$ |
|---|---|---|---|---|
| $r_1$ | 2 | 2 | - | - |
| $r_2$ | 1 | - | - | - |
| $r_3$ | 3 | 3 | 3 | - |
| $r_4$ | - | - | - | - |
| $r_5$ | 5 | 5 | 5 | 5 |

(c) $R_3$'s successor table ($T_3$).

**Fig. 2.** The successor tables of $R_1$, $R_2$ and $R_3$

We use an example to illustrate how to construct successor tables. Given $R_1 = \{r_1, r_2, r_4, r_3, r_5\}$, $R_2 = \{r_2, r_1, r_4, r_3, r_5\}$, and $R_3 = \{r_2, r_1, r_3, r_5\}$, we build $R_U = \{r_1, r_2, r_3, r_4, r_5\}$ and construct $T_1$, $T_2$, and $T_3$ as shown in Fig. 2.

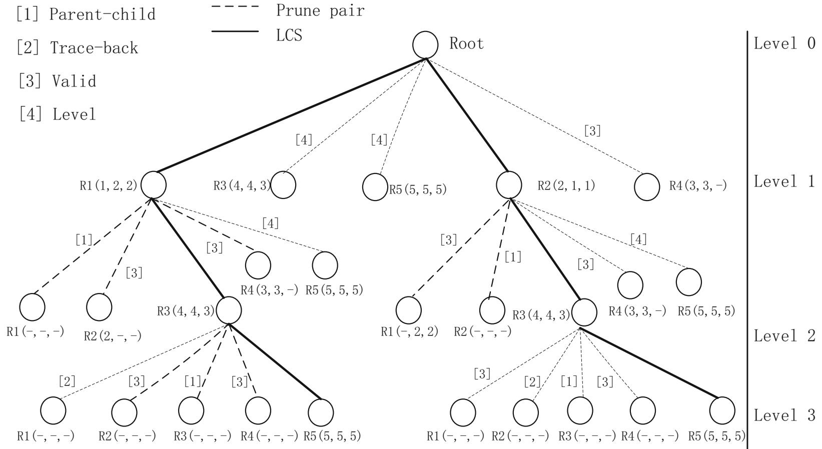



**Fig. 3.** Example of `Fast-rtd`'s pruning operations

### 3.2 Constructing LCS Tree

The LCS tree is constructed from the successor tables. We define an identical tuple as $(i_1, \ldots, i_m)$ if $r_{1,i_1} = r_{2,i_2} = \ldots = r_{m,i_m}$. Let $(i_1, \ldots, i_m)$ and $(j_1, \ldots, j_m)$ be two identical tuples. If $i_k < j_k$ for $k = 1, \ldots, m$, $(i_1, \ldots, i_m)$ is a predecessor of $(j_1, \ldots, j_m)$, or $(j_1, \ldots, j_m)$ is a successor of $(i_1, \ldots, i_m)$. For an identical tuple $(i_1, \ldots, i_m)$, its direct successors can be identified through Eqn.3. Starting from the identical tuples on the first level, we can emulate all direct successors and construct a tree.

$$(i_1, \ldots, i_m) \rightarrow (T_1(k, i_1), \ldots, T_m(k, i_m)) \tag{3}$$

From (3) we can see that the operation of producing successor tuples is to couple the elements of the $(i_m)$th column of $T_m$.

Following the example in Section 3.1, we enumerate all the first identical tuples: $r_1(1, 2, 2)$, $r_2(2, 1, 1)$, $r_3(4, 4, 3)$, $r_4(3, 3, -)$, $r_5(5, 5, 5)$. The first identical tuples are those whether $r_1$, $r_2$, $r_3$, $r_4$, $r_5$ appear firstly in route sequences $R_1$, $R_2$ and $R_3$, individually. Take $r_1(1, 2, 2)$ as an example, we can see $R_1[1] = R_2[2] = R_3[2]$. To generate its direct successors, $r_1(1, 2, 2)$ couples the 1st column of $T_1$, the 2nd column of $T_2$, and the 2nd column of $T_3$ to produce new tuples:$r_1(-, 2, 2)$, $r_2(-, -, -)$, $r_3(4, 4, 3)$, $r_4(3, 3, -)$, and $r_5(5, 5, 5)$. Note that although all tuples can produce its successors, not all its successors are valid. Four pruning operations, to be introduced in the next section, are used to remove the invalid successors.

