On the Time Complexity of Bucket Elimination Algorithms

Javier Larrosa
Information and Computer Science
University of California at Irvine, USA

January 23, 2001

Abstract

In this short note, we prove the time complexity of full-bucket and mini-bucket elimination [1, 2]. In previous papers, when discussing the complexity of these algorithms, only the importance of the exponential contribution was emphasized and the rest of the contributions to the cost were not carefully considered. In this note, we address this fact and provide a non-trivial bound for the non-exponential contribution to the complexity of both algorithms. We demonstrate the result for the Additive Combinatorial Optimization Problem case.

1 Introduction

1.1 Additive Combinatorial Optimization Problem

An *additive combinatorial optimization problem* (ACOP) is defined by a tuple $(\mathcal{X}, \mathcal{D}, \mathcal{F})$ where

- $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a set of n variables identified by their index;
- $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a collection of finite domains, D_i is the set of possible values for variable x_i;
- $\mathcal{F} = \{f_1, \ldots, f_r\}$ is a set of r *cost functions.* A cost function f_i is a real valued function defined over a set of variables $var(f_i)$, called its *scope*. It assigns a *cost* to each tuple over $var(f_i)$. The scope size of a function is called its *arity*.

In the sequel, t will denote a tuple (an assignment of domain values to a set of variables) and $var(t)$ will denote the set of variables assigned by t. For notation simplicity, when we evaluate a tuple t on a cost functions f_i such that $var(f_i) \subset var(t)$, we write $f_i(t)$ when we really mean $f_i(t')$, where t' is the subtuple of t containing the assignment to variables in $var(f_i)$.

The sum of the cost functions defines the problem's *objective function* and the task is to *minimize* the objective function,

$$\min_t\{\sum_{i=1}^{r} f_i(t)\},\ t \in (D_1 \times \ldots \times D_n)$$

It is well known that solving ACOP is NP-*hard*. We assume that quering a cost functions is time $O(1)$.

1.2 Operations on Functions

- *Sum of functions*: Let f_i and f_j be two arbitrary functions. Their sum $f_i + f_j$ is a new function defined over the scope $var(f_i) \cup var(f_j)$ by,

$$(f_i + f_j)(t) = f_i(t) + f_j(t)$$

- *Variable elimination by minimization*: Let f_j be an arbitrary function and x_i a variable in its scope. The elimination of x_i from f_j, $\min_i\{f_j\}$, is a new function with scope $var(f_j) - \{x_i\}$ that assigns to each tuple the minimum cost extension to x_i. Formally, (t is a tuple over $var(f_j) - \{x_i\}$)

$$(\min_i\{f_j\})(t) = \min_{a \in D_i}\{f_j(t, a)\}$$

  where $f_j(t, a)$ means the evaluation of f_j on the extention of the tuple t with the assignment of value a to variable x_i.

Lemma 1.1 *Let $g_1, \ldots, g_k$ be a set of functions. The complexity of computing $\sum_{j=1}^{k} f_j$ is time $O(k \times exp(q))$, where q is the arity of the resulting function (i.e.: $q = |\cup_{j=1}^{k} var(f_j)|$).*

Proof: Clearly, computing $\sum_{j=1}^{k} f_{i_j}$ requires the computation of $O(exp(q))$ values (each one of the new function entries). Computing each individual value requires to query every original function, which has complexity $O(k)$. Therefore, the total cost is $O(k \times exp(q))$.

Lemma 1.2 *Let f be a function of arity k and V a subset of its scope. The complexity of computing $\min_V\{f\}$ is time exponential in the arity of f, $O(exp(k))$.*

Proof: Let s denote the size of V. The scope of $\min_V\{f\}$ is $var(f) - V$ and its arity is $k - s$. Consequently, computing $\min_V\{f\}$ requieres the computation of $O(exp(k - s))$ values. Computing each individual value requires to obtain the minimum out of $O(exp(s))$. Therefore, the total cost is $O(exp(k - s) \times exp(s))$ which is $O(exp(k))$.

