J Comb Optim (2013) 26:44–70
DOI 10.1007/s10878-011-9431-1

Optimal key tree structure for two-user replacement and deletion problems

Weiwei Wu · Minming Li · Enhong Chen

Published online: 30 November 2011

Abstract We study the optimal tree structure for the key management problem. In the key tree, when two or more leaves are deleted or replaced, the updating cost is defined to be the number of encryptions needed to securely update the remaining keys. The objective is to find the optimal tree structure where the worst case updating cost is minimum. We extend the result of 1-replacement problem in Snoeyink et al. (Proceedings of the twentieth annual IEEE conference on computer communications, pp. 422–431, 2001) to the k-user case. We first show that the k-deletion problem can be solved by handling k-replacement problem. Focusing on the k-replacement problem, we show that the optimal tree can be computed in $O(n^{(k+1)^2})$ time given a constant k. Then we derive a tight degree bound for optimal tree when replacing two users. By investigating the maximum number of leaves that can be placed on the tree given a fixed worst case replacement cost, we prove that a balanced tree with certain root degree $5 \leq d \leq 7$ where the number of leaves in the subtrees differs by at most one and each subtree is a 2-3 tree can always achieve the minimum worst case 2-replacement cost. Thus the optimal tree for two-user replacement problem can be efficiently constructed in $O(n)$ time.

Keywords Key tree · Optimality · Updating cost · User deletion · Multicast cost

W. Wu (✉) · M. Li
Department of Computer Science, City University of Hong Kong, Hong Kong, Hong Kong
e-mail: wweiwei2@student.cityu.edu.hk

M. Li
e-mail: minmli@cs.cityu.edu.hk

E. Chen
School of Computer Science, University of Science and Technology of China, Hefei, Anhui, China
e-mail: cheneh@ustc.edu.cn

1 Introduction

In the applications that require content security, encryption technology is widely used. Asymmetric encryption is usually used in a system requiring stronger security, while symmetric encryption technology is also widely used because of the easy implementation and other advantages. In the applications such as teleconferencing and online TV, the most important security problem is to ensure that only the authorized users can enjoy the service. Centralized key management technology can achieve efficiency and satisfy the security requirement of the system. Hence, several models based on the key tree management are proposed to safely multicast the content. Two kinds of securities should be guaranteed in these applications: one is *Future Security* which prevents a deleted user from accessing the future content; the other is *Past Security* which prevents a newly joined user from accessing the past content. Key tree model, which was proposed by Wong et al. (2000) and Wallner et al. (1999), is widely studied in recent years. In this model, the Group Controller (GC) maintains a tree structure for the whole group. The root of the tree stores a Traffic Encryption Key (TEK) which is used to encrypt the content that should be broadcast to the authorized users. To update the TEK securely, some auxiliary keys are maintained. Whenever a user leaves or joins, the GC would update keys accordingly to satisfy Future Security and Past Security. Because updating keys for each user change is too frequent in some applications, Li et al. (2001) proposed *batch rekeying model* where keys are only updated after a certain period of time. Zhu et al. (2003) studied the scenario of popular services with limited resources which always has the same number of joins and leaves during the batch period (because there are always users on the waiting list who will be assigned to empty positions whenever some authorized users leave). A good survey for key tree management can be found in Goodrich et al. (2004).

An important research problem in the key tree model is to find an optimal structure for a certain pattern of user behaviors so that the total number of encryptions involved in updating the keys is minimized. Graham et al. (2007) studied the optimal structure in batch rekeying model where every user has a probability p to be replaced in the batch period. They showed that the optimal tree for n users is an n-star when $p > 1 - 3^{-1/3} \approx 0.307$, and when $p \leq 1 - 3^{-1/3}$, the optimal tree can be computed in $O(n)$ time. Specially when p approaches 0, the optimal tree resembles a balanced ternary tree to varying degrees depending on certain number-theoretical properties of n. This is then extended to include a special group of users called "loyal users" (Chan et al. 2010). The normal users have probability p to issue a leave request, while loyal users have probability zero to leave the group. Chan et al. (2009) considers the case that each member v has an update probability w_v and they aim at minimizing the average communication cost of an update. There provide a hardness result and a constant-factor approximation algorithm. Chen et al. (2008) studied the optimal structure in traditional *key tree model* under the assumption that users in the group are all deleted one by one (sequence of deletion). They show that the optimal tree is a tree where every internal node has degree at most 5 and the children of nodes which have degree $d \neq 3$ are all leaves. Wu et al. (2008a) improved the result of Chen et al. (2008) and showed that a balanced tree where every subtree has nearly the same number of leaves can achieve the optimal cost. They also investigate the optimal structure when the insertion cost in the initial setup period is considered.

More related to this paper, Snoeyink et al. (2001) proved that any distribution scheme has a worst-case cost of $\Omega(\log n)$ for deleting a user. They also found an optimal structure when only one user is deleted from the tree. In this paper, we further investigate the problem when more users are simultaneously deleted from a tree. We show that the k-deletion problem can be solved by handling the k-replacement problem. Thus we only focus on the k-replacement problem. Given a constant k, the optimal tree can be compute in $O(n^{(k+1)^2})$ time. Then we give a tight degree bound for the optimal tree when replacing two users which leads to an $O(n^8)$ algorithm. To solve the 2-replacement problem more efficiently, we then investigate the maximum number of leaves that can be placed on the tree given a fixed worst case replacement cost (denoted by capacity). Based on this, we prove that a balanced tree with certain root degree $5 \leq d \leq 7$ where the number of leaves in the subtrees differs by at most one and each subtree is a 2-3 tree can always achieve the minimum worst case 2-replacement cost. Thus the optimal tree can be efficiently constructed in $O(n)$ time. Interestingly, we propose a conjecture on the capacity function for general k-replacement problem which can be used to construct its optimal tree in linear time.

The remaining of this paper is organized as follows. We review the key tree model in Sect. 2. Section 3 shows the relationship between the k-deletion problem and the k-replacement problem. From Sect. 4 on, we focus on the two-user replacement problem. We start by deriving degree bound for this problem in Sects. 4.1 and 4.2. In Sect. 4.3, we study the maximum number of leaves that can be placed on a tree given a fixed replacement cost and use the result to prove the optimal tree structure for the 2-replacement problem. Finally, we conclude our work and propose a conjecture on the optimal tree cost for the general k-replacement problem in Sect. 5. This paper is an extended version of our ISAAC 2008 paper (Wu et al. 2008b) with all the proofs included.

2 Preliminaries

We first review the key tree model (Wong et al. 2000) which is also referred to in the literature as LKH (Logical Key Hierarchy) (Wallner et al. 1999).

In the key tree model, there is a Group Controller maintaining a key tree for the group. A leaf on the key tree represents a user and stores an individual key that is only known by this user. An internal node stores a key that is shared by all its leaf descendants. Thus a user always knows the keys stored in the path from its leaf to the root. To guarantee content security, the GC encrypts the content by the Traffic Encryption Key (TEK) which is stored in the root and then broadcast it to the users. Only the authorized users knowing the TEK can decrypt the content. When a user joins or leaves, the GC will update the keys in a bottom-up fashion. As shown in Fig. 1(a), there are 7 users in the group. We take the deletion of user u_4 as an example. Since u_4 knows k_4, k_9 and k_{10}, the GC need to update the keys k_9 and k_{10} (the node that stores k_4 disappears because u_4 is already deleted from the group). GC will encrypt the new k_9 with k_5 and broadcast it to notify u_5. Note that only u_5 can decrypt the message. Then GC encrypts the new k_{10} with k_6, k_7, k_8 and the new k_9 respectively, and then

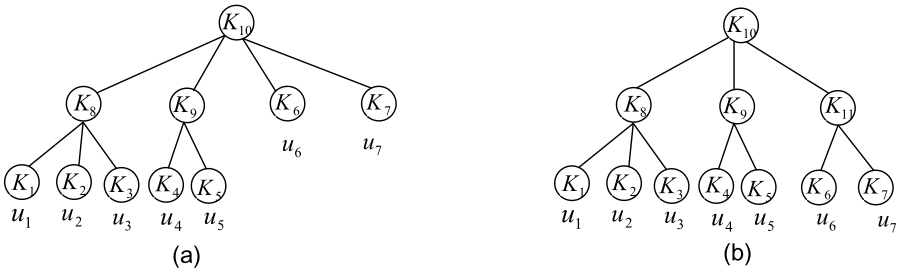

Fig. 1 Two structures of a group with 7 users

broadcast the encrypted messages to notify the users. Since all the users and only the users in the group can decrypt one of these messages, the GC can safely notify the users except user u_4 about the new TEK. The deletion cost measured as the number of encryptions is 5 in this example.

In the following, we say that a node u has degree d if it has d children in the tree. Note that the worst case deletion cost of the tree shown in Fig. 1(a) is 6 where one of the users u_1, u_2, u_3 is deleted. In Snoeyink et al. (2001), the authors investigate the optimal tree structure with n leaves where the worst case single deletion cost is minimum. Their result shows that the optimal tree is a special kind of 2-3 tree defined below. Figure 1(b) demonstrate the 2-3 tree with 7 leaves.

Definition 1 In the whole paper, we use 2-3 tree to denote a tree with n leaves constructed in the following way.

(1) When $n \geq 5$, the root degree is 3 and the number of leaves in three subtrees of the root differs by at most 1. When $n = 4$, the tree is a complete binary tree. When $n = 2$ or $n = 3$, the tree has root degree 2 and 3 respectively. When $n = 1$, the tree only consists of a root.
(2) Each subtree of the root is a 2-3 tree defined recursively.

In popular servers, since there are always users on the waiting list, the batch rekeying model will update the TEK by replacing constant k revoked users with k waiting users. The newly joined k users will take the k positions which are vacant due to the leave of k users, thus keeping the structure of the tree unchanged for each update. This reduces the update frequency. The updating cost equals the summation of degrees of all ancestors of the k old users (leaves). Thus the objective is to find an optimal tree where the worst case updating cost incurred by the k-user membership change is minimum. We denote this problem to be *k-replacement problem*. In fact, Snoeyink et al. (2001) showed that 2-3 tree is the optimal tree for 1-replacement problem. We further denote the problem to find the optimal tree when k users are deleted from the tree as *k-deletion problem*. As shown in Fig. 1(b), the tree has a worst case cost 5 for 1-deletion problem and worst case cost 6 for 1-replacement problem. We study the two-user case for both deletion and replacement problems.

We first define the k-replacement problem formally as follows.

Definition 2 Given a tree T, we denote the number of encryptions incurred by replacing $u_{i_1}, \ldots, u_{i_k}$ with k new users as $C_T(u_{i_1}, \ldots, u_{i_k}) = \sum_{v \in (\bigcup_{1 \le j \le k} ANC(u_{i_j}))} d_v$ where $ANC(u)$ is the set of u's ancestor nodes and d_v is v's degree. We use k-replacement cost to denote the maximum cost among all possible combinations and write it as $C_k(T, n) = \max_{i_1, i_2, \ldots, i_k} C_T(u_{i_1}, u_{i_2}, \ldots, u_{i_k})$.

We further define an optimal tree $T_{n,k,opt}$ (abbreviated as $T_{n,opt}$ if the context is clear) for the k-replacement problem as a tree which has the minimum k-replacement cost over all trees with n leaves, i.e. $C_k(T_{n,opt}, n) = \min_T C_k(T, n)$. We also denote this optimal cost as $OPT_k(n)$. The k-replacement problem is to find the $OPT_k(n)$ and $T_{n,k,opt}$. The k-deletion cost and the k-deletion problem are defined similarly by using the cost incurred by permanently deleting the leaves instead of the cost incurred by replacing the leaves. Note that some keys need not be updated if all its leaf descendants are deleted and the number of encryptions needed to update that key is also reduced if some branches of that node totally disappear due to deletion. We remark that the k-deletion problem and the k-replacement problem are not trivially equivalent.

3 Relationship between the k-deletion problem and the k-replacement problem

In this section, we will show that the k-deletion problem can be solved by handling the k-replacement problem.

