A Dynamic Programming Algorithm for Tree Trimming-based Text Summarization

Masaaki Nishino[1], **Norihito Yasuda**[2], **Tsutomu Hirao**[1], **Shin-ichi Minato**[2,3], **Masaaki Nagata**[1]
[1]NTT Communication Science Laboratories, NTT Corporation
[2]ERATO Minato Discrete Structure Manipulation System Project, JST
[3]Graduate School of Information Science and Technology, Hokkaido University
nishino.masaaki@lab.ntt.co.jp

Abstract

Tree trimming is the problem of extracting an optimal subtree from an input tree, and sentence extraction and sentence compression methods can be formulated and solved as tree trimming problems. Previous approaches require integer linear programming (ILP) solvers to obtain exact solutions. The problem of this approach is that ILP solvers are black-boxes and have no theoretical guarantee as to their computation complexity. We propose a dynamic programming (DP) algorithm for tree trimming problems whose running time is $O(NL \log N)$, where N is the number of tree nodes and L is the length limit. Our algorithm exploits the zero-suppressed binary decision diagram (ZDD), a data structure that represents a family of sets as a directed acyclic graph, to represent the set of subtrees in a compact form; the structure of ZDD permits the application of DP to obtain exact solutions, and our algorithm is applicable to different tree trimming problems. Moreover, experiments show that our algorithm is faster than state-of-the-art ILP solvers, and that it scales well to handle large summarization problems.

1 Introduction

Extractive text summarization and sentence compression are tasks that basically select a subset of the input set of textual units that is appropriate as a summary or a compressed sentence. Current text summarization and sentence compression methods regard the problem of extracting such a subset as a combinatorial optimization problem (e.g., (Filatova and Hatzivassiloglou, 2004; McDonald, 2007; Lin and Bilmes, 2010)). Tree trimming, the problem of finding an optimal subtree of an input tree, is one kind of these combinatorial optimization problems, and it is used in three classes of text summarizations: sentence compression (Filippova and Strube, 2008; Filippova and Altun, 2013), single-document summarization (Hirao et al., 2013), and the combination of sentence compression and single-document summarization (Kikuchi et al., 2014). In these tasks, the set of input textual units is represented as a rooted tree whose nodes correspond to the minimum textual units such as sentences and words. Next, a subset is made by forming a subtree by trimming the input tree. Since the optimal trimmed subtree preserves the relationships between textual units, it is a concise representation of the original set that preserves linguistic quality.

A shortcoming of tree trimming-based methods is that they are formulated as integer linear programming (ILP) problems and so an ILP solver is needed to solve them. Although modern ILP solvers can solve many instances of tree trimming problems in a short time, there is no theoretical guarantee that they obtain an optimal solution. Furthermore, even if an optimal solution can be obtained, we cannot estimate the running time. Estimating the running time is critical for practical applications.

In this paper, we propose a dynamic programming (DP) algorithm for tree trimming problems that focus on text summarization. The algorithm can solve all three different classes of tree trimming problems proposed so far in a unified way, and it can always find an optimal solution in $O(NL \log N)$ time for these problems, where N is the number of nodes of the input tree and L is the length limit. The running time of our algorithm only depends on N and L and so is independent of the input trees structure. Finding an exact solution is important since we can use it to evaluate the performance of heuristic algorithms.

The key idea of our algorithm is to use the zero-suppressed binary decision diagram (ZDD) (Minato, 1993) to represent the set of all subtrees of the input tree. ZDD is a data structure that represents a family of sets as a directed acyclic graph (DAG). It can represent a family of sets in compressed form. We use ZDD to represent the set of subtrees of the input tree, and then run a DP algorithm on the ZDD to obtain the optimal solution that satisfies the length limit. The algorithm runs in time $O(|Z|L)$, where $|Z|$ is the number of nodes of ZDD, and L is the length limit. Although the number of ZDD nodes depends on the set we want to represent, we can give theoretical upper bounds when we represent the set of all subtrees of an input tree. ZDD uses $O(N \log N)$ nodes to represent the set of all subtrees of an N node input tree. Hence the DP algorithm runs in $O(NL \log N)$ time. The main virtues of the proposed algorithm are that (1) it can always find an exact solution, (2) its running time is theoretically guaranteed, and (3) it can solve the three known tree trimming problems. Furthermore, our algorithm is fast enough to be practical and scalable. Since text summarization methods are often applied to large scale inputs (e.g., (Christensen et al., 2014; Nakao, 2000)), scalability is important. We compare it to state-of-the-art ILP solvers and confirm that the proposed algorithm can be hundreds of times faster.

Since our method assumes known formuations for text summarization, the summary created by our algorithm is exactly the same as that obtained by applying previous methods. However, we believe that algorithmic improvements in computational cost is as important as improvements in accuracy in order to make better practical systems.