Theorem 1.1 *Let $F = \{f_1, \ldots, f_k\}$ be a set of functions and let V be a subset of the combined scope of F.*

The complexity of computing $\min_V\{\sum_{j=1}^{k} f_j\}$ is $O(k \times exp(q))$, where q is the size of the combined scope of F

Proof: Trivial after the previous Lemmas.

2 Time Complexity of Bucket Elimination

In this section we prove that the complexity of Bucket Elimination (BE) [2] along variable ordering o is $O(r \times exp(w^*(o)))$, where r is the number of cost functions in the problem and $w^*(o)$ is the induced width of the problem along the ordering o. We show it by associating the algorithm to a tree such that leaves are the original cost functions in the problem and each internal node is a sequence of computations performed by the algorithm. Thus the cost of BE is the sum of costs over the nodes of the tree. We show that the cost of the computations in an arbitrary node is bounded by $O(exp(w^*(o)))$. We also show that the number of internal nodes of the tree is bounded by $2 \times r$. Therefore, it is clear that the total complexity of BE is $O(r \times exp(w^*(o)))$.

BE can be seen as a sequence of sums of function and variable eliminations. Therefore, given a problem instance and a variable ordering we can write the BE execution as an algebraic formulae of the form ($|V| \geq 0, k > 0$):

$$\min_V \{g_1 + \ldots + g_k\}$$

where each g is either an original cost function from the problem or has, recursively, the same form. It is important to note that each original function appear only once in the formulae.

Example 2.1 *Let's consider a problem instance defined over five variables* $\{x_1, \ldots, x_5\}$ *and four functions* $\{f_1(x_1, x_3), f_2(x_2, x_3), f_3(x_3, x_4), f_4(x_2, x_5)\}$. *The execution trace of bucket elimination along the lexicographical variable ordering is,*

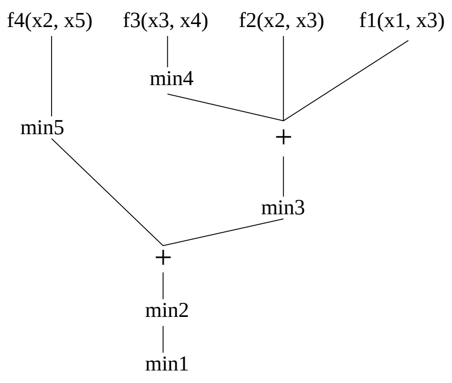
bucket$_5$*:*	$f_4(x_2, x_5)$
bucket$_4$*:*	$f_3(x_3, x_4)$
bucket$_3$*:*	$f_2(x_2, x_3), f_1(x_1, x_3)$ $\lambda_4(x_3) := \min_4\{f_3(x_3, x_4)\}$
bucket$_2$*:*	$\lambda_5(x_2) := \min_5\{f_4(x_2, x_5)\}$ $\lambda_3(x_1, x_2) := \min_3\{f_2(x_2, x_3) + f_1(x_1, x_3) + \lambda_4(x_3)\}$
bucket$_1$*:*	$\lambda_2(x_1) := \min_2\{\lambda_5(x_2) + \lambda_3(x_1, x_2)\}$
Result:	$\lambda_1() := \min_1\{\lambda_2(x_1)\}$

Therefore, we can expand the computation of $\lambda_1()$ *and express the whole execution as*

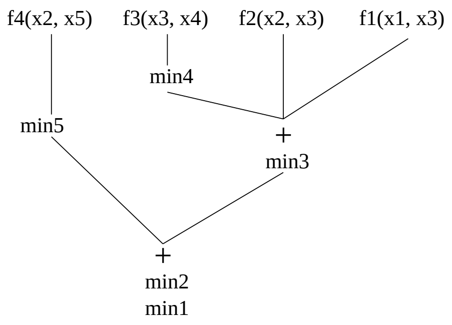
$$\min_{(1,2)}\{\min_5\{f_4(x_2, x_5)\} + \min_3\{\min_4\{f_3(x_3, x_4)\} + f_2(x_2, x_3) + f_1(x_1, x_3)\}\}$$

We associate a tree, denoted *computation tree* (CT), to the algebraic formulae. The original functions (arguments of the formulae) are the leaves and computations (either sum of functions or variable elimination) are the internal nodes. Figure 1.a depicts the computation tree associated to the example in 2.1.