Definition 3 We say a node v is a pseudo-leaf node if its children are all leaves. In the following two lemmas, we use t to denote the number of pseudo-leaf nodes in a tree T.

Lemma 1 *If $t \le k$, then the k-deletion cost of T is at least $n - k$.*

Proof When $t \le k$, we claim that in order to achieve k-deletion cost, we need to delete at least one leaf from each pseudo-leaf node. Suppose on the contrary there exists one pseudo-leaf node v where none of its children belongs to the k leaves we delete. We divide the discussion into two cases.

First, if each of the k leaves is a child of the remaining $t - 1$ pseudo-leaf nodes, then there exists one pseudo-leaf node u with at least two children deleted. In this case, a larger deletion cost can be achieved if we delete one child of v while keeping one more child of u undeleted.

Second, if some of the k leaves are not from the remaining $t - 1$ pseudo-leaf nodes, then we assume that u is one of them whose parent is not a pseudo-leaf. Then there exists one of u's siblings w that has at least one pseudo-leaf w' (w' can be w itself) as its descendant. If no children of w' belong to the k leaves, then deleting a child of w' while keeping u undeleted incurs larger deletion cost. If at least one child of w' belongs to the k leaves, then deleting a child of v while keeping u undeleted incurs larger deletion cost.

We see that in the worse case deletion, each pseudo-leaf node has at least one child deleted, which implies that all the keys in the remaining $n-k$ leaves should be used once as the encryption key in the updating process. Hence the k-deletion cost of T is at least $n-k$. □

Lemma 2 *If $t > k$, then the k-deletion cost is $C_k(T,n)-k$ where $C_k(T,n)$ is the k-replacement cost.*

Proof Using similar arguments as in the proof of Lemma 1, we can prove that when $t > k$, the k-deletion cost can only be achieved when the k deleted leaves are from k different pseudo-leaf nodes. Then it is easy to see that the k-deletion cost is $C_k(T,n)-k$ where $C_k(T,n)$ is the k-replacement cost. □

Although worst case costs between these two problems do not always differ by a constant k given some tree T, we can show that the worst case costs between their optimal tree structures always differ by k.

Theorem 1 *When considering trees with n leaves, the optimal k-deletion cost is $OPT_k(n)-k$ where $OPT_k(n)$ is the optimal k-replacement cost.*

Proof Note that in the tree where all n leaves have the same parent (denoted as one-level tree), the k-deletion cost is $n-k$. By Lemma 1, any tree with the number of pseudo-leaf nodes at most k has the k-deletion cost at least $n-k$. Hence we only need to search the optimal tree among the one-level tree and the trees with the number of pseudo-leaf nodes larger than k. Moreover, in the one-level tree T, the k-deletion cost is $n-k=C_k(T,n)-k$ where $C_k(T,n)$ is the k-replacement cost. Further by Lemma 2, all the trees in the scope for searching the optimal tree have k-deletion cost $C_k(T,n)-k$, which implies that the optimal k-deletion cost is $OPT_k(n)-k$ where $OPT_k(n)$ is the optimal k-replacement cost. □

The above analysis implies that the optimal tree for the k-deletion problem can be obtained by solving the k-replacement problem. Therefore, we only focus on the k-replacement problem in the following.

4 Two-user replacement problem

4.1 First step: loose degree bound for the k-replacement problem

Given a constant k, we show that k-replacement problem can be computed in polynomial time by deducing a loose degree bound. In the following proofs, we will often choose a template tree T and then construct a tree T' by removing from T some leaves together with the exclusive part of leaf-root paths of those leaves. Here, the exclusive part of a leaf-root path includes those edges that are not on the leaf-root path of any of the remaining leaves. We also say that T is a template tree of T'. By the definition of the k-replacement cost, we have the following fact.

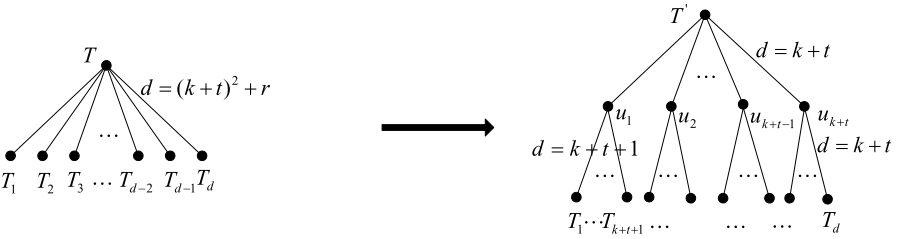

Fig. 2 Transformation of the tree which has root degree $d = (k+t)^2 + r$

Fact 2 *If T is a template tree of T', then the k-replacement cost of T' is no larger than that of T.*

Lemma 3 *$OPT_k(n)$ is non-decreasing when n increases.*

Proof Suppose on the contrary $OPT_k(n_1) > OPT_k(n_2)$ when $n_1 \le n_2$, then there exist two trees T_1 and T_2 satisfying $C_k(T_1, n_1) = OPT_k(n_1)$ and $C_k(T_2, n_2) = OPT_k(n_2)$. We can take T_2 as a template tree and delete the leaves until the number of leaves decreases to n_1. The resulting tree T_2' satisfies $C_k(T_2', n_1) \le C_k(T_2, n_2) < OPT_k(n_1)$ by Fact 2, which contradicts the definition of $OPT_k(n_1)$. The lemma is then proved. □

Lemma 4 *$T_{n,opt}$ has root degree upper bounded by $(k+1)^2 - 1$.*

Proof We divide the value of root degree $d \ge (k+1)^2$ into two sets, $\{d|(k+t)^2 \le d < (k+t)(k+t+1), d, k, t \in N, t \ge 1\}$ and $\{d|(k+t-1)(k+t) \le d < (k+t)^2, d, k, t \in N, t \ge 2\}$. Take $k = 2$ for instance, the first set is $\{9, 10, 11, 16, 17, 18, 19, 25, \ldots\}$ while the other is $\{12, 13, 14, 15, 20, 21, 22, 23, \ldots\}$.

Case 1: $(k+t)^2 \le d < (k+t)(k+t+1)$ $(t \ge 1)$.

We write d as $(k+t)^2 + r$ where $0 \le r < k+t$. Given a tree T, we can transform it into a tree with root degree $k+t$ as Fig. 2 shows. In the resulting tree T', subtrees $T_{u_1}, \ldots, T_{u_{k+t}}$ are $k+t$ subtrees where the root $u_1, \ldots, u_{k+t}$ are on level one. Among the $k+t$ subtrees, there are r subtrees with root degree $k+t+1$ and $k+t-r$ subtrees with root degree $k+t$. Suppose that the k-replacement cost of T' is incurred by replacing $k_1, k_2, \ldots, k_s$ users from subtrees $T_{i_1}, T_{i_2}, \ldots, T_{i_s}$ respectively where $k_1 + k_2 + \cdots + k_s = k$ and $s \le k$. The corresponding cost is $C_k(T', n) = \sum_{j=1}^{s} C_{k_j}(T_{i_j}, n_{i_j}) + D_0$ where n_{i_j} is the number of leaves in T_{i_j} and D_0 is the cost incurred in the first two levels.

In the original tree T, the corresponding cost for replacing those leaves is $C_k(T, n) = \sum_{j=1}^{s} C_{k_j}(T_{i_j}, n_{i_j}) + d = \sum_{j=1}^{s} C_{k_j}(T_{i_j}, n_{i_j}) + (k+t)^2 + r$. We will prove that when $t \ge 1$ we always have $C_k(T, n) \ge C_k(T', n)$, i.e. $D_0 \le (k+t)^2 + r$.

Firstly, if $r \le k$, the cost D_0 is at most $r(k+t+1) + (k-r)(k+t) + k + t$ where there are r users coming from r subtrees with root degree $k+t+1$ and $k-r$ users coming from $k-r$ subtrees with root degree $k+t$. Therefore, we have

$$D_0 \le (k+t+1)r + (k+t)(k-r) + k + t = (k+t)(k+1) + r \le (k+t)^2 + r.$$

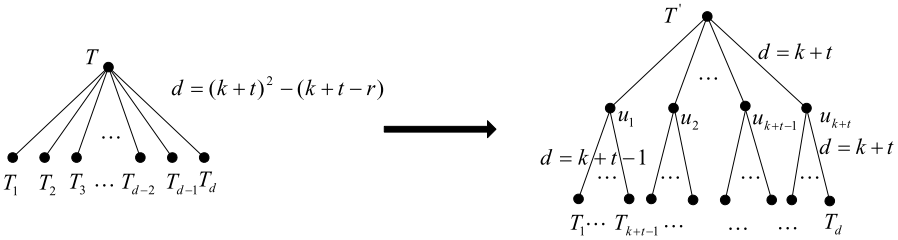

Fig. 3 Transformation of the tree which has root degree $d = (k+t)^2 - (k+t-r)$

Secondly, if $r > k$, the cost D_0 is at most $(k+t) + (k+t+1)k$ where the k users are all from k subtrees which have root degree $k+t+1$. Therefore, we have

$$D_0 \le (k+t) + (k+t+1)k \le (k+t)(k+1) + k \le (k+t)^2 + r.$$

Hence, in both situations, the condition $t \ge 1$ ensures that the transformation from T to T' does not increase the k-replacement cost.

Case 2: $(k+t-1)(k+t) \le d < (k+t)^2$ $(t \ge 2)$.

In this case, we write d as $(k+t)^2 - (k+t-r)$ where $0 \le r < k+t$. We transform T into a tree with root degree $k+t$ where there are r subtrees with root degree $k+t$ and $k+t-r$ subtrees with root degree $k+t-1$, as Fig. 3 shows. Similarly for $r \le k$, we have

$$\begin{aligned} D_0 &\le (k+t)r + (k+t-1)(k-r) + k + t = (k+t)(k+1) - (k-r) \\ &\le (k+t)^2 - (k+t-r). \end{aligned}$$

The last inequality holds because $t \ge 2$.

While for $r > k$, we have

$$D_0 \le (k+t)k \le (k+t)^2 - (k+t-r).$$

Thus, $t \ge 2$ ensures that the transformation from T to T' does not increase the k-replacement cost.

Therefore, for any root degree $d \ge (k+1)^2$, through a series of transformations, we can transform the tree into a tree with root degree less than $(k+1)^2$ without increasing the k-replacement cost. For example, when $d = 120$ and $k = 2$, we have $120 = (k+9)^2 - 1$. Firstly we transform it into a tree with root degree $k + 9 = 11$. Since $11 = (k+1)^2 + 2$ is still greater than $(k+1)^2$, we further transform the resulting tree into a tree with root degree $k + 1 = 3$. Because $3 < (2+1)^2$, we stop our transformation. The lemma is finally proved. □

Lemma 4 suggests that we can find an optimal tree for the k-replacement problem among trees whose root degree is at most $(k+t)^2 - 1$. Note that our degree bound in Lemma 4 is only for the root. We can also extend this property to all the internal nodes. This leads to a polynomial time algorithm to compute the optimal k-replacement tree given a constant k.