### 3.3 Pruning the Tree and Outputting LCSes

Since not all paths in the tree lead to LCSes, we employ four pruning operations to remove tuples that do not belong to LCSes. Two operations (i.e., valid pruning operation and level pruning operation) are from `Fast-LCS` and the other two

operations (i.e., parent-child pruning operation and track-back pruning operation) are proposed by us exploiting the features of route sequences. We detail the four pruning operations belows.

*Valid Pruning Operation.* The valid pruning operation removes identical tuples with $'-'$. Tuples like $(k, -)$ or $(-, k)$ are invalid and can be pruned directly. Given $N$ route sequences, to determine whether a tuple $(i_1, i_2, ..., i_n)$ contains $'-'$ or not, `Fast-rtd` will do a linear search, and hence the time complexity is $O(N/2)$.

*Level Pruning Operation.* The level pruning operation removes redundant identical tuples on the same level. More precisely, given two identical tuples $(i_1, \ldots, i_m)$ and $(j_1, \ldots, j_m)$, if $i_1 < j_1$ and $i_k \leq j_k$ (k=2,...,m), then $(j_1, \ldots, j_m)$ will be removed. Given $N$ route sequences, for two tuples $(i_1, i_2, \ldots, i_n)$ and $(j_1, j_2, \ldots, j_n)$, `Fast-rtd` will conduct $N$ times comparison to make sure whether the one should be pruned or not, thus the time complexity will be $O(N)$.

*Parent-Child Pruning Operation.* The parent-child pruning operation deletes a child identical tuple if it has the same upper level label as its parent. Given a subsequence $\{r_1, r_2, \ldots\}$ where $AS(r_1) = AS(r_2) = AS0$ and $AS(r_3) = AS(r_4) = AS1$, in the $AS - level$, the subsequence can be represented as $\{AS0, AS0, AS1, AS1, \ldots\}$ with redundant $AS0$ and $AS1$. Parent-child pruning operation can help to avoid the redundance during the construction of the LCS tree, because it makes sure that all the tuples have different level label with their parent. As it just processes the level label to decide whether a tuple should be pruned or not,the time complexity of it will be $O(1)$.

*Trace-Back Pruning Operation.* The trace-back pruning operation deletes tuples whose labels have occurred on the path from itself to the root of the LCS tree. This is motivated by the observation that legitimate route sequences do not contain loops. Given a subsequence $\{r_1, r_2, \ldots\}$, if $L(r_j)! = L(r_i)$,$(j > i)$, then $L(r_j)! = L(r_k)$, $(0 < k < i)$. Trace-back pruning operation removes all the tuples that may form a loop by tracing back to the root of the LCS tree. The tracing time is $O(D)$, where $D$ is the depth of the tuple in the tree. Since the length of a route is usually short (i.e., less than 30), $O(D)$ can be approximated as $O(1)$.

*Example.* By applying these four pruning operations during the construction of the LCS tree, `Fast-rtd` can prune a large amount of tuples during the construction of the LCS tree to improve the efficiency. Fig. 3 demonstrates the LCS tree and how pruning operations remove tuples. The LCS tree begins with the initial tuples $r_1(1, 2, 2)$, $r_2(2, 1, 1)$, $r_3(4, 4, 3)$, $r_4(3, 3, -)$, and $r_5(5, 5, 5)$ on the first level. The valid pruning operation will prune the tuple $r_4(3, 3, -1)$, and then using level pruning operation, we can prune $r_3(4, 4, 3)$ and $r_5(5, 5, 5)$ with $r_1(1, 1, 2)$, $r_2(2, 1, 1)$ left on the first level. Then $r_1(1, 1, 2)$ produces its child tuples $r_1(-, -, -)$, $r_2(2, -, -)$, $r_3(4, 4, 3)$, $r_4(3, 3, -)$, $r_5(5, 5, 5)$ on level 2 and

$r_2(2,1,1)$ generates its child pairs $r_1(-,2,2)$, $r_2(-,-,-)$, $r_3(4,4,3)$, $r_4(3,3,-)$, $r_5(5,5,5)$ on level 2.