2 Tree Trimming Problems

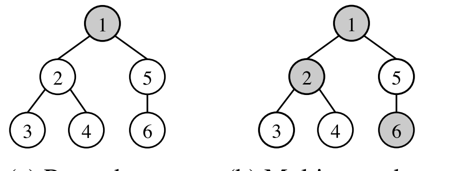
We briefly review the three tree trimming formulations used in text summarization and sentence compression. They all try to find the subtree that maximizes the sum of item weights while satisfying the length limit. Let $D = \{e_1, \dots, e_N\}$ be the input set of textual units, where e_i represents the i-th unit. We use w_i and l_i to represent the weight and length of e_i, respectively. Given length limit L, these methods solve the following optimization problem:

$$\begin{aligned} &\underset{T \subseteq D}{\text{Maximize}} \sum_{e_i \in T} w_i \\ &\text{Subject to } T \in \mathcal{T} \text{ and } \sum_{e_i \in T} l_i \leq L\,, \end{aligned} \tag{1}$$

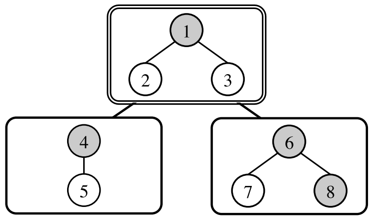
where $T \subseteq D$ and $\mathcal{T} \subseteq 2^D$. We use $\mathcal{T}$ to represent the set of subtrees that can be feasible solutions if we ignore the length limit. The following problems employ different $\mathcal{T}$ to match each problem setting. If $\mathcal{T} = 2^D$, i.e., $\mathcal{T}$ equals the set of all possible subsets of D, it is equivalent to the 0-1 knapsack problem, and is solved with the standard DP algorithm.

Sentence Extraction Hirao et al. (2013) proposed a single-document summarization algorithm to solve a tree trimming problem. They represent a document as a set of elementary discourse units (EDUs) and then select an optimal subset to make a summary. Each EDU is a minimal unit that composes the discourse structure of the document; it usually corresponds to a clause. Their summarization method first represents a document as a dependency discourse tree (DEP-DT) that represents the dependency structure between EDUs. DEP-DT is a rooted tree in which each node corresponds to an EDU. They then select the rooted subtree that maximizes the sum of weights and satisfies the length limit to make a summary, where we say a subtree is rooted if it contains the root node of the input tree. This problem can be formulated as the combinatorial optimization problem of Eq.(1), where $\mathcal{T}$ is the set of all rooted subtrees of the input DEP-DT.

Sentence Compression Filippova and Strube (2008) proposed a sentence compression method based on the trimming of a word dependency tree. Its recently proposed variant shows state-of-the-art performance (Filippova and Altun, 2013). They trim a syntactical dependency tree to compress a sentence. Their formulation is similar to the previous sentence extraction method except that it allows the root node of a subtree to be other than the root node of the input tree. In other words, their formulation allows multiple candidate root nodes for a subtree. We represent such a set of candidate root nodes as R, and the set of possible solutions $\mathcal{T}$ for this formulation



(a) Rooted tree (b) Multi-rooted tree



(c) Nested tree

Figure 1: Example trees.

is the set of all subtrees of the input tree whose root node is contained in R.

Sentence Extraction & Compression Kikuchi et al. (2014) proposed a single-document summarization method that can select compressed sentences. It is an extension of the sentence extraction method proposed in (Hirao et al., 2013). They represent a document as a sentence dependency tree that is obtained from DEP-DT, and then represent each sentence in the sentence dependency tree as a word dependency tree. In the following, inner trees refer to the word dependency trees that correspond to sentences, while the outer tree represents the sentence dependency tree that represents a document. Hence a document is represented as a nested tree where each node of the outer tree corresponds to an inner tree. They then make a summary by first selecting a rooted subtree of the outer tree, and then selecting a subtree for each inner tree that corresponds to a node of the selected subtree of the outer tree. Each inner tree has multiple root candidate nodes, and the root node of a subtree of an inner tree is a root candidate node of the tree. The set of feasible solutions, $\mathcal{T}$, corresponds to all possible nested trees constructed in this way[1].

Fig. 1 shows example input trees used in the above three tasks: (a) a rooted tree used in sentence extraction, (b) a multi-rooted tree used in sentence compression, and (c) a nested tree used in sentence extraction & compression. Gray nodes are root candidate nodes. Each tree yields a different set of valid subtrees. Tab. 1 shows examples of valid and invalid subtrees of each input tree, where we assume that each subtree in $\mathcal{T}$ is represented by a set of nodes that is contained in the subtree.

[1]Kikuchi et al. (2014) set further constraints on possible subtrees of a syntactical tree. Our method can also cope with these additional constraints (see Sect. 7).

Table 1: Examples of valid and invalid subtrees of the input trees in Fig. 1

	Valid	Invalid
Rooted tree	$e_1e_2e_3$, $e_1e_2e_5$	$e_2e_3e_4$, e_6
Multi-rooted tree	$e_1e_2e_5$, $e_2e_3e_4$	e_3, e_5e_6
Nested tree	$e_1e_2e_4$, $e_1e_4e_5e_8$	$e_4e_5e_6$, $e_1e_2e_7$

3 Zero-suppressed Binary Decision Diagram (ZDD)

The key idea of the proposed algorithm is to represent the set of candidate subtrees $\mathcal{T}$ as a zero-suppressed binary decision diagram (ZDD) (Minato, 1993). ZDD is a variant of binary decision diagram (BDD) (Bryant, 1986; Akers, 1978), and is a data structure that can succinctly represent a family of sets as a DAG. ZDD has two types of nodes, namely branch nodes and terminal nodes. Branch nodes are non-terminal nodes. Each branch node has exactly two out edges, called low-edge and high-edge, and a label that represents the item that the node corresponds to. We use $\mathrm{hi}(i)$, $\mathrm{lo}(i)$, and $\mathrm{v}(i)$ to represent the node pointed to by the high-edge, low-edge, and the label of the i-th node of the ZDD, respectively. The branch node that has no parent node is the root node. Terminal nodes have no outgoing edges, and a ZDD has exactly two terminal nodes whose labels are $\top$ and $\bot$. A path from the root node to terminal node $\top$ represents a set of items contained in the family of sets represented by the ZDD. We can recover the set of items that corresponds to a path by selecting the labels of the branch nodes whose high-edges lie on the path.