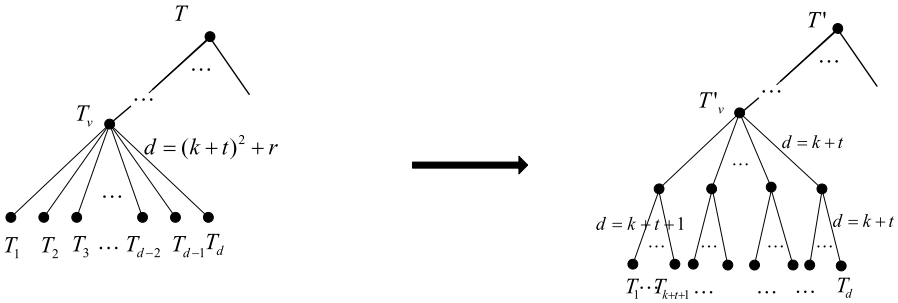

Fig. 4 Transformation of tree T which has an internal node with degree $d \geq (k+t)^2$

Theorem 3 *Any internal node in $T_{n,opt}$ has degree upper bounded by $(k+1)^2 - 1$. Given a constant k, the k-replacement problem can be computed in $O(n^{(k+1)^2})$ time.*

Proof Suppose T is an optimal tree and has an internal node v which has degree $d_v \geq (k+1)^2$, as shown in Fig. 4. We transform the subtree T_v rooted at v into T'_v in the similar way as in Lemma 4 (The resulting tree is denoted by T'). We will prove that the k-replacement cost in the resulting tree T' does not increase. We consider all the three possible relative positions of those k leaves whose membership change achieves k-replacement cost in T'. First, if none of the leaves are from T'_v, then the replacement cost of the corresponding leaves in T will be the same as the replacement cost in T'. If there are m ($m < k$) leaves from T'_v while the others are not, then replacing the corresponding leaves in T incurs no smaller replacement cost compared to that in T', because replacing m leaves of T_v incurs no smaller cost than replacing the corresponding m leaves in T'_v due to similar analysis shown in Lemma 4. If all the k leaves are from T'_v, then replacing the corresponding leaves in T again incurs no smaller replacement cost than T' by Lemma 4. In all the above three cases, T' has replacement cost no greater than T, which enables us to transform the degree of every internal node to be equal to or less than $(k+1)^2 - 1$ without increasing the k-replacement cost. By enumeration all the possible degrees of the internal node, we can design a $O(n^{(k+1)^2})$ time dynamic programming algorithm to iterative compute the optimal k-replacement cost $OPT_k(i)$ $(1 \leq i \leq n)$. By keeping the branching information for each iteration, we can construct the optimal k-replacement tree with the same complexity. □

4.2 Tight degree bound for 2-replacement problem

From this subsection on, we focus on the 2-replacement problem to derive efficient algorithm.

Definition 4 We denote the maximum cost to delete a single leaf in a tree T as S_T and the maximum cost to delete two leaves in T as D_T, i.e. $S_T = C_1(T, n)$ and $D_T = C_2(T, n)$.

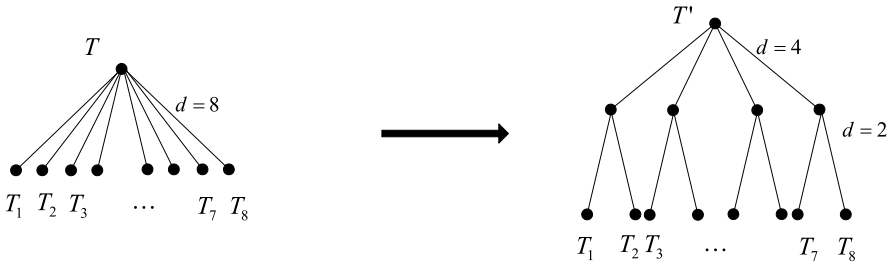

Fig. 5 Transformation of tree T which has an internal node with degree 8

According to Theorem 3, for the 2-replacement problem, $T_{n,opt}$ has degree upper bounded by 8. Furthermore, we need not consider root degree $d = 1$ when we are searching for the optimal tree. Because merging the root and its single child into one node can strictly decrease the worst case updating cost.

Fact 4 *For the* 2*-replacement problem, suppose that a tree* T *has root degree* d *where* $d \geq 2$ *and the* d *subtrees are* $T_1, T_2, \ldots, T_d$*. We have*

$$D_T = \max_{1 \leq i,j \leq d} \{D_{T_i} + d, S_{T_i} + S_{T_j} + d\}.$$

Proof We know that replacing any two leaves from a subtree T_i will incur a cost at most $D_{T_i} + d$, while replacing two leaves from two different subtrees T_i and T_j will incur a cost at most $S_{T_i} + S_{T_j} + d$. The 2-replacement cost comes from one of the above cases and therefore the fact holds. □

We can further remove the possibility of degree 8 by the following lemma.

Lemma 5 *For the* 2*-replacement problem, we can find an optimal tree among the trees with node degrees bounded between* 2 *and* 7*.*

Proof By Theorem 3, we only need to remove the possibility of degree 8 for any internal node. We first prove that we can transform any tree with root degree 8 into a tree with a smaller root degree and no larger 2-replacement cost. Given a tree T with root degree 8 and subtrees $T_1, T_2, \ldots, T_8$. We transform the structure at the root v into degree 4, and each child of v has degree 2, as shown in Fig. 5. In the original tree, 2-replacement cost is $D_T = \max\{D_{T_i} + 8, S_{T_i} + S_{T_j} + 8\}$ (Fact 4). In the resulting tree T', the cost of replacing two leaves from different subtree T_i and T_j is at most $S_{T_i} + S_{T_j} + 8$, and the cost of replacing two leaves from one subtree T_i is at most $D_{T_i} + 6$. Hence, 2-replacement cost in T' is at most $\max\{D_{T_i} + 6, S_{T_i} + S_{T_j} + 8\} \leq D_T$.

Note that the 1-replacement cost of T also does not increase in the transformation. We then can extend this transformation to any internal node as we have shown in the proof of Theorem 3. The lemma is then proved. □

Based on this bound, the optimal tree can be computed in $O(n^8)$ time by enumerating all possible degrees that are at most 7 using a dynamic programming algorithm.

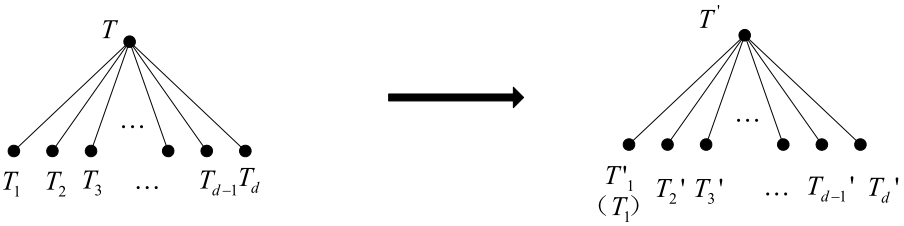

Fig. 6 Transformation of a tree which has $S_{T_1} < S_{T_2}$

However, $O(n^8)$ is still a bit unaffordable in real computations. Therefore, in the following, we reduce the complexity to $O(n)$ by accurately determining the structure of the optimal tree. In the following, we show two important properties of the optimal tree (monotone property and 2-3 tree property) and then further remove the possibility of root degree 2 and 3 to reduce the scope of trees within which we search for the optimal tree.

Lemma 6 (Monotone property) *For the* 2*-replacement problem*, *suppose a tree* T *has root degree* d *where* $d \geq 2$ *and* d *subtrees are* $T_1, T_2, \ldots, T_d$. *Without loss of generality*, *we assume that* T *has a non-increasing leaf descendant vector* $(n_1, n_2, \ldots, n_d)$, *where* n_i *is the number of leaves in subtree* T_i. *Then*, *there exists an optimal tree where* $S_{T_1} \geq S_{T_2} \geq \cdots \geq S_{T_d}$ *and* T_i *is a template of* T_{i+1} *for* $2 \leq i \leq d-1$.

Proof If a given optimal tree contradicts the lemma, we prove that we can always change this tree into a tree satisfying the monotone property but with equal or less 2-replacement cost.

Case 1: $S_{T_1} < S_{T_2}$.

We first Let T_1' be the same as T_1, and then choose T_1' to be the template tree and construct a subtree T_2' by removing $n_1 - n_2$ leaves from T_1. Similarly, we construct the subtrees T_i' $(3 \leq i \leq d)$ by removing $n_{i-1} - n_i$ leaves from T_{i-1}', as shown in Fig. 6. Since each subtree T_i' is a template of T_{i+1}', the final set of new trees satisfy $S_{T_1'} \geq S_{T_2'} \geq \cdots \geq S_{T_d'}$, and $D_{T_1'} \geq D_{T_2'} \geq \cdots \geq D_{T_d'}$. We then prove that 2-replacement cost is not increased in the resulting tree T'. Note that $S_{T_1'} = S_{T_1} < S_{T_2}$ and $D_{T_1'} = D_{T_1}$. Since $D_{T'} = \max_{1 \leq i < j \leq d}\{D_{T_i'} + d, S_{T_i'} + S_{T_j'} + d\} = \max\{D_{T_1} + d, S_{T_1'} + S_{T_2'} + d\}$ and $D_T \geq \max\{D_{T_1} + d, S_{T_1} + S_{T_2} + d\}$, we have $D_{T'} < D_T$ because $S_{T_1'} + S_{T_2'} < S_{T_1} + S_{T_2}$. Hence, the lemma holds in this case.

Case 2: $S_{T_1} \geq S_{T_2} \geq \cdots \geq S_{T_j}$ and $S_{T_j} < S_{T_{j+1}}$.

In this case, we let T_1', T_2' be the same as T_1, T_2 respectively. Then we choose T_2' to be the template tree and construct a subtree T_3' by removing $n_2 - n_3$ leaves from T_2. Similarly, we construct subtrees T_i' $(4 \leq i \leq d)$ by removing $n_{i-1} - n_i$ leaves from T_{i-1}', as shown in Fig. 7. Since each subtree T_i' is a template of T_{i+1}', the final set of new trees satisfy $S_{T_1} \geq S_{T_2} \geq S_{T_3'} \geq \cdots \geq S_{T_d'}$ and $D_{T_2} \geq D_{T_3'} \geq \cdots \geq D_{T_d'}$. Therefore, we have $D_{T'} = \max_{1 \leq i,j \leq d}\{D_{T_i'} + d, S_{T_i'} + S_{T_j'} + d\} = \max\{D_{T_1} + d, D_{T_2} + d, S_{T_1} + S_{T_2} + d\}$, which implies $D_{T'} \leq D_T$.

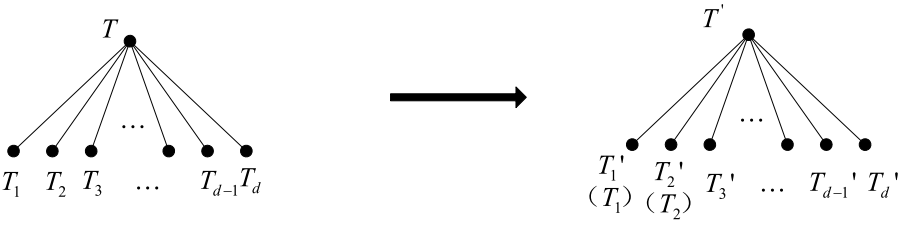

Fig. 7 Transformation of a tree T which has $S_{T_j} < S_{T_{j+1}}$ where $j \geq 2$

The 2-replacement cost is not increased in T' in both cases and therefore the lemma is proved. □

Fact 5 *For trees satisfying the monotone property* (*Lemma* 6), *we have*

$$D_T = \max\{D_{T_1} + d, S_{T_1} + S_{T_2} + d\}.$$

Proof By Fact 4 we have $D_T = \max_{1\leq i,j\leq d}\{D_{T_i} + d, S_{T_i} + S_{T_j} + d\}$. Lemma 6 further ensures that $\max_{1\leq i,j\leq d}\{D_{T_i} + d, S_{T_i} + S_{T_j} + d\} = \max\{D_{T_1} + d, D_{T_2} + d, S_{T_1} + S_{T_2} + d\}$. Since $D_{T_2} < 2S_{T_2} \leq S_{T_1} + S_{T_2}$, we have $D_T = \max\{D_{T_1} + d, S_{T_1} + S_{T_2} + d\}$. □

In the following, we further reduce the scope for searching the optimal tree by proving the following lemma.

Lemma 7 (2-3 tree property) *For a tree T satisfying Lemma* 6, *we can transform subtrees $T_2, \ldots, T_d$ into* 2-3 *trees without increasing the* 2*-replacement cost*.