On level 2, using the parent-child pruning operation, we prune $r_1(-,-,-)$, $r_2(-,-,-)$. The valid pruning operation removes $r_2(2,-,-)$, $r_1(-,2,2)$, $r_4(3,3,-)$, $r_4(3,3,-)$ and the level pruning operation removes $r_5(5,5,5)$, $r_5(5,5,5)$ with the only $r_3(4,4,3)$, $r_3(4,4,3)$ left on level 2. Then we produce the child tuples of $r_3(4,4,3)$, $r_3(4,4,3)$, including $r_1(-,-,-)$, $r_2(-,-,-)$, $r_3(-,-,-)$, $r_4(-,-,-)$, $r_5(5,5,5)$, $r_1(-,-,-)$, $r_2(-,-,-)$, $r_3(-,-,-)$, $r_4(-,-,-)$, $r_5(5,5,5)$ on level 3.

By adopting the parent-child pruning operation, we prune the tuples $r_3(-,-,-)$ and $r_3(-,-,-)$ on level 3. The trace-back pruning operation removes $r_1(-,-,-)$ and $r_2(-,-,-)$, because they have appeared on LCS path to the root. The valid pruning operation eliminates $r_1(-,-,-)$, $r_2(-,-,-)$, $r_4(-,-,-)$, and $r_4(-,-,-)$. Continuing the pruning process, we find that $r_5(5,5,5)$ and $r_5(5,5,5)$ are the leaf tuples left on the level 3, meaning that the construction of the LCS tree is finished. By tracing back from the two $r_5(5,5,5)$ on the level 3 to the root, we will obtain two LCS: $r_1r_3r_5$ and $r_2r_3r_5$.

### 3.4 Order of Using the Four Pruning Operations

While there are four pruning operations, we find through extensive experiments against various data sets that they had better be used in the order of parent-child, trace back, valid and finally level pruning. By first using the parent-child pruning operation and the trace-back pruning operation, a large number of tuples on the same level will be pruned. Then the number of tuples to be pruned by the valid pruning operation and/or the level pruning operation will be significantly decreased, thus reducing the computation time.

*Analysis.* Let $N$ be the number of routes and $K$ denote the number of tuples to be pruned by the valid pruning operation or the level pruning operation in the `Fast-LCS` algorithm. Since the time complexity of the valid pruning operation and the level pruning operation are $O(N/2)$ and $O(N)$, the time complexity of pruning tuples in the `Fast-LCS` will be $T_1 = K * O(N)$ if we regard both as $O(N)$. Note that the number of tuples to be pruned by the valid or the level pruning operations is reduced in the `Fast-rtd` algorithm, because the parent-child and the trace-back pruning operations have pruned a portion of these tuples. Let $p$ be the pruning rate of the parent-child and the trace-back pruning operations. Then the time complexity of using the parent-child or the trace-back pruning operations will be $T_{2_1} = K * p * O(1)$ and that of using the valid or the level pruning operations will be $T_{2_2} = K * (1-p) * O(N)$. Therefore, the total time complexity in the `Fast-rtd` algorithm will be $T_2 = T_{2_1} + T_{2_2}$. Compared to the `Fast-LCS`, we can see that the saved time $T' = T_2 - T_1 = K * p * (O(N) - O(1))$. When $N$ and $p$ increase, `Fast-rtd` will be more efficient.