CTs are not directly useful in our proof, because their number of nodes is not proportional to the number of original functions. This is because CTs may have *internal linear paths*, which are paths between an internal node v and one of its ancestors w, $(v, v_1 \ldots, v_k, w)$, such that all nodes in the path but v have only one child. For instance, the path between the lowest $+$ and the root in 1.a is an internal linear path. To overcome this problem, we define the *compact computation tree* (CCT) as the tree obtained from the CT where *internal linear paths* are merged into one node representing the sequence of computations . Figure 1.b depicts the compact computation tree of the example in 2.1. In the CCT, leaves are the original cost functions and internal nodes are sequences of computations. The sequence of computations of an internal node is has



a) Computation-tree



b) Compact Computation-tree

Figure 1: .

the following form $\min_V\{\sum_{j=1}^{k} g_j\}$ $(k \geq 0)$, where each g is either an original function or the result of the computation of a a child.

It is important to note that the number of nodes in the CCT is bounded by $3 \times r$ and the number of internal nodes in the CCT is bounded by $2 \times r$. The reason is that there are exactly r leaves and the compactation of internal linear paths guarantees that only leaf parents may have one child. Therefore there are at most $2r$ internal nodes.

It is clear that the CCT provides a trace of the execution of bucket elimination. Therefore, the complexity of BE is the complexity of the set of computations depicted in the CCT.

Theorem 2.1 *The complexity of bucket elimination along variable ordering o is time $O(r \times exp(w^*(o)))$, where r is the number of cost functions and $w^*(o)$ is the problem induced width along ordering o.*

Proof: The computation performed in an internal node v in the CTT is the sum of $ch(v)$ functions ($ch(v)$ denotes the number of children of node v, $ch(v) > 0$ in internal nodes) and the subsequent elimination of q variables $(q \geq 0)$ from the resulting function. The combined arity of the $ch(v)$ functions (namely, the functions *arriving* to node v) is bounded by $w^*(o) + 1$. Therefore, using

Theorem 1.1 we can bound the complexity of the computation at node v by $O(ch(v) \times exp(w^*(o)+1))$. The total cost is the sum of costs over internal nodes $O((\sum_v ch(v)) \times exp(w^*(o)+1))$. It is obvious that the sum of children over non-leave nodes is equal to the number of nodes in the tree minus one, which we showed that is bounded by $3 \times r$. Consequently, the total cost is bounded by $O(3 \times r \times exp(w^*(o)+1))$ which is equal to $O(r \times exp(w^*(o)))$

3 Time Complexity of Mini-bucket Elimination

Mini-bucket elimination (MB) [1] can also be expressed as a sequence of sum functions and variable eliminations. Therefore, given a problem instance and a variable ordering we can also express the MB execution as an algebraic formulae. As before, it is important to note that each function appear only once in the formulae.