Proof Given a tree T satisfying Lemma 6 and Fact 5, we transform subtrees $T_2, T_3, \ldots, T_d$ into 2-3 trees $T_2', T_3', \ldots, T_d'$ to get a new tree T'. For $2 \leq i \leq d$, since $S_{T_i'} = OPT_1(n_i)$, we have $S_{T_d'} \leq \cdots \leq S_{T_3'} \leq S_{T_2'} \leq S_{T_2}$ (Lemma 3) and $D_{T_i'} \leq 2S_{T_i'} \leq 2S_{T_2'} \leq S_{T_1} + S_{T_2'}$ $(2 \leq i \leq d)$. Thus $D_{T'} = \max\{D_{T_1} + d, D_{T_2'} + d, S_{T_1} + S_{T_2'} + d\} \leq \max\{D_{T_1} + d, S_{T_1} + S_{T_2} + d\} = D_T$, which implies the transformation does not increase 2-replacement cost. The lemma is then proved. □

We denote the trees satisfying Lemma 7 as *candidate-trees*. By Lemma 7, we can find an optimal tree among all the candidate trees. For a candidate tree T with root degree d, we define branch B_i to be the union of T_i and the edge connecting the root of T with the root of T_i. We say branch B_1 is the *dominating branch* and other branches $B_2, \ldots, B_d$ are *ordinary branches*. To make the discussion self-contained, we introduce the following two properties.

Fact 6 *Suppose that $c(u)$ is the ancestor weight of leaf u, i.e. $c(u) = \sum_{v\in ANC(u)} d_v$ where $ANC(u)$ is the set of u's ancestor nodes and d_v is v's degree, then the cost of replacing two leaves u_1 and u_2 is $c(u_1, u_2) = c(u_1) + c(u_2) - c(v) - d_v$ where v is the nearest common ancestor of u_1 and u_2.*

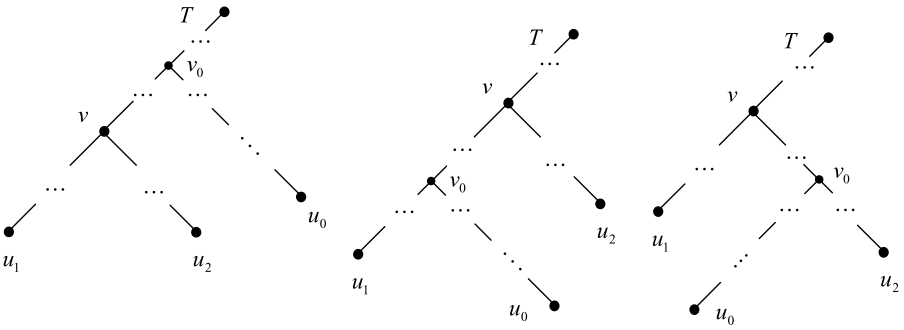

Fig. 8 The three cases of u_0 in the proof of Fact 8

Lemma 8 *In a tree T, assume that u_1, u_2 are the two leaves which incur the 2-replacement cost ($c(u_1, u_2) = D_T$) and v is the nearest common ancestor of u_1 and u_2, then*

(1) *one of the leaves satisfies $c(u) = S_T$ (let this leaf be u_1) and therefore $D_T = c(u_1) + c(u_2) - c(v) - d_v = S_T + c(u_2) - c(v) - d_v$.*
(2) $c(u_2) - c(v) - d_v \leq S_T - 1$.

Proof The fact implies that if two leaves' replacement cost in T is maximum, then at least one of the leaves has the maximum ancestor weight S_T. Suppose on the contrary that two leaves u_1, u_2 satisfy $c(u_1, u_2) = D_T$ but $c(u_1) < S_T$ and $c(u_2) < S_T$. Then, for a leaf u_0 which has $c(u_0) = S_T$, we prove $c(u_0, u_1) > c(u_1, u_2)$ or $c(u_0, u_2) > c(u_1, u_2)$, which contradicts $c(u_1, u_2) = D_T$. Figure 8 shows the transformations for the following three cases.

Case 1: u_0, u_1's nearest common ancestor $v_0 \in ANC(v)$.

In this case, we have $c(u_0, u_2) = c(u_0) + c(u_2) - c(v_0) - d_{v_0}$ and therefore $c(u_0, u_2) - c(u_1, u_2) = c(u_0) - c(u_1) + c(v) + d_v - c(v_0) - d_{v_0}$. Since $c(u_0) > c(u_1)$ and $c(v_0) + d_{v_0} \leq c(v)$ (v_0 is v_1's ancestor), we have $c(u_0, u_2) > c(u_1, u_2)$, the lemma is proved in this case.

Case 2: u_0, u_1's nearest common ancestor $v_0 \in ANC(u_1)$ but $v_0 \notin ANC(v)$.

In this case, we have $c(u_0, u_2) - c(u_1, u_2) = c(u_0) - c(u_1) > 0$.

Case 3: u_0, u_2's nearest common ancestor $v_0 \in ANC(u_2)$ but $v_0 \notin ANC(v)$.

In this case, we have $c(u_0, u_1) - c(u_1, u_2) = c(u_0) - c(u_2) > 0$.

In all the three cases, we have either $c(u_0, u_1) > c(u_1, u_2)$ or $c(u_0, u_2) > c(u_1, u_2)$. Therefore, Item (1) of the fact is proved.

Item (2) is correct because $c(u_2) \leq S_T$. □

Through a series of lemmas in the Appendix, we are able to prove the following theorem to further remove the possibility of root degree 2 and 3 in the optimal tree. While removing large degree $d \geq 8$ is easy, it is comparatively complicate to remove small degrees 2 or 3.

Theorem 7 *For the 2-replacement problem, a tree T with root degree 2 or 3 can be transformed into a tree with root degree 4 without increasing the 2-replacement cost.*

4.3 Linear time algorithm: balanced structure of 2-replacement problem

Note that the optimal structure can be computed in $O(n^8)$ time by enumerating all possible degrees that are at most 7 by the result of Sect. 4.2. Although we have removed the possibility of the root degree 2 and 3 in Sect. 4.2 and have fixed the structure of the ordinary branches, we still do not have an effective algorithm to exactly compute the optimal structure because we need to enumerate all the possible structures of the dominating branch. In this subsection, we will prove that among the candidate trees with n leaves, a balanced structure can achieve 2-replacement cost $OPT_2(n)$. This reduces the complexity of the algorithm substantially to $O(n)$.

The basic idea is to first investigate the *capacity* $g(R)$ defined below for candidate trees with 2-replacement cost R (Theorem 8). Note that the optimal tree has the minimum 2-replacement cost with n leaves, which reversely implies that if we want to find a tree with 2-replacement cost R and at the same time has the maximum possible number of leaves, then computing the optimal tree for increasing n until $OPT_2(n) > R$ will produce one such solution. We then analyze and calculate the exact value for the capacity (maximum number of leaves) given a fixed 2-replacement cost R (Theorem 11). Finally, we prove that certain balanced structure can always be the optimal structure that minimizes the 2-replacement cost (Theorem 12).

Definition 5 We use capacity to denote the maximum number of leaves that can be placed in a certain type of trees given a fixed replacement cost. According to Snoeyink et al. (2001), function $f(r)$ defined below is the capacity for 1-replacement cost r (among all the possible trees). We use function $g(R)$ to denote the capacity for 2-replacement cost R (among all the possible trees). In other words, when $g(R-1) < n \leq g(R)$, we have $OPT_2(n) = R$.

$$f(r) = \begin{cases} 3 \cdot 3^{i-1} & \text{if } r = 3i \\ 4 \cdot 3^{i-1} & \text{if } r = 3i+1 \\ 6 \cdot 3^{i-1} & \text{if } r = 3i+2 \end{cases}$$

To facilitate the discussion, according to Fact 5, we can divide the candidate trees with 2-replacement cost R and root degree d into two categories as summarized in the following definition.

Definition 6 Candidate trees of category 1: The two leaves whose replacement cost achieves 2-replacement cost are from different branches, i.e. $D_T = S_{T_1} + S_{T_2} + d$, which implies $S_{T_1} + S_{T_2} \geq D_{T_1}$.

Candidate trees of category 2: The two leaves whose replacement cost achieves 2-replacement cost are both from the dominating branch B_1, i.e. $D_T = D_{T_1} + d$, which implies $S_{T_1} + S_{T_2} < D_{T_1}$.

Correspondingly, we denote the capacity of the candidate trees belonging to category 1 with 2-replacement cost R by $g_1(R)$ and denote the capacity of the candidate trees belonging to category 2 with 2-replacement cost R by $g_2(R)$. Note that we can find the optimal tree among the candidate trees according to Lemma 7, which

implies that with the same 2-replacement cost R, the best candidate tree can always have equal or larger number of leaves than the general trees. That is, we have $g(R) = \max\{g_1(R), g_2(R)\}$. Thus in the following discussions, we only focus on the candidate trees. On the other hand, because we are looking for trees with the maximum number of leaves, it is easy to see that we can assume the number of leaves in ordinary branches are all the same (Otherwise, we can make the tree bigger without affecting the 2-replacement cost).

In all candidate trees with 2-replacement cost R, by Fact 5, we only need to consider the case where at most one of the two leaves whose replacement cost achieves 2-replacement cost is from an ordinary branch. Suppose each ordinary branch has 1-replacement cost r^-, and correspondingly T_1 has 1-replacement cost r^+ where $r^+ \leq R - d - r^-$ (otherwise we have $D_T \geq r^+ + r^- + d > R$, a contradiction). For fixed cost R, Lemma 6 (monotonous property) implies that $r^+ \geq r^-$. We first prove the following capacity bound.

Theorem 8 *We have* $g_i(R) \leq (R - 2r^-) \cdot f(r^-)$ $(i = 1, 2)$.

To prove this theorem, we will respectively study the two categories of candidate trees to prove $g_1(R) \leq (R - 2r^-) \cdot f(r^-)$ and $g_2(R) \leq g_1(R)$. We use critical node defined below to denote a special kind of nodes in the following proofs.

Definition 7 For a selected node u, we say the path from u to the root is a critical path. Correspondingly, u's ancestor nodes on the path are named critical nodes.

We first investigate the capacity of the candidate tree T in category 1 which has 2-replacement cost R. In Lemmas 9, 10, 11, 12, we assume that the number of leaves in T is the maximum possible and try to fix the $r_{\max}(\cdot)$ (Definition 8) of some critical nodes and their siblings to prove the capacity bound $g_1(R)$.

Definition 8 For the subtree T_v rooted at node v, the maximum ancestor weight with respect to T_v is denoted as $r_{\max}(v)$. Correspondingly, the maximum number of leaves T_v can hold given the 1-replacement cost $r_{\max}(v)$ is denoted as $cap(v)$.

Given a candidate tree in category 1 where $D_T = S_{T_1} + S_{T_2} + d$, there exists at least one leaf $u \in B_1$ satisfying $c(u) = r^+ + d = R - r^-$. We choose such a leaf u to obtain a critical path, correspondingly u's ancestors are critical nodes. Notice that critical node v has $r_{\max}(v) = R - r^- - c(v)$. Furthermore, without loss of generality, we suppose v_1 is the nearest ancestor of u which satisfies $r_{\max}(v_1) \geq r^-$ (which implies $r_{\max}(v_0) < r^-$ where critical node v_0 is v_1's child). In the following, we investigate the capacity for the critical node and their siblings.

Lemma 9 *If node v is a critical node, then for any sibling v' of v, we have* $r_{\max}(v') = \min\{r^-, r_{\max}(v)\}$.