*Example.* Consider the example in Fig. 3 where there are three routes. Tuples $r_3(4,4,3)$ on level 2 produces its child tuples $r_3(4,4,3)$ and $r_3(4,4,3)$, including $r_1(-,-,-)$, $r_2(-,-,-)$, $r_3(-,-,-)$, $r_4(-,-,-)$, $r_5(5,5,5)$, $r_1(-,-,-)$,

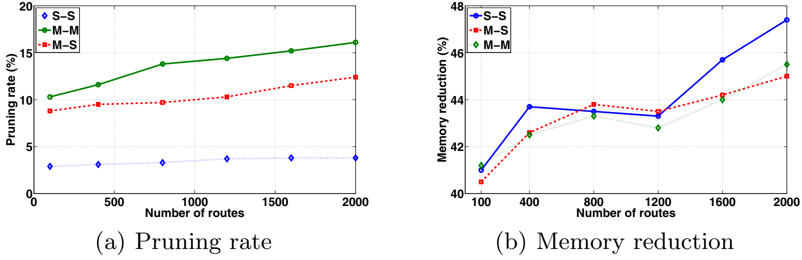
$r_2(-,-,-)$, $r_3(-,-,-)$, $r_4(-,-,-)$, and $r_5(5,5,5)$ on level 3. In the Fast-LCS algorithm, using only the valid pruning operation and the level pruning operation, it will prune the child tuples $r_1(-,-,-)$, $r_2(2,-,-)$, $r_3(-,-,-)$, $r_4(-,-,-)$, $r_1(-,-,-)$, $r_2(2,-,-)$, $r_3(-,-,-)$, and $r_4(-,-,-)$. The time complexity will be $8 * O(N/2)$. However, in the Fast-rtd, by using the parent-child operation first, we can prune $r_3(-,-,-)$ and $r_3(-,-,-)$ in 2*O(1), and then prune $r_1(-,-,-)$ and $r_2(-,-,-)$ by using the trace-back pruning operation in 2*O(1). After that, we will prune $r_4(-,-,-)$ and $r_4(-,-,-)$ through the valid pruning operation in time 2*O(N/2). Therefore, the saved time $t = 8*O(N/2) - 2*O(N/2) - 4*O(1) = 6*O(N/2) - 4*O(1)$. When $N$ increases, $t$ will increase significantly. Therefore, by first using the parent-child pruning operation and the trace-back pruning operation, the number of tuples on the same level will be largely reduced. Hence, the number of tuples to be pruned by the valid pruning operation or the level pruning operation will decrease, thus reducing computation time.

## 4 Evaluation

We implement both Fast-rtd and Fast-LCS for comparisons. They are tested against two sets of real traceroute data. The first one is the iPlane data set [15] from April to July in 2012. The other one contains traceroute data from Planetlab nodes to two subnets in Taiwan, which were collected by ourselves through paris-traceroute [16]. The two data sets have around 100K routes. For each unique IP address in these routes, we get its subnet and AS information through WHOIS database and team cymru's IP to ASN mapping service [17].

Since Fast-rtd is designed to handle multiple routes, we evaluate it using three types of routes, which represent different use cases. The first type of data (denoted as S-S) include routes from one IP address to another collected during a period of time. Such kind of data will be examined when a user wants to know the evolving of the routes between two hosts. The second type of data (denoted as M-S) comprises of routes from a set of IPs to one IP, for example, from an AS to one IP. Such type of data is useful for inspecting the multiple routes to a destination. For example, a multi-homing user may have server upstream providers, which provides different paths and even performance for the user to communicate with another IP. The third type of data (denoted as M-M) consists of routes from a set of IPs to another set of IPs, for example, from an AS to another AS. Such type of data may be used by network administrator for investigating the routes between ASes, which are useful to traffic engineering.