Fig. 2(a) is a ZDD that represents the family of sets $\{e_1e_2, e_2e_3, e_1e_3\}$. We use circles to represent branch nodes and rectangles to represent the terminal nodes. A dashed edge represents a low-edge and full edge represents a high-edge. The number on each circle node represents the label of the node. For example, the label of the root node of the ZDD

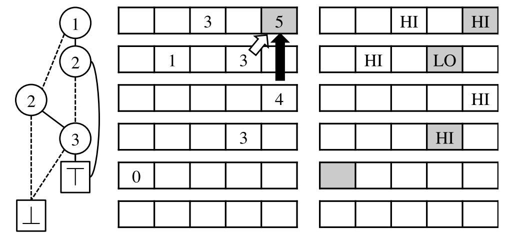


Figure 2: An example ZDD and how the dynamic programming algorithm works with the ZDD. (a) An example ZDD that represents the family of sets $\{e_1e_2, e_1e_3, e_2e_3\}$, (b) table S and (c) table B after completion of the table filling phase.

in Fig. 2(a) is 1. The ZDD has three different paths that start at the root node and end at $\top$. Each path corresponds to an item contained in the family of sets.

In the following, let $z_1, \ldots, z_{|Z|}$ be the nodes of a ZDD. We use Z to represent a ZDD, and $|Z|$ to represent the number of nodes in Z. We assume $i < \text{hi}(i), \text{lo}(i)$ for every $i = 1, \ldots, |Z| - 2$. z_1 corresponds to the root node, and $z_{|Z|-1}, z_{|Z|}$ corresponds to $\top$ and $\perp$ terminal nodes, respectively. We also assume that the ZDD is ordered, i.e., there is a total order on the labels, and the label of a parent node comes before that of a child node for every parent-child node pair. The ZDD in Fig. 2(a) is an ordered ZDD whose order is e_1, e_2, e_3.

4 Dynamic Programming Algorithm for Tree Trimming Problems

Our algorithm takes the following three-step procedure. First, we represent the set of subtrees $\mathcal{T}$ for each tree trimming problem as a ZDD. Then we apply a bottom-up and table-filling style DP algorithm to the ZDD. Finally, we backtrack the filled table to obtain an optimal solution.

Our algorithm is similar to the standard DP algorithm for the 0-1 knapsack problem, which solves the problem in $O(NL)$ time with N items and length limit L. The DP algorithm solves a knapsack problem by filling an $N \times (L+1)$ table by recursively exploiting previously computed partial solutions. Our algorithm also fills a table for problem solving, but the table's size is $|Z| \times (L + 1)$. That is, the size of the table equals the number of nodes of the ZDD that represents a set of subtrees $\mathcal{T}$. The tables can be seen as the set of $|Z|$ arrays with $(L+1)$ entries, and each array is associated with each ZDD node. We fill these tables by referring to previously computed results by using the ZDD's structure.

Algorithm 1 Dynamic Programming Algorithm

```
Input: ZDD $Z$ that represent $\mathcal{T}$, length limit $L$, and $w_i$, $l_i$ for
    $1 \leq i \leq N$
Output: Optimal subtree $\mathbf{r}$
 1: Initialize $S[i][j] \leftarrow -\infty$ for all $i, j$.
 2: $S[|Z| - 1][0] \leftarrow 0$.
 3: for $i = |Z| - 2, \ldots, 1$ do
 4:     for $j = 0, \ldots, L$ do
 5:         if $j \geq l_{\text{v}(i)}$ and
            $S[\text{hi}(i)][j - l_{\text{v}(i)}] + w_{\text{v}(i)} > S[\text{lo}(i)][j]$ then
 6:             $S[i][j] \leftarrow S[\text{hi}(i)][j - l_{\text{v}(i)}] + w_{\text{v}(i)}$
 7:             $B[i][j] \leftarrow HI$
 8:         else
 9:             $S[i][j] \leftarrow S[\text{lo}(i)][j]$,   $B[i][j] \leftarrow LO$
10: $k^* \leftarrow \text{argmax}_{0 \leq k \leq L} S[1][k]$
11: $i \leftarrow 1$,   $j \leftarrow k^*$,   $\mathbf{r} \leftarrow \emptyset$
12: while $(i, j) \neq (|Z| - 1, 0)$ do
13:     if $B[i][j] = HI$ then
14:         $\mathbf{r} \leftarrow \mathbf{r} \cup \{\text{v}(i)\}$,   $i \leftarrow \text{hi}(i)$,   $j \leftarrow j - l_{\text{v}(i)}$
15:     else
16:         $i \leftarrow \text{lo}(i)$
17: return $\mathbf{r}$
```