Proof By the definition of r^+ we have $r_{\max}(v') + c(v') - d \leq r^+ = r_{\max}(v) + c(v) - d$, which implies $r_{\max}(v') \leq r_{\max}(v)$. On the other hand, if we choose to delete a

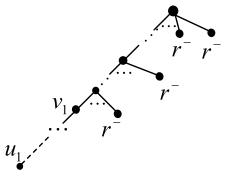

Fig. 9 *spine*$(v_1, root)$ that belongs to candidate trees in category 1

leaf u in T_v and a leaf w in $T_{v'}$, the replacement cost should be at most R, which means $r_{\max}(v) + r_{\max}(v') + c(v) \leq R$. Note that $r_{\max}(v) = R - r^- - c(v)$. We then have $r_{\max}(v') \leq r^-$. The possibility $r_{\max}(v') < \min\{(r^-, r_{\max}(v))\}$ can be removed because otherwise the capacity of $T_{v'}$ is not maximum which contradicts our assumption that the number of leaves in T is the maximum possible. In other words, if $r_{\max}(v') < \min\{r^-, r_{\max}(v)\}$, we can enlarge the tree by increasing $r_{\max}(v')$ to $\min\{r^-, r_{\max}(v)\}$ while ensuring that the tree still belongs to candidate trees in category 1 and has 2-replacement cost R. This ends the proof. □

Lemma 10 *Given a critical node v which is on the path from v_1 to the root, if v' is a sibling of v, we have $r_{\max}(v') = r^-$.*

Proof For the critical node v, we have $r_{\max}(v) = R - r^- - c(v)$. Furthermore, since v is v_1's ancestor or v_1 itself, we have $r_{\max}(v) \geq r_{\max}(v_1) \geq r^-$. Hence, the sibling of v has $r_{\max}(v') = \min\{r^-, r_{\max}(v)\} = r^-$ (Lemma 9). The lemma is proved. □

Consider the structure composed by the critical nodes on the path from v_1 to the root and the siblings of these critical nodes, as shown in Fig. 9. This structure is denoted as *spine*$(v_1, root)$ according to the definition below. Suppose that w is a leaf in this structure. The value of $r_{\max}(w)$ is r^- by Lemma 10.

Definition 9 Suppose v_a and v_b (v_a is a descendent of v_b) are critical nodes on the selected critical path, we denote the structure composed by the critical nodes from v_a to v_b and their siblings as *spine*(v_a, v_b) (v_b's siblings are not included).

We will consider two cases of *spine*$(v_1, root)$ for candidate trees in category 1 and prove $g_1(R) \leq (R - 2r^-) \cdot f(r^-)$ in the following.

Case 1: $r_{\max}(v_1) = r^-$.
Case 2: $r_{\max}(v_1) > r^-$.

Lemma 11 *When $r_{\max}(v_1) = r^-$, we have $g_1(R) \leq (R - 2r^-) \cdot f(r^-)$.*

Proof In this case, besides node v_1, any other leaf w of *spine*$(v_1, root)$ also has $r_{\max}(w) = r^-$ according to Lemma 10. On the other hand, we know that the total degrees from the parent of v_1 to the root is $c(v_1) = R - r^- - r_{\max}(v_1) = R - 2r^-$, which equals the number of leaves on *spine*$(v_1, root)$. Therefore, the capacity is bounded by $(R - 2r^-) \cdot f(r^-)$. □

Lemma 12 *When $r_{\max}(v_1) > r^-$, we have $g_1(R) \leq (R - 2r^-) \cdot f(r^-)$.*

Proof Theorem 3 implies that $d_{v_1} \leq 7$, thus $r_{\max}(v_1) = r_{\max}(v_0) + d_{v_1} \leq r^- + 6$ (because $r_{\max}(v_0) < r^-$).

Similar with the proof in the previous lemma, for any leaf v except v_1 on $spine(v_1, root)$, we have $cap(v) \leq f(r^-)$, which results in at most $(R - r^- - r_{\max}(v_1) - 1) \cdot f(r^-)$ leaves. To prove the capacity bound, we only need to show that $cap(v_1) \leq (r_{\max}(v_1) + 1 - r^-) \cdot f(r^-)$. If we write $r_{\max}(v_1)$ as $r^- + m$ where $m \leq 6$, then it is equivalent to proving $(m+1) \cdot f(r^-) - f(r^- + m) \geq 0$. When $1 \leq m \leq 5$, we can verify that $f(r^- + m) < (m+1) f(r^-)$. When $m = 6$, the above relation does not hold because $f(r^- + 6) = 9 f(r^-) > 7 f(r^-)$. Therefore we apply another method. We know that when $r_{\max}(v_1) = r^- + m = r^- + 6$, we have $d_{v_1} = 7$. Lemma 9 implies that the sibling u of v_0 has $r_{\max}(u) = \min\{r^-, r_{\max}(v_0)\} = r_{\max}(v_0) = r^- + 6 - 7 = r^- - 1$. Hence $cap(v_1) \leq 7 f(r^- - 1) \leq 7 f(r^-)$. This completes the proof. □

Until now, we only focus on the candidate trees in category 1. By Lemma 11 and 12 we have proved the capacity bound of candidate trees in category 1, i.e $g_1(R) \leq (R - 2r^-) \cdot f(r^-)$. We then prove that candidate trees in category 2 has a capacity no greater than that of category 1.

Note that the following fact holds for candidate trees T in category 2.

Fact 9 *Given a candidate tree T in category 2, suppose that u_1, u_2 are the two leaves (from subtree T_1) whose replacement cost equals R, i.e. $c(u_1) + c(u_2) - c(v) - d_v = R$ where v is the nearest common ancestor of u_1, u_2 and $c(u_1) = S_T$. In order to find an optimal tree, among candidate trees in category 2, we only need to search among those trees satisfying $c(u_2) - c(v) - d_v < OPT_1(n_1)$ where n_1 is the number of leaves in T_1.*

Proof If on the contrary in the optimal tree, we have $c(u_2) - c(v) - d_v \geq OPT_1(n_1)$, then we have $D_T = c(u_1) + c(u_2) - c(v) - d_v \geq 2OPT_1(n_1) + d$ since $c(u_1) = S_{T_1} + d \geq OPT_1(n_1) + d$. In this case, if we transform T into T' by replacing subtree T_1 with a 2-3 tree with the same number of leaves, we have $D_{T'} \leq D_T$ because $S_{T_1'} = OPT_1(n_1) \leq S_{T_1}$ and $D_{T_1'} + d < 2OPT_1(n_1) + d \leq c(u_1) + c(u_2) - c(v) - d_v = R$. We know that T' is either a candidate tree in category 1 or a candidate tree in Category 2 satisfying $c(u_2) - c(v) - d_v < OPT_1(n_1)$. Thus the fact follows. □

Lemma 13 *A candidate tree in category 2 has capacity $g_2(R) \leq g_1(R)$.*

Proof Given a candidate tree T in category 2, we have $S_{T_1} + S_{T_2} + d < D_{T_1} + d = R = D_T$. By the monotone property in Lemma 6, the d subtrees satisfy $S_{T_1} \geq S_{T_2} \geq \cdots \geq S_{T_d}$. If $S_{T_1} = S_{T_2}$, then $D_{T_1} + d \leq 2S_{T_1} + d = S_{T_1} + S_{T_2} + d$ which leads to a contradiction. Thus we have $S_{T_1} \geq S_{T_2} + 1$. Note that the ordinary branches are 2-3 trees, we do a tree expansion by adding one leaf to each of the ordinary branches (where each resulting subtree is still a 2-3 tree). In this way, we increase the number of leaves in the tree without affecting the 2-replacement cost.

The resulting subtree's 1-replacement cost will increase by at most 1. Suppose that two leaves in the resulting tree T' whose replacement cost achieves the 2-replacement cost are u_1 and u_2. We discuss three cases. If u_1, u_2 are both from T_1, then replacing

u_1, u_2 incurs a cost $D_{T_1} + d = R$. If one of the two leaves is from T_1 while the other is from T_i, then replacing u_1, u_2 incurs a cost at most $S_{T_1} + S_{T_2} + 1 + d \le R$. If u_1, u_2 are both from the ordinary branches, then replacing u_1, u_2 incurs a cost at most $2(S_{T_2} + 1) + d \le S_{T_2} + 1 + S_{T_1} + d \le R$.

We can apply the above tree expansion until $S_{T_1} + S_{T_i} + d = R$ (The tree becomes a candidate tree of category 1) and denote the final tree as T''. We will prove that T'' is a candidate tree of category 1. To show this, we only need to prove that in T'' subtree T_2'' has the number of leaves less than T_1. This follows because $OPT_1(n_2'') = S_{T_2''} = R - S_{T_1} - d = c(u_2) - c(v) - d_v < OPT(n_1)$ implies $n_2'' < n_1$.

Therefore, with 2-replacement cost R fixed, there is always a candidate tree of category 1 which has more leaves than any candidate tree of category 2. This finishes the proof of $g_2(R) \le g_1(R)$. □

Lemma 13 implies that the maximum capacity can always be achieved by the candidate trees in category 1. Thus the capacity bound is $(R - 2r^-) \cdot f(r^-)$ and Theorem 8 is then proved.

In the following, among these candidate trees we study the optimal structure which achieves the maximum capacity for different values of 2-replacement cost R. We will only elaborate the proof for $R = 6i + 6$ due to the similarity of other proofs. The correctness of other proofs can be referred to in Table 1. The following fact is used in our proofs.

Fact 10

$$(s + 2t) \cdot f(r - t) \le s \cdot f(r) \quad \textit{when} \begin{cases} r = 3i, & t \ge 1,\ s \ge 4 \\ r = 3i + 1, & t \ge 2,\ s \ge 4 \\ r = 3i + 1, & t \ge 1,\ s \ge 6 \\ r = 3i + 2, & t \ge 1,\ s \ge 4. \end{cases}$$

Proof We only need to prove $\frac{f(r-t)}{f(r)} \le \frac{s}{s+2t}$. Since $\frac{s}{s+2t}$ is a monotonously increasing relative to s, thus when $s \ge 4$ it is sufficient to prove $\frac{f(r-t)}{f(r)} \le \frac{4}{4+2t}$. By the definition of function $f(\cdot)$, we discuss about three cases.

Case 1: $r = 3i$.

We have

$$\frac{f(3i - t)}{f(3i)} = \begin{cases} \frac{f(3i-1-3j)}{f(3i)} = \frac{6 \cdot 3^{i-2}/3^j}{3^i} = \frac{2}{3} \cdot \frac{1}{3^j} \le \frac{4}{4+2(3j+1)} & \text{if } t = 3j + 1 \\ \frac{f(3i-2-3j)}{f(3i)} = \frac{4 \cdot 3^{i-2}/3^j}{3^i} = \frac{4}{9} \cdot \frac{1}{3^j} < \frac{4}{4+2(3j+2)} & \text{if } t = 3j + 2 \\ \frac{f(3i-3-3j)}{f(3i)} = \frac{3 \cdot 3^{i-2}/3^j}{3^i} = \frac{1}{3} \cdot \frac{1}{3^j} < \frac{4}{4+2(3j+3)} & \text{if } t = 3j + 3 \end{cases}$$

Thus in this case, we have $(s + 2t) \cdot f(r - t) \le s \cdot f(r)$ when $s \ge 4$ and $t \ge 1$.

Case 2: $r = 3i + 2$.