Example 3.1 *The execution trace of* (2)*-mini-bucket elimination in the problem of Example 2.1 along the lexicographical variable ordering is,*

bucket$_5$:	$f_4(x_2, x_5)$
bucket$_4$:	$f_3(x_3, x_4)$
bucket$_3$:	$f_2(x_2, x_3), f_1(x_1, x_3)$ $\lambda_4(x_3) := \min_4\{f_3(x_3, x_4)\}$
bucket$_2$:	$\lambda_5(x_2) := \min_5\{f_4(x_2, x_5)\}$ $\lambda_{(3,1)}(x_2) := \min_3\{f_2(x_2, x_3) + \lambda_4(x_3)\}$
bucket$_1$:	$\lambda_{(3,2)}(x_1) := \min_3\{f_1(x_1, x_3)\}$
Result:	$\lambda_2() := \min_2\{\lambda_5(x_2) + \lambda_{(3,1)}(x_2)\}$ $\lambda_1() := \min_1\{\lambda_{(3,2)}(x_1)\}$

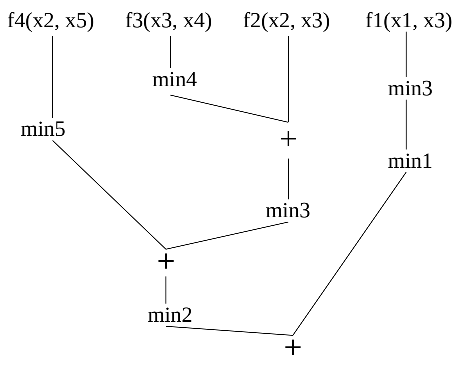
Therefore, we can expand the computation and express the whole execution as,

$$\min_2\{\min_5\{f_4(x_2, x_5)\} + \min_3\{\min_4\{f_3(x_3, x_4)\} + f_2(x_2, x_3)\}\} + \min_{(1,3)}\{f_1(x_1, x_3)\}$$

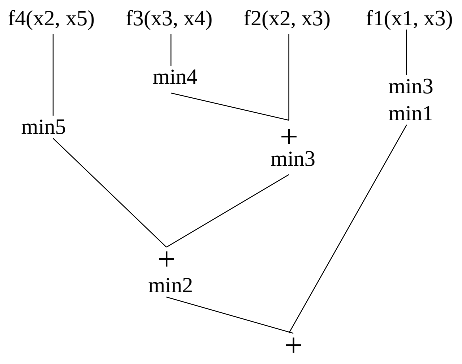
We can also represent a MB execution by means of a computation tree that can be compacted into a *compact computation tree* (CCT). The *computation tree* of the example is depicted in Figure 2.a and the *compact computation tree* (CCT) is depicted in Figure 2.b

If BE and MB are executed with the same problem and the same variable ordering, The CTT of MB is likely to have more nodes (with less computation in each). It is important to note that, although possibly having more nodes, the total number nodes of the CTT associated to a MB execution is still bounded by $3r$. As in the BE case, the CTT has exactly r leaves and the compactation guarantees that only parents of leaves may have one child. Therefore, the number of internal nodes is bounded by $2r$ and the total number of nodes is bounded by $3r$. The complexity of MB is the sum of complexities of internal nodes in the CTT, which yields the following result.

Theorem 3.1 *The complexity of mini-bucket elimination with accuracy parameter i along variable ordering o is time $O(r \times exp(i))$, where r is the number of cost functions.*



a) Computation-tree



a) Compact Computation-tree

Figure 2: .

Proof: The computation performed at an internal node v in the CTT is the sum of $ch(v)$ functions and the subsequent elimination of q variables ($q \geq 0$) from the resulting function. By definition of (i)-MB, the combined arity of the $ch(v)$ functions *arriving* to node v is bounded by i. Therefore, using Theorem 1.1 we can bound the complexity of the computation at node v by $O(ch(v) \times exp(i))$. The sum of children over non-leave nodes is equal to the number of nodes in the tree minus one, which is bounded by $3 \times r$. Consequently, the total cost is bounded by $O(3 \times r \times exp(i+1))$ which is assymptotically equivalent to $O(r \times exp(i))$

References

[1] R. Dechter. Mini-Buckets: A General Scheme for Generating Approximations in Automated Reasoning. In *Proceeding of IJCAI'97*, pg. 1297–1303, 1997.

[2] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85, 1999.