Alg. 1 is the DP algorithm that can solve the problem of Eq.(1), given the ZDD that represents the family of sets $\mathcal{T}$. We first prepare two tables, S and B; both have $|Z| \times (L + 1)$ entries. Table S is used for storing intermediate weights, and B is used for storing information used in recovering the optimal solution. We first fill the elements in S and B while traversing the ZDD in order from the terminal nodes to the root node. We then use B to recover the solution that maximizes the weight. In the table filling phase (lines 1 to 9), we update $S[i][j]$ and $B[i][j]$, recursively. Weight $S[i][j]$ represents the maximum weight of the ZDD path from the i-th node to the $\top$ terminal node, whose total length is j. We compare $S[\text{hi}(i)][j - l_{\text{v}(i)}] + w_{\text{v}(i)}$ and $S[\text{lo}(i)][j]$, and select the maximum weight to set $S[i][j]$. The value of $B[i][j]$ stores which candidate we set to $S[i][j]$. If we use the former one, we set label HI to $B[i][j]$, otherwise LO. After filling the table, we run a backtracking procedure to obtain an optimal solution. In the backtracking phase (lines 10 to 16), we start from $B[i][k^*]$ and repeat backtracking using the entries of B.

We give here a proof of the correctness of the algorithm. We use the fact that the ZDD is constructed recursively; given the i-th branch node z_i of a ZDD, the subgraph induced by the set of nodes that are descendants of z_i is also a ZDD. Let the ZDD whose root node is z_i be Z_i, and the family of sets represented by Z_i be $\mathcal{T}_i$. Family of sets $\mathcal{T}_i$, $\mathcal{T}_{\mathrm{lo}(i)}$ and $\mathcal{T}_{\mathrm{hi}(i)}$ satisfy the following relationship.

$$\mathcal{T}_i = \mathcal{T}_{\mathrm{lo}(i)} \cup \{e_{\mathrm{v}(i)} \cup T | T \in \mathcal{T}_{\mathrm{hi}(i)}\}$$

Proposition 1. *Alg. 1 can find the optimal solution of the problem of Eq.(1), where we assume $\mathcal{T}$ is represented as an ordered ZDD.*

Proof. We use induction to give a proof that $S[i][j] = \max_T \sum_{e_i \in T} w_i$ after running our algorithm, where T is a set of tree nodes that satisfies $T \in \mathcal{T}_i$ and $\sum_{e_i \in T} l_i = j$. If $i = |Z| - 1$, then $\mathcal{T}_i = \{\emptyset\}$ and $S[i][0] = 0$ and $S[i][j] = -\infty$ for $j \neq 0$, which satisfies the condition. Suppose that $S[\mathrm{lo}(i)][j]$ and $S[\mathrm{hi}(i)][j]$ both satisfy the condition for $j = 0, \ldots, L$. If the set that maximizes $S[i][j]$ does not have $e_{\mathrm{v}(i)}$, then the set is contained in $\mathcal{T}_{\mathrm{lo}(i)}$, and its size is j. Therefore, the maximum weight equals $S[\mathrm{lo}(i)][j]$ (Alg.1 line 9). Otherwise, the set that maximizes $S[i][j]$ has $e_{\mathrm{v}(i)}$, so the item is contained in $\{e_{\mathrm{v}(i)} \cup T | T \in \mathcal{T}_{\mathrm{hi}(i)}\}$, and its weight is $S[\mathrm{hi}(i)][j - l_{\mathrm{v}(i)}] + w_{\mathrm{v}(i)}$ (Alg.1 line 6). Since Z_1 corresponds to the root node and it represents all possible solutions, $\max_j S[1][j]$ is the maximum weight of the subset that satisfies the length limit and is contained in $\mathcal{T}$. □

Proposition 2. *The time and space complexity of Alg. 1 are both $O(|Z|L)$.*

Proof. We have to store tables S, B and solution $\mathbf{r}$. The tables have $|Z| \times (L+1)$ entries and $|Z| \geq |\mathbf{r}|$, the space complexity is $O(|Z|L)$. For the time complexity, filling entries in S and B requires $O(|Z|L)$ time since to fill an entry requires constant time. Backtracking requires at most N updates, hence the time complexity is $O(|Z|L)$. □

We show an example of our algorithm in Fig. 2. Suppose that $D = \{e_1e_2, e_1e_3, e_2e_3\}$, $(l_1, l_2, l_3) = (1, 1, 3)$ and $(w_1, w_2, w_3) = (2, 1, 3)$. Set D is represented as the ZDD in Fig. 2(a). Let $L = 4$ and run the DP algorithm yielding tables S and B shown in