We have

$$\frac{f(3i + 2 - t)}{f(3i + 2)} = \begin{cases} \frac{2}{3} \cdot \frac{1}{3^j} \le \frac{4}{4+2(3j+1)} & \text{if } t = 3j + 1 \\ \frac{1}{2} \cdot \frac{1}{3^j} \le \frac{4}{4+2(3j+2)} & \text{if } t = 3j + 2 \\ \frac{1}{3} \cdot \frac{1}{3^j} < \frac{4}{4+2(3j+3)} & \text{if } t = 3j + 3 \end{cases}$$

Table 1 Capacity bound for different values of R

R	$d=7$	$d=6$	$d=5$	$d=4$
$\mathbf{6i+6}$	$(8+2t)f(3i-1-t)$ $\le 8\cdot f(3i-1)$ $=16\cdot 3^{i-1}$	$(6+2t)f(3i-t)$ $\le 6\cdot f(3i)$ $=18\cdot 3^{i-1}$	$(6+2t)f(3i-t)$ $\le 6\cdot f(3i)$ $=18\cdot 3^{i-1}$	$\mathbf{(4+2t)f(3i+1-t)}$ $\mathbf{\le 4\cdot f(3i+1)}$ $=16\cdot 3^{i-1}$
$\mathbf{6i+5}$	$(7+2t)f(3i-1-t)$ $\le 7\cdot f(3i-1)$ $=14\cdot 3^{i-1}$	$(7+2t)f(3i-1-t)$ $\le 7\cdot f(3i-1)$ $=14\cdot 3^{i-1}$	$(5+2t)f(3i-t)$ $\le 5\cdot f(3i)$ $=15\cdot 3^{i-1}$	$(5+2t)f(3i-t)$ $\le 5\cdot f(3i)$ $=15\cdot 3^{i-1}$
$\mathbf{6i+4}$	$(8+2t)f(3i-2-t)$ $\le 8\cdot f(3i-2)$ $=\frac{32}{3}\cdot 3^{i-1}$	$(6+2t)f(3i-1-t)$ $\le 6\cdot f(3i-1)$ $=12\cdot 3^{i-1}$	$(6+2t)f(3i-1-t)$ $\le 6\cdot f(3i-1)$ $=12\cdot 3^{i-1}$	$(4+2t)f(3i-t)$ $\le 4\cdot f(3i)$ $=12\cdot 3^{i-1}$
$\mathbf{6i+3}$	$(7+2t)f(3i-2-t)$ $\le 7\cdot f(3i-2)$ $=\frac{28}{3}\cdot 3^{i-1}$	$(7+2t)f(3i-2-t)$ $\le 7\cdot f(3i-2)$ $=\frac{28}{3}\cdot 3^{i-1}$	$(5+2t)f(3i-1-t)$ $\le 5\cdot f(3i-1)$ $=10\cdot 3^{i-1}$	$(5+2t)f(3i-1-t)$ $\le 5\cdot f(3i-1)$ $=10\cdot 3^{i-1}$
$\mathbf{6i+2}$	$(8+2t)f(3i-3-t)$ $\le 8\cdot f(3i-3)$ $=8\cdot 3^{i-1}$	$(6+2t)f(3i-2-t)$ $\le 6\cdot f(3i-2)$ $=8\cdot 3^{i-1}$	$(6+2t)f(3i-2-t)$ $\le 6\cdot f(3i-2)$ $=8\cdot 3^{i-1}$	$(4+2t)f(3i-1-t)$ $\le 4\cdot f(3i-1)$ $=8\cdot 3^{i-1}$
$\mathbf{6i+1}$	$(7+2t)f(3i-3-t)$ $\le 7\cdot f(3i-3)$ $=7\cdot 3^{i-1}$	$(7+2t)f(3i-3-t)$ $\le 7\cdot f(3i-3)$ $=7\cdot 3^{i-1}$	$\mathbf{(5+2t)f(3i-2-t)}$ $\mathbf{\le 5\cdot f(3i-2)}$ $=\frac{20}{3}\cdot 3^{i-1}$	$\mathbf{(5+2t)f(3i-2-t)}$ $\mathbf{\le 5\cdot f(3i-2)}$ $=\frac{20}{3}\cdot 3^{i-1}$

Hence we have $(s+2t)\cdot f(r-t)\le s\cdot f(r)$ when $s\ge 4$ and $t\ge 1$.

Case 3: $r=3i+1$.

$$\frac{f(3i+1-t)}{f(3i+1)}=\begin{cases}\frac{3}{4}\cdot\frac{1}{3^j}<\frac{4}{4+2(3j+1)} & \text{if } t=3j+1,\ t\neq 1\\ \frac{1}{2}\cdot\frac{1}{3^j}\le\frac{4}{4+2(3j+2)} & \text{if } t=3j+2\\ \frac{1}{3}\cdot\frac{1}{3^j}<\frac{4}{4+2(3j+3)} & \text{if } t=3j+3\end{cases}$$

In this case, when $s\ge 4$, $t=1$, the inequality $(s+2t)\cdot f(r-t)\le s\cdot f(r)$ fails. However, we still have $(s+2t)\cdot f(r-t)\le s\cdot f(r)$ when $s\ge 4$ and $t\ge 2$. For similar reasons, because

$$\frac{f(3i+1-t)}{f(3i+1)}=\begin{cases}\frac{3}{4}\cdot\frac{1}{3^j}\le\frac{6}{6+2(3j+1)} & \text{if } t=3j+1\\ \frac{1}{2}\cdot\frac{1}{3^j}<\frac{6}{6+2(3j+2)} & \text{if } t=3j+2\\ \frac{1}{3}\cdot\frac{1}{3^j}<\frac{6}{6+2(3j+3)} & \text{if } t=3j+3\end{cases}$$

we have $(s+2t)\cdot f(r-t)\le s\cdot f(r)$ when $s\ge 6$ and $t\ge 1$. □

Lemma 14 *When $R=6i+6$, the maximum capacity is $18\cdot 3^{i-1}$.*

Proof We only need to consider root degree $4 \le d \le 7$ according to the root degree bound derived in Sect. 4.2. In the following cases, we first compare the capacity of candidate trees with the same root degree. We try to prove that the tree at certain balance point ($r^+ - r^- \le 1$) has larger capacity in each case.

Case 1: Candidate trees with root degree $d = 7$.

All these candidate trees have $r^+ \ge r^-$ and $r^+ + r^- + d = 6i + 6$, thus we have $1 \le r^- \le 3i - 1$. Given such a candidate tree, when writing r^- as $3i - 1 - t$ where $0 \le t \le 3i - 1$, we have capacity bound $(R - 2(3i - 1 - t)) \cdot f(3i - 1 - t) = (8 + 2t) \cdot f(3i - 1 - t)$ by Theorem 8. We will prove that this capacity is maximum when $t = 0$. Hence, we only need to prove $(8 + 2t) \cdot f(3i - 1 - t) \le 8 f(3i - 1)$. Let $r = 3i - 1$, $s = 8$. Obviously for any $t \ge 1$, Fact 10 implies $(8+2t) \cdot f(3i-1-t) \le 8 f(3i-1) = 16 \cdot 3^{i-1}$.

Case 2: Candidate trees with root degree $d = 6$.

All these candidate trees have $r^+ \ge r^-$ and $r^+ + r^- + d = 6i + 6$, thus we have $1 \le r^- \le 3i$. Given such a candidate tree, when writing r^- as $3i - t$ where $0 \le t \le 3i$, we have capacity bound $(R - 2(3i - t)) \cdot f(3i - t) = (d + 2t) \cdot f(3i - t)$. We will prove that this capacity is maximum when $t = 0$. That is, we need to prove $(6+2t) \cdot f(3i-t) \le 6 \cdot f(3i)$ when $t \ge 1$. Let $r = 3i$, $s = 6$. Fact 10 implies $(6+2t) \cdot f(3i - t) \le 6 \cdot f(3i)$. Hence, the balance point $t = 0, r^- = 3i$, $r^+ = 3i$ achieves maximum capacity in this case.

Case 3: Candidate trees with root degree $d = 5$.

In this case, we have $1 \le r^- \le 3i$. We write r^- as $3i - t$. Similarly, let $r = 3i$, $s = 6$. Fact 10 implies $(6 + 2t) \cdot f(3i - t) \le 6 f(3i)$, which further implies that the capacity is maximized to be $18 \cdot 3^{i-1}$ when $t = 0$.

Case 4: Candidate trees with root degree $d = 4$.

In this case, we have $1 \le r^- \le 3i + 1$. Let $r = 3i + 1$, $s = 4$. When $t \ge 2$, Fact 10 implies $(4 + 2t) \cdot f(3i + 1 - t) \le 4 \cdot f(3i + 1)$. Thus the capacity of candidate tree which has $r^- = 3i + 1 - t$ and $t \ge 2$ is at most $4 f(3i + 1) = 16 \cdot 3^{i-1}$. While for $t = 1$, the inequality fails as $(4 + 2) \cdot f(3i) = 18 \cdot 3^{i-1} > 4 \cdot f(3i + 1) = 16 \cdot 3^{i-1}$. However, in this special case, the capacity $6 \cdot f(3i)$ is equal to that of case 2. Finally, note that when $t = 0$ the capacity bound $4 f(3i + 1) = 16 \cdot 3^{i-1}$ is less than that in case 2.

Since in all the four cases, the maximum capacity is $18 \cdot 3^{i-1}$, the lemma is proved. □

Considering other values of R, we finally arrive at our theorem.

Theorem 11 *For 2-replacement cost R, the maximum capacity is*

$$g(R) = \begin{cases} 6 \cdot 3^{i-1} & \text{if } R = 6i \\ 7 \cdot 3^{i-1} & \text{if } R = 6i + 1 \\ 8 \cdot 3^{i-1} & \text{if } R = 6i + 2 \\ 10 \cdot 3^{i-1} & \text{if } R = 6i + 3 \\ 12 \cdot 3^{i-1} & \text{if } R = 6i + 4 \\ 15 \cdot 3^{i-1} & \text{if } R = 6i + 5. \end{cases}$$

Proof Due to the similarity of the proofs for different values R, we only put the key inequality and capacity bound in Table 1 for $R = 6i+1, 6i+2, 6i+3, 6i+4, 6i+5$. Notice that in all cases, when $t \geq 2$ the inequality holds. In the special case that $t = 1$, when $R = 6i+6$, $d = 4$, $R = 6i+1$, $d = 5$ or $R = 6i+1$, $d = 4$, the inequality fails. The analysis of the case where $R = 6i+1$, $d = 5$ or $R = 6i+1$, $d = 4$ is similar to that of the case $R = 6i+6$, $d = 4$ as shown in the proof of Lemma 14. For example, when $R = 6i+1$, $d = 5$ and $t = 1$, the capacity bound $(6i+1-2(3i-2-t)) = 7f(3i-3)$ is the capacity that can be achieved in the case $R = 6i+1$, $d = 7$. □

After we have obtained the capacity for the 2-replacement cost R, we finally prove that among the candidate trees with n leaves, the optimal cost can be achieved by some balanced structure as shown below.

Definition 10 We use the balanced tree to denote a tree with root degree d where each subtree is 2-3 tree and has number of leaves differed by at most 1.

Theorem 12 *Among trees with n leaves*,

(1) *when* $n \in (15 \cdot 3^{i-1}, 18 \cdot 3^{i-1}]$, *the optimal tree is a balanced tree which has root degree* 6 *and* 2*-replacement cost* $6i+6$
(2) *when* $n \in (12 \cdot 3^{i-1}, 15 \cdot 3^{i-1}]$, *the optimal tree is a balanced tree which has root degree* 5 *and* 2*-replacement cost* $6i+5$
(3) *when* $n \in (10 \cdot 3^{i-1}, 12 \cdot 3^{i-1}]$, *the optimal tree is a balanced tree which has root degree* 6 *and* 2*-replacement cost* $6i+4$
(4) *when* $n \in (8 \cdot 3^{i-1}, 10 \cdot 3^{i-1}]$, *the optimal tree is a balanced tree which has root degree* 5 *and* 2*-replacement cost* $6i+3$
(5) *when* $n \in (7 \cdot 3^{i-1}, 8 \cdot 3^{i-1}]$, *the optimal tree is a balanced tree which has root degree* 6 *and* 2*-replacement cost* $6i+2$
(6) *when* $n \in (6 \cdot 3^{i-1}, 7 \cdot 3^{i-1}]$, *the optimal tree is a balanced tree which has root degree* 7 *and* 2*-replacement cost* $6i+1$.

Proof When $n \in (15 \cdot 3^{i-1}, 18 \cdot 3^{i-1}]$, we have $OPT_2(n) = 6i+6$. We will prove that the balanced tree with root degree 6 can always achieve this optimal cost. In the balanced tree, each subtree T_j $(1 \leq j \leq 6)$ has the number of leaves $n_j = \lceil \frac{n-j+1}{6} \rceil \in [\lfloor \frac{5}{2} \cdot 3^{i-1} \rfloor, 3 \cdot 3^{i-1}]$. By function $f(\cdot)$, we have $S_{T_i} \leq 3i$. Thus any two leaves from the tree will incur a replacement cost at most $2 \cdot 3i + 6 = 6i+6$.

When $n \in (12 \cdot 3^{i-1}, 15 \cdot 3^{i-1}]$ we have $OPT_2(n) = 6i+5$. Then we will prove that the balanced tree with root degree 5 can always achieve the optimal cost. In the balanced tree, each subtree T_j $(1 \leq j \leq 5)$ has the number of leaves $n_j = \lceil \frac{n-j+1}{5} \rceil \in [\lfloor \frac{12}{5} \cdot 3^{i-1} \rfloor, 3 \cdot 3^{i-1}]$. By function $f(\cdot)$, we have $S_{T_i} \leq 3i$. Thus any two leaves from the tree will incur a replacement cost at most $2 \cdot 3i + 5 = 6i+5$.