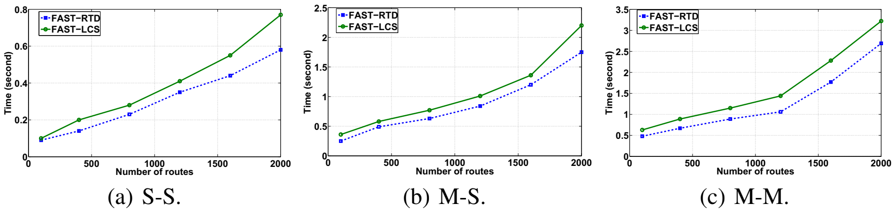
Fast-rtd improves Fast-LCS by taking into account the features of Internet routes. Fig. 4 illustrates the increased pruning rate and decreased memory usages. The X-axis is the number of routes and each point represents an average value from 20 times experiments, where a certain number of routes were randomly selected from our data set. Fig. 4(a) shows that the increased pruning rate is within the range of [3%,15%]. The value increases with the number of routes. The M-M types of routes have much larger pruning rate than the M-S and S-S types of routes. Moreover, the increment of pruning rate is fast for the M-M type




(a) Pruning rate (b) Memory reduction

**Fig. 4.** The performance improvement introduced by `Fast-rtd` in different scenarios

of routes when the number of routes increases. The reason may be that the M-M type of routes have much larger number of unique IPs. Fig. 4(a) demonstrates that `Fast-rtd` can lead to more than 40% memory reduction. Moreover, the reduction rate increases along with the number of roues.




(a) S-S. (b) M-S. (c) M-M.

**Fig. 5.** The speed comparison between `Fast-rtd` and `Fast-LCS` in different scenarios

Fig. 5 compares the time required by `Fast-rtd` and `Fast-LCS` to process different scale of routes in different scenarios. The X-axis is the number of routes and the Y-axis is the computation time. Each point is also an average value from 20 times experiments with randomly selected routes. We can see that `Fast-rtd` uses much less time than `Fast-LCS` and the improvement increases along with the number of routes. For example, when conducting experiments on 2000 routes, we can observe up to 25% speed improvement. When the number of routes is small, the difference is small. The type of data also affects the improvement. For example, as the improvement of pruning rate due to `Fast-rtd` is small for the S-S type of data compared to other type of data as shown in Fig. 4, the time reduction in Fig. 5(a) is less obvious than that in Fig. 5(b) and Fig. 5(c).

## 5 Related Work

Contrasting routes on different levels is very useful to many applications, such as characterizing route changes [2–6], measuring route asymmetry and diversity

[7–10], and designing efficient route-tracing solutions[11]. Some research uses Jaccard Distance to quantify the changes in routes [4,7,8]. Since this metric does not contain order information, people propose using Edit distance and its variants to profile route changes [2,9,10,18]. However, all these approaches may result in computational redundancy because they process the information on different levels independently [1]. We propose `rtd` to eliminate redundancy by integrating all levels, thus achieving much better efficiency. However, all these approaches including `rtd` have two deficiencies. First, they only compare two routes at a time and cannot be easily extended to handling multiple routes. Second, they just apply algorithms designed for processing general sequences to routes without exploiting the unique features in Internet routes. Inheriting the basic idea of integrating all levels, `Fast-rtd` extends `rtd` by identifying LCS on multiple routes and improving the performance in terms of time and memory.

## 6 Conclusion

In this paper, we propose and implement a new approach named `Fast-rtd` for multi-level route analysis. Different from existing approaches that can only deal with two routes at a time, `Fast-rtd` can contrasts multiple routes simultaneously. Moreover, instead of directly applying algorithms for processing sequences, `Fast-rtd` adopts new pruning operation and storage techniques, which are motivated by Internet routes' features, to decrease the computational complexity in terms of time and memory. Our extensive evaluations on real traceroute data demonstrate the efficiency of `Fast-rtd`, such as more than 45% memory reduction, 3% to 15% pruning rate increase, and up to 25% speed improvement, compared with `Fast-LCS`, the approach for analysis of general sequences.