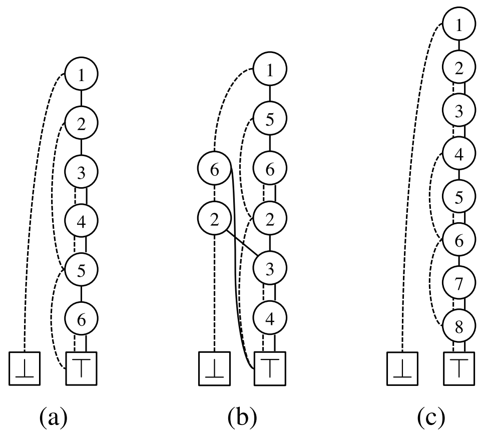


Figure 3: Example ZDDs representing the set of trimmed subtrees of the trees in Fig. 1. (a) Rooted-tree, (b) multi-rooted tree, and (c) nested-tree

Fig. 2(b,c). Suppose that we want to fill entry $S[1][4]$ (the upper right cell). There are two possible paths to reach the entry; the first path takes the high-edge from $S[2][3]$, and the second path takes the low-edge from $S[3][4]$. We use hollow and black arrows to represent these paths in Fig. 2(b). Since the former path results in weight 5, which is higher than that of the later path, hence we set $S[1][4] = 5$ and $B[1][4] =$ HI. After filling all the entries in tables, we can see $S[1][4]$ has the maximum weight, and the backtracking from $B[1][4] \to B[2][3] \to B[4][3] \to B[5][0]$. $B[5][0]$ corresponds to the $\top$ terminal node, and the backtracking yields the optimal solution e_1e_3.

5 ZDD Sizes

We give upper bounds on the size of the ZDD representing the family of sets $\mathcal{T}$ of Eq.(1) for the three problems. The number of subtrees contained in $\mathcal{T}$ may grow exponentially with the size of the original tree, however, we can represent them as a ZDD with very few nodes. Since the running time of our algorithm is $O(|Z|L)$, these theoretical upper bounds determine the running time of the proposed tree trimming algorithms.

We first give a proof of the size of the ZDD that represents all rooted subtrees of a given tree.

Proposition 3. *Given a tree with N nodes, we can construct a ZDD that represents all rooted subtrees of the tree whose number of nodes is $N + 2$, if we use a depth first pre-order of tree nodes as the order* *of ZDD labels.*

This result can be derived from the result of (Knuth, 2011), Chap.7.1.4, exercise 266. Fig. 3(a) is a ZDD that represents the set of all rooted subtrees of the tree in Fig. 1(a), where we employ pre-ordering $e_1, e_2, e_3, e_4, e_5, e_6$.

We next show the size of the ZDDs that represent the set of all subtrees of a multi-rooted tree.

Proposition 4. *Given an N node tree and the set of candidate root nodes R, the set of all possible subtrees can be represented by a ZDD whose number of nodes is $O(N \log |R|)$.*

Proof. (Sketch) The set of all possible subtrees can be represented as the union of the sets of rooted subtrees for different root $r \in R$. The set of rooted subtrees for a root node r can be represented as a ZDD that has $O(N)$ nodes, hence the set of ZDDs for different root nodes has $O(N|R|)$ nodes in total. We can further reduce this upper bound by employing appropriate depth first pre-ordering so as to share as many ZDD substructures as possible, and this ordering results in a union ZDD whose number of nodes is $O(N \log |R|)$. □

This proposition is related to a recently proved result that the set of all subtrees of an N-node tree can be represented as a ZDD whose number of nodes is $O(N \log N)$ (Yasuda et al., 2014). This is a special case of the above theorem that R equals the set of all nodes of the tree, i.e., $|R| = N$. The key point is to use the *heaviest-last depth first pre-order* as the ZDD label order. In this order, a node with the heaviest weight always comes after other siblings, where we define the weight of a node as the size of the maximum rooted subtree $T \in \mathcal{T}$ that is contained in its descendant tree. Fig. 3(b) is an example of the ZDD that represents the set of all possible rooted subtrees of the multi-rooted tree in Fig. 1(b), where the heaviest-last depth first pre-order is $e_1, e_5, e_6, e_2, e_3, e_4$.

The upper bound size of a ZDD for nested subtrees can be estimated by combining the above two theoretical results on rooted subtrees and multi-rooted subtrees.

Proposition 5. *For a nested tree whose sum of the number of nodes of inner trees is N, and the sets of candidate root nodes for inner trees are $R_1, \ldots, R_M$, where M is the number of inner trees, we can represent the set of possible nested subtrees by $O(N \log |R^*|)$, where $|R^*| = \max_i |R_i|$.*

Proof. (Sketch) The ZDD corresponding to the set of nested subtrees can be constructed as follows: first we make ZDDs that represent the set of rooted subtrees of the outer tree and inner trees. The outer tree is represented as a ZDD with $O(N)$ nodes, and the i-th inner tree is represented as a ZDD with $O(N_i \log |R_i|)$ nodes, where N_i is the number of nodes of the i-th inner tree. Then we can construct the ZDD for the nested tree by replacing each ZDD node of the outer-tree ZDD with the inner-tree ZDD corresponding to that node. □

Fig. 3(c) is a ZDD that represents the set of nested subtrees of the tree in Fig. 1(c), where we employ the order $e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8$.

We can directly prove the running time of the DP algorithm by exploiting the above three results to show the DP algorithm for the three problems takes $O(NL)$, $O(NL \log |R|)$, and $O(NL \log |R^*|)$ time, respectively. Here we assume that a ZDD that represents the set $\mathcal{T}$ is given. We need additional time for constructing a ZDD that represents $\mathcal{T}$ i.e. the input tree. However, ZDD construction also can be done in $O(|Z|)$ for the three tree trimming problems. We show details of ZDD construction in the next section.