When $n \in (10 \cdot 3^{i-1}, 12 \cdot 3^{i-1}]$ we have $OPT_2(n) = 6i+4$ and $n_j = \lceil \frac{n-j+1}{6} \rceil \in [\lfloor \frac{5}{3} \cdot 3^{i-1} \rfloor, 2 \cdot 3^{i-1}]$. By function $f(\cdot)$, we have $S_{T_i} \leq 3i-1$. Thus any two leaves from the tree will incur a replacement cost at most $2 \cdot (3i-1) + 6 = 6i+4$.

When $n \in (8 \cdot 3^{i-1}, 10 \cdot 3^{i-1}]$, we have $OPT_2(n) = 6i+3$ and $n_j = \lceil \frac{n-j+1}{5} \rceil \in [\lfloor \frac{8}{5} \cdot 3^{i-1} \rfloor, 2 \cdot 3^{i-1}]$. By function $f(\cdot)$, we have $S_{T_i} \leq 3i-1$. Thus any two leaves from the tree will incur a replacement cost at most $2 \cdot (3i-1) + 5 = 6i+3$.

When $n \in (7 \cdot 3^{i-1}, 8 \cdot 3^{i-1}]$, we have $OPT_2(n) = 6i + 2$ and $n_j = \lceil \frac{n-j+1}{6} \rceil \in [\lfloor \frac{7}{6} \cdot 3^{i-1} \rfloor, \frac{4}{3} \cdot 3^{i-1}]$. By function $f(\cdot)$, we have $S_{T_i} \leq 3i - 2$. Thus any two leaves from the tree will incur a replacement cost at most $2 \cdot (3i - 2) + 6 = 6i + 2$.

When $n \in (6 \cdot 3^{i-1}, 7 \cdot 3^{i-1}]$, we have $OPT_2(n) = 6i + 1$. The balanced tree with root degree 7 where $n_j = \lceil \frac{n-j+1}{7} \rceil \in [\lfloor \frac{6}{7} \cdot 3^{i-1} \rfloor, 3^{i-1}]$ can always achieve the optimal cost. By function $f(\cdot)$, we have $S_{T_i} \leq 3i - 3$. Thus any two leaves from the tree will incur a replacement cost at most $2 \cdot (3i - 3) + 7 = 6i + 1$. □

Finally we have fixed the structure of the dominating branch and obtained the optimal tree structure for the 2-replacement problem. We conjecture the general result for the k-replacement problem in the next section.

5 Conclusion

In this paper, we study the optimal structure for the key tree problem. The k-deletion problem is proved to be equivalent with k-replacement problem. We show that the optimal tree for the 2-replacement problem is a balanced tree which can be computed in $O(n)$ time.

The capacity $f(\cdot)$ where $k = 1$ and $g(\cdot)$ where $k = 2$ stimulates us to conjecture the general form of capacity $G_k(R)$ which denotes the maximum number of leaves that can be placed in a tree given the k-replacement cost R in the k-replacement problem. Based on the form of $f(\cdot)$ and $g(\cdot)$, we conjecture the capacity $G_k(R)$ to be of the form shown in (1). We believe such a capacity function is generated by some balanced structure of the trees. Furthermore, if the conjecture is proved to be correct, it is also possible to obtain the optimal structure in a similar way as in the proof of Theorem 12 and construct the optimal tree in $O(n)$ time.

$$G_k(R) = \begin{cases} (3k + \alpha) \cdot 3^{i-1} & \text{if } R = 3k \cdot i + \alpha,\ \alpha \in [0, k) \\ (4k + 2(\alpha - k)) \cdot 3^{i-1} & \text{if } R = 3k \cdot i + \alpha,\ \alpha \in [k, 2k) \\ (6k + 3(\alpha - 2k)) \cdot 3^{i-1} & \text{if } R = 3k \cdot i + \alpha,\ \alpha \in [2k, 3k) \end{cases} \quad (1)$$

One of the possible future work is hence to investigate the capacity for the general k-replacement problem which indicates a linear time algorithm to compute the optimal structure. The extension clearly needs some new ideas. We believe that the concept of capacity will also be very important to this problem.

Appendix: Proof of Theorem 7

Lemma 15 *Given a tree T with root degree 2 where two subtrees T_1, T_2 are both 2-3 trees and $|n_1 - n_2| \leq 1$, when we transform T into 2-3 tree T' with root degree 3 as shown in Fig. 10, we have $S_T \geq S_{T'}$, $D_T \geq D_{T'}$.*

Proof Without loss of generality, we assume that $n_1 \geq n_2$. According to the definition and optimality of 2-3 tree, when the 1-replacement cost is r, the maximum possible

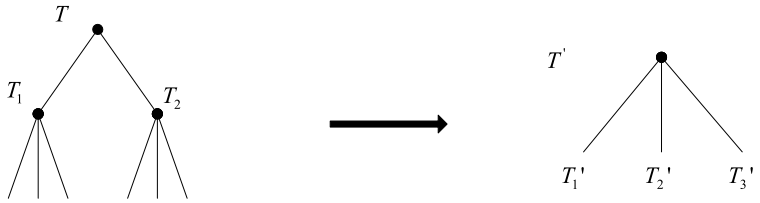

Fig. 10 Transform a tree T where the two subtrees are 2-3 trees into a 2-3 tree T'

number of leaves on the tree is $f(\cdot)$ where

$$f(r) = \begin{cases} 3 \cdot 3^{i-1} & \text{if } r = 3i \\ 4 \cdot 3^{i-1} & \text{if } r = 3i+1 \\ 6 \cdot 3^{i-1} & \text{if } r = 3i+2 \end{cases}$$

Notice that T_1 and T_2 are 2-3 trees which satisfy function $f(\cdot)$. When $n_1 \in (3 \cdot 3^{i-1}, 4 \cdot 3^{i-1}]$, we have $S_{T_2} \le S_{T_1} = 3i+1$ and $S_T = S_{T_1} + 2 = 3i+3$. Since $n_1 + n_2 \le 8 \cdot 3^{i-1} < 3 \cdot 3^i$, we have $S_{T'} \le 3i+3 = S_T$. Similarly for $n_1 \in (4 \cdot 3^{i-1}, 6 \cdot 3^{i-1}]$, we have $S_T = S_{T_1} + 2 = 3i+4$ and $S_{T'} \le 3i+4$. And for $n_1 \in (6 \cdot 3^{i-1}, 9 \cdot 3^{i-1}]$, we have $S_T = S_{T_1} + 2 = 3i+5$ and $S_{T'} \le 3i+5$. Therefore the first inequality of the lemma holds.

Suppose that the three subtrees of T' are T'_1, T'_2, T'_3 which are also 2-3 trees and satisfy $S_{T'_1} \ge S_{T'_2} \ge S_{T'_3}$. Since $S_T = S_{T_1} + 2$ and $S_{T'} = S_{T'_1} + 3$, we have $S_{T'_1} \le S_{T_1} - 1$ and $D_{T'} \le 2S_{T'_1} + 3 \le 2S_{T_1} + 1$. Note that in tree T, we have $S_{T_2} \ge S_{T_1} - 1$ because both subtrees are 2-3 trees with the number of leaves differed by at most 1. Hence, we have $D_{T'} \le S_{T_1} + S_{T_2} + 2 \le D_T$. The lemma is finally proved. □

Lemma 16 *For the* 2*-replacement problem, a tree* T *with root degree* 2 *can be transformed into a tree with root degree* 3 *or* 4 *without increasing the* 2*-replacement cost.*

Proof Lemma 7 implies that we can transform T into a candidate-tree where T_2 is a 2-3 tree. We divide our discussion into three cases.

Case 1: $D_{T_1} < S_{T_1} + S_{T_2}$.

According to Lemma 8, in subtree T_1, we have $D_{T_1} = S_{T_1} + c(u_2) - c(v) - d_v < S_{T_1} + S_{T_2}$ which implies $c(u_2) - c(v) - d_v \le S_{T_2} - 1$ where v is the nearest common ancestor of u_1, u_2. We discuss in two subcases $c(u_2) - c(v) - d_v \le S_{T_2} - 2$ and $c(u_2) - c(v) - d_v = S_{T_2} - 1$.

When $c(u_2) - c(v) - d_v \le S_{T_2} - 2$, we transform T into a tree T' which is composed of the dominating branch B_1 and T'_2 (T'_2 is the same as T_2) as shown in Fig. 11. In T', we consider three subcases according to the positions of the two leaves whose replacement cost reaches $D_{T'}$. For the subcase where two leaves are both from T'_2, we have $D_{T'} = D_{T'_2} + 1 = D_{T_2} + 1 < D_{T_2} + 2 \le D_T$. For the subcase where one of the leaves is from T'_1 and the other is from T'_2, we have $D_{T'} = S_{T'_1} + S_{T'_2} + 1 = S_{T_1} + S_{T_2} + 1 < S_{T_1} + S_{T_2} + 2 \le D_T$. For the subcase where both leaves u_1 and u_2 are from T'_1, we have $D_{T'} = D_{T'_1} + 4 = D_{T_1} + 4$. Since $c(u_2) - c(v) - d_v \le S_{T_2} - 2$, we have $D_{T_1} \le S_{T_1} + S_{T_2} - 2$, which implies $D_{T'} \le S_{T_1} + S_{T_2} + 2 \le D_T$.

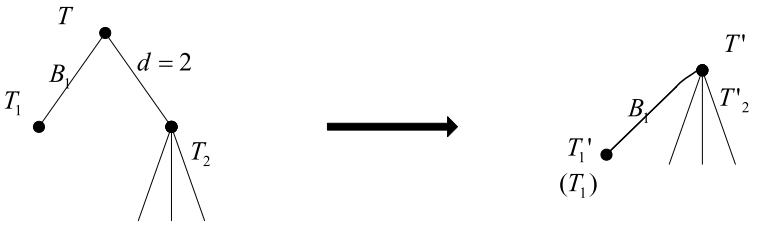

Fig. 11 Transform a tree T from root degree 2 into degree 4

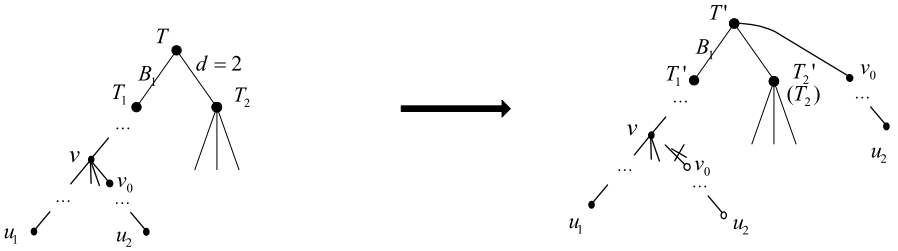

Fig. 12 Transform a tree T from root degree 2 to degree 3

When $c(u_2) - c(v) - d_v = S_{T_2} - 1$, we will consider another transformation for T. Notice that $S_{T_1} = c(u_1) - 2 \geq c(u_2) - c(v) = S_{T_2} - 1 + d_v \geq S_{T_2} + 1$, which means $S_{T_2} \leq S_{T_1} - 1$. Assume that T_{v_0} is the subtree of v which contains leaf u_2, then we can move T_{v_0} to the root to produce a tree T', as Fig. 12 shows. We also consider three subcases for T' according to the positions of the two leaves whose replacement cost reaches $D_{T'}$. Firstly, if the two leaves are originally from T_1, then we have $D_{T'} \leq D_{T_1} + 3 \leq S_{T_1} + S_{T_2} + 2 = D_T$. Secondly, if one of the two leaves is originally from T_1 while the other is not, then we will prove that replacing a leaf from T_1' or T_{v_0} incurs cost at most $S_{T_1} - 1$ which implies $D_{T'} \leq (S_{T_1} - 1) + S_{T_2} + 3 = D_T$. Suppose first the leaf u is from subtree $T_v \backslash T_{v_0}$ which is rooted at v. Since d_v decreases by 1 in the resulting tree, the cost to delete u in T_1' is at most $S_{T_1} - 1$. If the leaf u is from T_{v_0}, then the cost to delete u in T_{v_0} is at most $S_{T_1} - d_v$. Otherwise if the leaf is not from T_v, then we suppose the leaf u is a leaf descendant of $\hat{v}$ where $\hat{v}$ is an ancestor of v, as shown in Fig. 13. Note that $c(u_1) + c(u) - c(\hat{v}) - d_{\hat{v}} \leq c(u_1) + c(u_2) - c(v) - d_v = c(u_1) + S_{T_2} - 1$, thus we have $c(u) \leq S_{T_2} - 1 + c(\hat{v}) + d_{\hat{v}} \leq S_{T_2} - 1 + c(v) = c(u_2) - d_v \leq c(u_1) - d_v = S_{T_1} + 2 - d_v \leq S_{T_1}$, which implies $c(u) \leq S_{T_1}$. Therefore, replacing u from T_1' is at most $S_{T_1} - 2$. Finally, if both leaves are originally from T_2, then we have $D_{T'} < 2S_{T_2} + d + 1 \leq S_{T_1} - 1 + S_{T_2} + d + 1 = S_{T_1} + S_{T_2} + d \leq D_T$.