**Acknowledgments.** We thank Ang Chen for his discussion and suggestions. This work is supported in part by the CCF-Tencent Open Research Fund, the Pearl River Nova Program of Guangzhou (No. 2011J2200088), Guangdong Natural Science Foundation (No. S2012010010670), the National Natural Science Foundation of China (No. 60903185), and the Academic Innovation Team Construction Project of Shantou University (No. ITC12001).

## References

1. Chen, A., Chan, E., Luo, X., Fok, W., Chang, R.: An efficient approach to multi-level route analytics. In: Proc. IFIP/IEEE IM (2013)
2. Schwartz, Y., Shavitt, Y., Weinsberg, U.: On the diversity, stability and symmetry of end-to-end Internet routes. In: Proc. IEEE GI Symposium (2010)
3. Logg, C., Cottrell, L., Navratil, J.: Experiences in traceroute and available bandwidth change analysis. In: Proc. ACM SIGCOMM Workshop on Network Troubleshooting (2004)
4. Chan, E.W.W., Luo, X., Fok, W.W.T., Li, W., Chang, R.K.C.: Non-cooperative diagnosis of submarine cable faults. In: Spring, N., Riley, G.F. (eds.) PAM 2011. LNCS, vol. 6579, pp. 224–234. Springer, Heidelberg (2011)

5. Fok, W., Luo, X., Mok, R., Li, W., Liu, Y., Chan, E., Chang, R.: Monoscope: Automating network faults diagnosis based on active measurements. In: Proc, IFIP/IEEE IM (2013)
6. Liu, Y., Luo, X., Chang, R., Su, J.: Characterizing inter-domain rerouting by betweenness centrality after disruptive events. IEEE JSAC 31(6) (2013)
7. Pucha, H., Zhang, Y., Mao, Z., Hu, Y.: Understanding network delay changes caused by routing events. In: Proc. ACM SIGMETRICS (2007)
8. Pathak, A., Pucha, H., Zhang, Y., Hu, Y.C., Mao, Z.M.: A measurement study of Internet delay asymmetry. In: Claypool, M., Uhlig, S. (eds.) PAM 2008. LNCS, vol. 4979, pp. 182–191. Springer, Heidelberg (2008)
9. He, Y., Faloutsos, M., Krishnamurthy, S.: Quantifying routing asymmetry in the Internet at the AS level. In: Proc. IEEE GLOBECOM (2004)
10. Han, J., Watson, D., Jahanian, F.: An experimental study of Internet path diversity. IEEE Trans. Dependable and Secure Computing (2006)
11. Beverly, R., Berger, A., Xie, G.: Primitives for active Internet topology mapping: Toward high-frequency characterization. In: Proc, ACM/USENIX IMC (2010)
12. Hyun, Y.: Archipelago measurement infrastructure. http://www.caida.org/projects/ark/
13. Chen, Y., Wan, A., Liu, W.: A fast parallel algorithm for finding the longest common sequence of multiple biosequences. BMC Bioinformatics 7(S4) (2006)
14. Wang, Q., Korkin, D., Shang. Y.: A fast multiple longest common subsequence (MLCS) algorithm. IEEE TKDE 23(3) (2011)
15. Madhyastha, H., Isdal, T., Piatek, M., Dixon, C., Anderson, T.: iPlane: An information plane for distributed services. In: Proc, USENIX OSDI (2006)
16. Augustin, B., Cuvellier, X., Orgogozo, B., Viger, F., Friedman, T., Latapy, M., Magnien, C., Teixeira, R.: Avoiding traceroute anomalies with Paris traceroute. In: Proc. ACM/USENIX IMC (2006)
17. Team Cymru. IP to ASN service. http://www.team-cymru.org/Services/ip-to-asn.html
18. Schwartz, Y., Shavitt, Y., Weinsberg, U.: A measurement study of the origins of end-to-end delay variations. In: Krishnamurthy, A., Plattner, B. (eds.) PAM 2010. LNCS, vol. 6032, pp. 21–30. Springer, Heidelberg (2010)