6 Efficient ZDD Construction

We introduce here an efficient algorithm for constructing a ZDD that is used in the tree trimming problems. A ZDD can be constructed by repeatedly applying set operations between intermediate ZDDs, however, this process may be too slow since the running time of the set operations depends on the size of input and output ZDDs.

We first show the flow of an efficient ZDD construction algorithm for multi-rooted trees. This algorithm also can be used for constructing a ZDD for all rooted subtrees of a tree since a single-root tree is also a multi-rooted tree. The algorithm consists of two steps: first, we determine the appropriate order of ZDD nodes. We then use the top-down ZDD construction algorithm shown in (Knuth, 2011) (Chap.7.1.4, Exercise 55) to construct a ZDD. The top-down algorithm can efficiently construct a ZDD that represents the set of all connected components of a graph, and we can use it for constructing the set of all rooted subtrees with small modification. The running time of top-down construction algorithms may not be $O(|Z|)$, but our modified algorithm can obtain the ZDD in $O(|Z|)$ time by exploiting the structure of the input tree to avoid to make unnecessary ZDD nodes.

We can extend this ZDD construction algorithm to create ZDDs that represent the set of nested subtrees. We first compute the orders of outer tree and each inner tree, and then construct ZDDs for them using the top-down construction algorithm. Finally, we obtain the required ZDD by replacing ZDD nodes of the outer tree with the corresponding inner ZDDs. These procedure also can be done in $O(|Z|)$ time, since constructing the ZDDs for each tree takes time proportional to its size, and the ZDD substitution phase also takes time proportional to ZDD size.

7 Discussion

When solving a tree trimming problem, we sometimes want to add constraints to the problem so as to obtain better results. For example, Kikuchi et al. (2014) use additional constraints to set the minimum number of words (say θ words) extracted from a sentence if the sentence is contained in a summary, and require each selected inner tree to contain at least one verb and noun if the inner tree has them. Since our tree trimming approach can work once the ZDD that represents the set of feasible solutions is constructed, adding new constraints to the set of solutions can be easily performed by applying ZDD operations. These operations can be performed efficiently for many cases and the proposed approach will still work well. Moreover, we can extend the algorithm to construct ZDDs that represent the extended set of feasible solutions. We can also give theoretical upper bounds for the new constraint-added problem. In this nested tree case, we can prove that the number of ZDD nodes is $O(N\theta \log |R^*|)$.

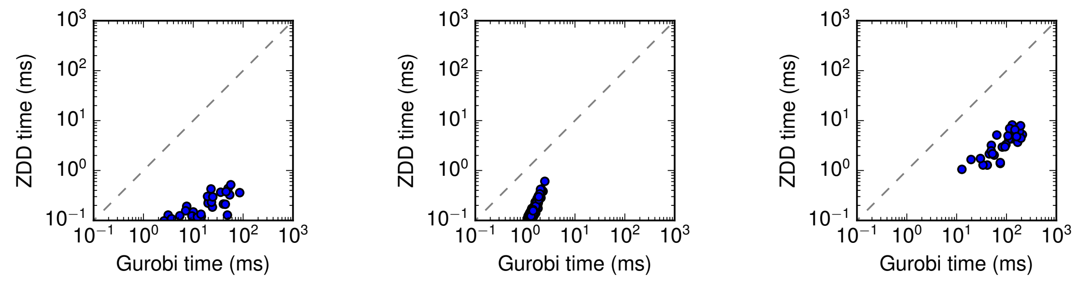
8 Experiments

We conduct experiments on the three tree trimming tasks of text summarization, sentence compression, and the combination of summarization and text compression. For the text summarization experiments, we use the test collection for summarization evaluation contained in the RST Discourse Treebank (RST-DTB) (Carlson et al., 2001), which is used in the previous work. The test collection consists of 30 documents with the reference summaries whose length is about 10% of the original document. We used the same parameters used in the previous papers. For sentence compression, we use the English compression corpus used in (Filippova and Strube, 2008), which consists of 82 news stories selected from the British National Corpus and American News Text Corpus, and consists of more than 1,300 sentences. We set the sizes of compressed sentences to be 70% of the original length, which is used in the original paper. We compare the proposed algorithm to Gurobi 5.5.0, a widely used commercial ILP solver[2]. It was run in the default settings and we used single-thread mode. We run Gurobi until it finds an optimal solution. Our algorithm was implemented in C++, and all experiments were conducted on a Linux machine with a Xeon E5-2670 2.60 GHz CPU and 192 GB RAM.

Fig. 4 compares the running time of our algorithm (includes ZDD construction time) and Gurobi. Each plotted marker in the figures represents a test instance, and if the position of a marker is below the dashed line, it means that our method is faster than Gurobi. We can see that our method is always faster than Gurobi; it was, at most, 300, 10, and 50 times faster in sentence extraction, sentence compression, and extraction & compression, respectively. Fig. 5,6 shows the relation between the input tree size and the ZDD construction times, and the relation between the input tree size and converted ZDD size respectively. These results show that both ZDD sizes and construction time were linear to the number of input tree nodes. The number of ZDD nodes looks like smaller than the $O(N \log N)$ bounds for multi-rooted trees and nested trees. This result is caused since the set of root candidate nodes R is small comparing with N for a typical input document.