Hence, when $D_{T_1} < S_{T_1} + S_{T_2}$, we can transform T into a better tree with root degree 3 or 4.

Case 2: $D_{T_1} = S_{T_1} + S_{T_2}$.

Suppose the root degree of T_1 is d_1. We move a branch B_{1d_1} of T_1 to the root (Fig. 14) to obtain a new tree T'. We will prove that 2-replacement cost of T' does not change compared to that of T. We also consider three subcases for T' according to the positions of the two leaves whose replacement cost reaches $D_{T'}$. If the two

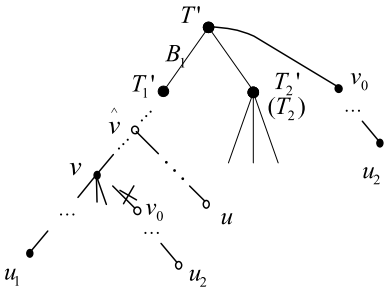

Fig. 13 u belongs to a leaf descendant of $\hat{v}$

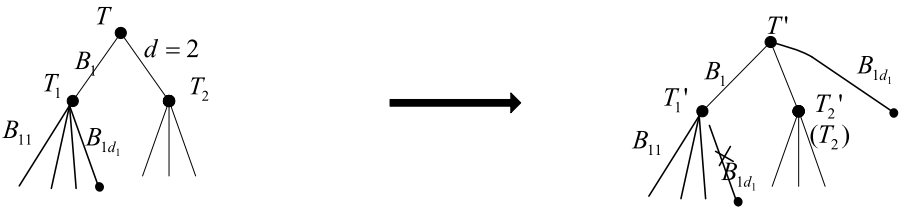

Fig. 14 Transform a tree T from root degree 2 to degree 3

leaves are originally from T_1, we have $D_{T'} \le D_{T_1'} + d + 1 = D_{T_1} - 1 + d + 1 = D_{T_1} + d \le D_T$. If one of the leaves is originally from T_1 while the other is not, the cost $D_{T'}$ is $S_{T_1'} + S_{T_2} + d + 1 = S_{T_1} - 1 + S_{T_2} + d + 1 = S_{T_1} + S_{T_2} + d \le D_T$ or $S_{T_2'} + S_{T_{1d_1}} + d + 1 = S_{T_2} + S_{T_{1d_1}} + d + 1 < S_{T_2} + S_{T_1} + d \le D_T$.

If both leaves are originally from T_2, the cost $D_{T'}$ is no more than $2S_{T_2} + d + 1$. Furthermore, by Lemma 8, $D_{T_1} = S_{T_1} + c(u_2) - c(v) - d_v \le 2S_{T_1} - 1$. Since $D_{T_1} = S_{T_1} + S_{T_2}$, we have $S_{T_2} \le S_{T_1} - 1$. Therefore, $D_{T'} \le 2S_{T_2} + d + 1 \le S_{T_1} - 1 + S_{T_2} + d + 1 = S_{T_1} + S_{T_2} + d \le D_T$.

Hence, we have $D_{T'} \le D_T$.

Case 3: $D_{T_1} > S_{T_1} + S_{T_2}$.

In this case, replacing two leaves from T_1 incurs the cost D_T. We first transform T into a tree satisfying case 1 or case 2. In subtree T_1, by Lemma 8, if $c(u_1, u_2) = D_T$, we can assume $c(u_1) = S_T$. Suppose v is the nearest common ancestor of u_1 and u_2, and v's children v_1 and v_2 are ancestors of u_1 and u_2 respectively (v_i can be u_i itself), then $c(u_1) - c(v_1) \ge c(u_2) - c(v_2)$. We can exchange subtree T_{v_2} which is rooted at v_2 with subtree T_2, as shown in Fig. 15. Because we have $S_{T_1} + S_{T_2} < D_{T_1} = S_{T_1} + c(u_2) - c(v_2) = S_{T_1} + S_{T_{v_2}}$, we know that T_{v_2} has larger 1-replacement cost than T_2. After the exchange, we have $D_{T'} \le D_T$ because replacing one leaf from T_{v_1} and one leaf from T_{v_2} incurs a cost at most $S_{T_1'} + c(u_2) - c(v_2) + d = S_{T_1} + c(u_2) - c(v_2) + d = D_T$, replacing two leaves from T_2' of T' incurs a cost at most $2S_{T_2} + c(v) + d_v < S_{T_2} + S_{T_{v_2}} + c(v) + d_v \le S_{T_1} + S_{T_2} + d < D_T$ and other combinations of two replaced leaves in T' incur at most the same cost as that of T correspondingly. Note that T' has degree 2 and belongs to case 1 or case 2. Hence, T' can be further transformed into a tree with degree 3 or 4 according to case 1 and case 2.

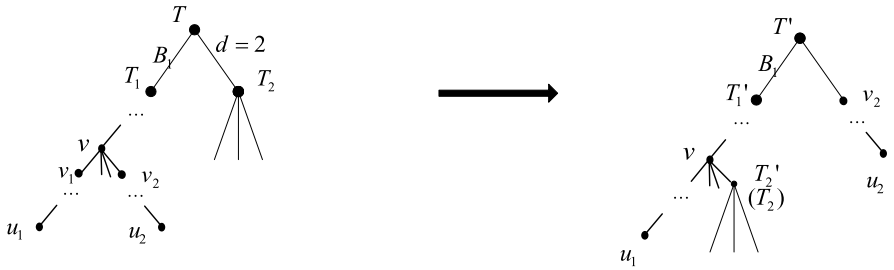

Fig. 15 Exchange two subtrees for the case that $D_{T_1} > S_{T_1} + S_{T_2}$

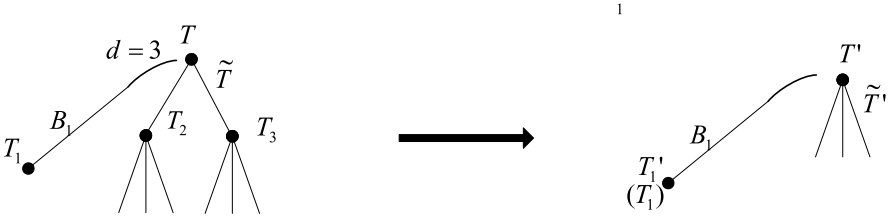

Fig. 16 Transform a tree T from root degree 3 to root degree 4

Since in all cases, we transform a tree with root degree 2 into a tree with root degree 3 or 4 without increasing 2-replacement cost, the lemma is finally proved. □

Lemma 17 *For* 2*-replacement problem, a tree* T *with root degree* 3 *can be transformed into a tree with root degree* 4 *without increasing the* 2*-replacement cost.*

Proof Case 1: $D_{T_1} < S_{T_1} + S_{T_2}$.

Lemma 7 implies that we can transform T into a candidate-tree where T_2, T_3 are 2-3 trees without increasing 2-replacement cost. We then reallocate the leaves in T_2, T_3 so that T_2 is a template tree of T_3 with $0 \le n_2 - n_3 \le 1$ and both subtrees are still 2-3 trees as Fig. 16 shows. First, it is easy to see that this transformation will not increase 2-replacement cost. We then prove that such a tree can be further transformed into a tree with root degree 4 without increasing 2-replacement cost. Consider T's two branches B_2 and B_3 (Suppose these two branches compose a new tree $\tilde{T}$) in T, according to the proof of Lemma 15, we can transform $\tilde{T}$ into tree $\tilde{T}'$ which is a 2-3 tree and we denote the whole new tree as T'. The transformation ensures that $S_{\tilde{T}} \ge S_{\tilde{T}'}$, $D_{\tilde{T}} \ge D_{\tilde{T}'}$. We can prove that the transformation does not increase the 2-replacement cost. We also consider three subcases for T' according to the positions of the two leaves whose replacement cost reaches $D_{T'}$. If two leaves are originally from T_1, we have $D_{T'} = D_{T_1'} + d + 1 = D_{T_1} + d + 1 \le S_{T_1} + S_{T_2} + d = D_T$. If one of the leaves is originally from T_1 while the other is from $\tilde{T}$, we have $D_{T'} = S_{T_1} + S_{\tilde{T}'} + 1 \le S_{T_1} + S_{\tilde{T}} + 1 = S_{T_1} + S_{T_2} + d = D_T$. If both leaves are from $\tilde{T}$, we have $D_{T'} = D_{\tilde{T}'} + 1 \le D_{\tilde{T}} + 1 \le D_T$. In all the subcases, we have shown that $D_{T'} \le D_T$.

Case 2: $D_{T_1} = S_{T_1} + S_{T_2}$.

The transformation and proof for this case is similar to the case 2 in the proof of Lemma 16, the only difference is that the original tree has root degree $d = 3$ instead of 2 in this case.

Case 3: $D_{T_1} > S_{T_1} + S_{T_2}$.

The transformation and proof for this case is similar to the case 3 in the proof of Lemma 16, the only difference is that the original tree has root degree $d = 3$ instead of 2 in this case.

Since in all cases, we transform a tree from root degree 3 into root degree 4 without increasing 2-replacement cost, the lemma is finally proved. □

References

Chan A, Rajaraman R, Sun Z, Zhu F (2009) Approximation algorithms for key management in secure multicast. In: Proceedings of the 15th annual international conference on computing and combinatorics (COCOON), pp 148–157

Chan Y-K, Li M, Wu W (2010) Optimal tree structure with loyal users and batch updates. J Combin Optim 1–10

Chen ZZ, Feng Z, Li M, Yao FF (2008) Optimizing deletion cost for secure multicast key management. Theor Comput Sci 401:52–61

Goodrich MT, Sun JZ, Tamassia R (2004) Efficient tree-based revocation in groups of low-state devices. In: Proceedings of the twenty-fourth annual international cryptology conference (CRYPTO), pp 511–527

Graham RL, Li M, Yao FF (2007) Optimal tree structures for group key management with batch updates. SIAM J Discrete Math 21(2):532–547

Li XS, Yang YR, Gouda MG, Lam SS (2001) Batch re-keying for secure group communications. In: Proceedings of the tenth international conference on world wide web, pp 525–534

Snoeyink J, Suri S, Varghese G (2001) A lower bound for multicast key distribution. In: Proceedings of the twentieth annual ieee conference on computer communications, pp 422–431

Wallner D, Harder E, Agee RC (1999) Key management for multicast: issues and architectures, RFC 2627, June

Wong CK, Gouda MG, Lam SS (2000) Secure group communications using key graphs. IEEE/ACM Trans Netw 8(1):16–30

Wu W, Li M, Chen E (2008a) Optimal tree structures for group key tree management considering insertion and deletion cost. In: 14th annual international computing and combinatorics conference

Wu W, Li M, Chen E (2008b) Optimal key tree structure for deleting two or more leaves. In: ISAAC08

Zhu F, Chan A, Noubir G (2003) Optimal tree structure for key management of simultaneous join/leave in secure multicast. In: Proceedings of military communications conference, pp 773–778