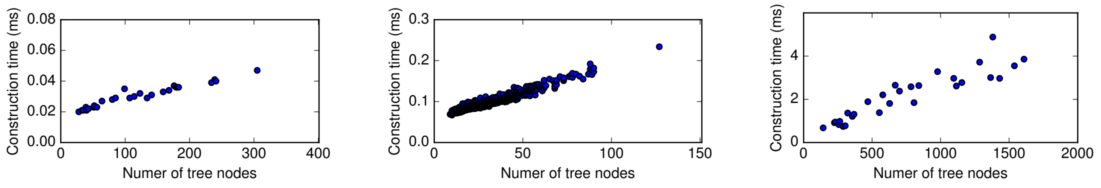
Next we conduct experiments to assess the scalability of the proposed method by solving problems with different input sizes. We choose the nested tree

[2]We also used CPLEX 12.5.1.0, but Gurobi shows better performance in most cases.



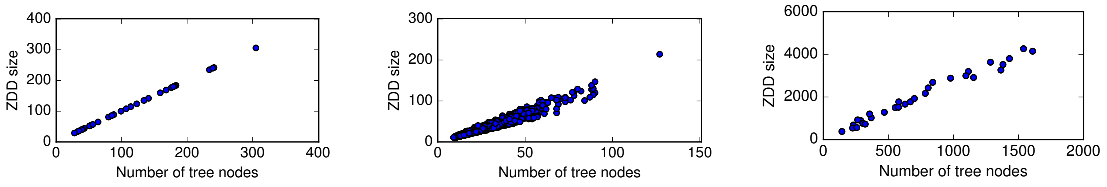
(a) Sentence extraction (b) Sentence compression (c) Extraction & compression

Figure 4: Performance comparison between the proposed method and Gurobi



(a) Sentence extraction (b) Sentence compression (c) Extraction & compression

Figure 5: ZDD construction time with number of input tree nodes



(a) Sentence extraction (b) Sentence compression (c) Extraction & compression

Figure 6: ZDD sizes with number of input tree nodes

trimming problem since it is the most complex problem. We make a large artificial nested tree by concatenating outer-trees of the nested trees of 30 RST-DT datasets. The results are shown in Fig. 7, and it shows that out method scales well with large inputs comparing with Gurobi.

9 Related Work

Recently proposed text summarization and sentence compression methods solve a task by formulating it as a combinatorial optimization problem (McDonald, 2007; Woodsend and Lapata, 2010; Martins and Smith, 2009; Clarke and Lapata, 2008). These combinatorial optimization-based formulations enable flexible models that can reflect the properties required. However, their complexity makes it difficult to solve optimization problems efficiently. These problems can be solved by using ILP solvers, however, they may fail to find optimal solutions and they have no guarantee on the running time. Since the proposed method is a DP algorithm and it has a theoretical guarantee, it always find an optimal solution in time proportional to the size of the input tree.

Our method also can be seen as a kind of fast text summarization algorithm. Previous fast algorithms are approximate algorithms (Qian and Liu, 2013; Lin and Bilmes, 2010; Lin and Bilmes, 2011; Davis et al., 2012), while our algorithm is an exact algorithm. Of course, there is a difference in task hardness since previous methods were designed for multi-document summarization and ours for single document summarization. Those works suggest

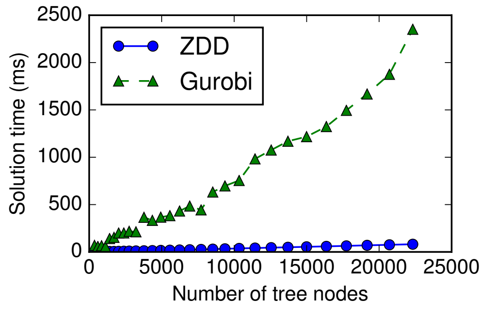


Figure 7: Solution time of our algorithm and Gurobi with different input tree sizes.

that algorithms that have guarantees on both running time and quality of solutions are highly demanding, and the proposed pseudo-polynomial time exact algorithm is valuable.

The Zero-suppressed Binary Decision Diagram (ZDD) (Minato, 1993) is a variant of the Binary Decision Diagram (BDD) (Akers, 1978; Bryant, 1986). BDD is a data structure that represents a Boolean function as a DAG, and ZDD can represent a family of sets in a compact form. Recently, ZDD and BDD have been used for solving optimization problems (Bergman et al., 2014a; Bergman et al., 2014b); they find the optimal solution by representing the set of feasible solutions in a BDD or its variants. Compared to these optimization methods, the proposed method differs in two main points. First, the proposed algorithm extends the ZDD-based optimization algorithm to solve knapsack problems. Second, it offers proofs of the size of ZDDs representing trimmed subtrees.

The ZDD-based method presented in this paper is related to our previous work of a BDD-constrained search (BCS) method (Nishino et al., 2015). In BCS, a BDD is used to solve constraints-added variants of shortest path problems on a DAG, and a 0-1 knapsack problem with additional constraints also can be solved by BCS. The main advantage of the DP-algorithm shown in this paper is that it has a theoretical guarantee on its running time which depends on only the size of the input tree. This advantage comes from using ZDD instead of BDD, and designing an algorithm specialized for variants of the knapsack problem. Though not obvious, it is possible to extend BCS to use ZDD instead of BDD and employ the label order used in this paper to give a theoretical bound that only depends on the size of an input tree. Nevertheless, the bound attained with this extension is worse than that shown in this paper.

10 Conclusion

We have proposed a DP algorithm for the tree trimming problems that appear in text summarization. Our approach always finds an optimal solution, and it runs in $O(NL \log N)$ time, where N is the number of tree nodes and L is the length limit. The key to our approach is to represent a set of subtrees of an input tree as a ZDD. By using ZDD, we can give a theoretical guarantee of the running time of the algorithm. Experiments show that the proposal allows three different tree trimming problems to be solved in the same way.

References

Sheldon B. Akers. 1978. Binary decision diagrams. *Computers, IEEE Transactions on*, C-27(6):509–516.

David Bergman, Andre A. Cire, and Willem-Jan van Hoeve. 2014a. MDD propagation for sequence constraints. *Journal of Artificial Intelligence Research*, 50:697–722.

David Bergman, Andre A Cire, Willem-Jan van Hoeve, and Tallys Yunes. 2014b. BDD-based heuristics for binary optimization. *Journal of Heuristics*, 20(2):211–234.

Randal E Bryant. 1986. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, C-35(8):677–691.

Lynn Carlson, Daniel Marcu, and Mary Ellen Okurowski. 2001. Building a discourse-tagged corpus in the framework of rhetorical structure theory. In *Proceedings of the Second SIGdial Workshop on Discourse and Dialogue - Volume 16*, SIGDIAL'01, pages 1–10.

Janara Christensen, Stephen Soderland, Gagan Bansal, and Mausam. 2014. Hierarchical summarization: Scaling up multi-document summarization. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, ACL'14, pages 902–912.

James Clarke and Mirella Lapata. 2008. Global inference for sentence compression an integer linear programming approach. *Journal of Artificial Intelligence Research*, 31(1):399–429.

Sashka T. Davis, John M. Conroy, and Judith .D. Schlesinger. 2012. Occams – an optimal combinatorial covering algorithm for multi-document summarization. In *IEEE 12th International Conference on Data Mining Workshops*, ICDMW, pages 454–463.

Elena Filatova and Vasileios Hatzivassiloglou. 2004. A formal model for information selection in multi-sentence text extraction. In *Proceedings of the 20th International Conference on Computational Linguistics*, COLING'04.

Katja Filippova and Yasemin Altun. 2013. Overcoming the lack of parallel data in sentence compression. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, EMNLP'13, pages 1481–1491.

Katja Filippova and Michael Strube. 2008. Dependency tree based sentence compression. In *Proceedings of the Fifth International Natural Language Generation Conference*, INLG'08, pages 25–32.

Tsutomu Hirao, Yasuhisa Yoshida, Masaaki Nishino, Norihito Yasuda, and Masaaki Nagata. 2013. Single-document summarization as a tree knapsack problem. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, EMNLP'13, pages 1515–1520.

Yuta Kikuchi, Tsutomu Hirao, Hiroya Takamura, Manabu Okumura, and Masaaki Nagata. 2014. Single document summarization based on nested tree structure. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, ACL'14, pages 315–320.

Donald E Knuth. 2011. *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1*. Addison-Wesley.

Hui Lin and Jeff Bilmes. 2010. Multi-document summarization via budgeted maximization of submodular functions. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, NAACL/HLT'10, pages 912–920.

Hui Lin and Jeff Bilmes. 2011. A class of submodular functions for document summarization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, ACL/HLT'11, pages 510–520.

André FT Martins and Noah A Smith. 2009. Summarization with a joint model for sentence extraction and compression. In *Proceedings of the Workshop on Integer Linear Programming for Natural Langauge Processing*, pages 1–9.

Ryan McDonald. 2007. A study of global inference algorithms in multi-document summarization. In *Proceedings of the 9th European Conference on Information Retrieval*, ECIR'07, pages 557–564.

Shin-ichi Minato. 1993. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Design Automation, 1993. 30th Conference on*, DAC'93, pages 272–277.

Yoshio Nakao. 2000. An algorithm for one-page summarization of a long text based on thematic hierarchy detection. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, ACL'00, pages 302–309.

Masaaki Nishino, Norihito Yasuda, Shin-ichi Minato, and Masaaki Nagata. 2015. BDD-constrained search: A unified approach to constrained shortest path problems. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, AAAI'15, pages 1219–1225.

Xian Qian and Yang Liu. 2013. Fast joint compression and summarization via graph cuts. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, EMNLP'13, pages 1492–1502.

Kristian Woodsend and Mirella Lapata. 2010. Automatic generation of story highlights. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL'10, pages 565–574.

Norihito Yasuda, Masaaki Nishino, and Shin-ichi Minato. 2014. On the size of the zero-suppressed binary decision diagram that represents all the subtrees in a tree. In *Trends and Applications in Knowledge Discovery and Data Mining*, Lecture Notes in Computer Science, pages 504–510.