When Can We Reduce Multi-Variable Range Estimation Problems to Two Fewer-Variable Problems?*

Joe Lorkowski, Olga Kosheleva, Luc Longpré, and Vladik Kreinovich
University of Texas at El Paso, 500 W. University,
El Paso, TX 79968, USA
lorkowski@computer.org,olgak@utep.edu,longpre@utep.edu.vladik@utep.edu

Abstract

Sometimes, a function f of n variables can be represented as a composition of two functions of fewer variables. In this case, the problem of computing the range of f on given intervals can be reduced to two range-computation problems with fewer variables. In this paper, we describe a feasible algorithm that checks whether such a reduction is possible – and, if it is possible, produces the desired reduction.

Keywords: interval computations, range estimation, reduction to simpler problems, directed graphs
AMS subject classifications: 65G20, 65G40, 05C20

1 Formulation of the Problem

Practical need for range estimation. In many practical situations, we are interested in the value of a physical quantity y that is difficult (or even impossible) to measure directly. Examples of such quantities are the distance to a faraway star, the amount of oil in an oil field, etc. To estimate this quantity y, we find easier-to-measure quantities $x_1, \ldots, x_n$ which are related to y by a known dependence $y = f(x_1, \ldots, x_n)$. Then, we measure x_i, and use the measurement results $\widetilde{x}_i$ to estimate y as $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$. Measurements are never absolutely accurate: the measurement result $\widetilde{x}_i$ is, in general, different from the actual (unknown) value x_i. In many practical situations, the only information about the measurement error $\Delta x_i \stackrel{\text{def}}{=} \widetilde{x}_i - x_i$ is the upper bound Δ_i on its absolute value: $|\Delta x_i| \le \Delta_i$; see, e.g., [11]. In this case, the only information that we have about the actual (unknown) value x_i is that this value belongs to the interval $[\underline{x}_i, \overline{x}_i] \stackrel{\text{def}}{=} [\widetilde{x}_i - \Delta_i, \widetilde{x}_i + \Delta_i]$. Thus, the set of possible values of $y = f(x_1, \ldots, x_n)$ is equal to

$$\{f(x_1, \ldots, x_n) : x_1 \in [\underline{x}_1, \overline{x}_1], \ldots, x_n \in [\underline{x}_n, \overline{x}_n]\}.$$

*Submitted: November 6, 2014; Revised: June 24, 2015; Accepted:August 6, 2015.

This set is known as the *range* of the function $f(x_1, \ldots, x_n)$ over intervals $[\underline{x}_i, \overline{x}_i]$. For a continuous function $f(x_1, \ldots, x_n)$, this range is an interval; this interval is usually denoted by $[\underline{y}, \overline{y}]$. Computing such a range is one of the main problems of *interval computations*; see, e.g., [7, 10].

Range estimation is difficult. It is known [2, 3, 8] that, in general, the problem of estimating the range is NP-hard. Crudely speaking, this means that, unless P=NP (which most computer scientists believe to be not possible), every algorithm for computing the exact range requires, in some cases, computational time which grows exponentially with the bit length of the formulation of the problem – and for problem of medium size $n \approx 300$, the value of the exponential function such as 2^n becomes larger than the lifetime of the Universe and thus, unfeasible.

When the dependence is rational (or algebraic) and the number of variables is fixed, the problem becomes feasible (solvable in polynomial time); see, e.g., [8]. However, as the number of variables increases, the problem is NP-hard.

Sometimes, the range estimation problem can be reduced to two simpler problems. In some cases, the function $f(x_1, \ldots, x_n)$ can be represented as a composition of two functions of fewer variables, i.e., has a form

$$f(x_1, \ldots, x_n) = F(a(x_1, \ldots, x_k), x_{k+1}, \ldots, x_n) \tag{1}$$

for some $k \geq 2$. In this case, the original problem of computing the range of a function of n variables can be reduced to two (simpler) problems of computing ranges of functions with fewer variables:

- first, we compute the range $[\underline{a}, \overline{a}] = \{a(x_1, \ldots, x_k) : x_i \in [\underline{x}_i, \overline{x}_i]\}$ of a function of $k < n$ variables;
- then, we compute the range

  $$[\underline{y}, \overline{y}] = \{F(a, x_{k+1}, \ldots, x_n) : a \in [\underline{a}, \overline{a}], x_j \in [\underline{x}_j, \overline{x}_j]\}$$

  of a function of $n - k + 1 < n$ variables.

Comment. It is worth mentioning that in terms of the computation graph, such a possibility corresponds, in effect, to a partitioning of this graph. We will explicitly use this graph reformulation of our problem in Section 3.

Example. Such a representation is possible, e.g., for *single-use expressions*, i.e., expressions in which each variable occurs only once [5, 7, 9, 10]. The simplest examples are the sum and product of multiple variables. For example, the sum $f(x_1, \ldots, x_n) = x_1 + \ldots + x_n$ can be represented as $(x_1 + x_2) + x_3 + \ldots + x_n$, i.e., as $F(a(x_1, x_2), x_3, \ldots, x_n)$, where $a(x_1, x_2) = x_1 + x_2$ and $F(a, x_3, \ldots, x_n) = a + x_3 + \ldots + x_n$. For $n \geq 4$, the function $F(a, x_3, \ldots, x_n)$ can also be represented in the same form, as $(a + x_3) + x_4 + \ldots + x_n$, etc.

Similarly, the product $f(x_1, \ldots, x_n) = x_1 \cdot \ldots \cdot x_n$ can be represented as

$$(x_1 \cdot x_2) \cdot x_3 \cdot \ldots \cdot x_n,$$

i.e., as $F(a(x_1, x_2), x_3, \ldots, x_n)$, where $a(x_1, x_2) = x_1 \cdot x_2$ and $F(a, x_3, \ldots, x_n) = a \cdot x_3 \cdot \ldots \cdot x_n$. For $n \geq 4$, the function $F(a, x_3, \ldots, x_n)$ can also be represented in the same form, as $(a \cdot x_3) \cdot x_4 \cdot \ldots \cdot x_n$, etc.

Another example of a single-use expression is $f(x_1, x_2, x_3) = (x_1 + x_2) \cdot x_3$. In this case, $k = 2$, $a(x_1, x_2) = x_1 + x_2$, and $F(a, x_3) = a \cdot x_3$.

When is such a reduction possible? In some cases, we have an explicit expression of the above type (1). Sometimes, the expression is different, but the function can be expressed in this form. For example, the function $f(x_1, x_2, x_3) = x_1 \cdot x_3 + x_2 \cdot x_3$ does not explicitly have the desired form, but it can be transformed into an equivalent expression $f(x_1, x_2, x_3) = (x_1 + x_2) \cdot x_3$, for which such a transformation is possible.

It is therefore desirable, given a function $f(x_1, \ldots, x_n)$, to be able to check when such a simplifying representation is possible – and if it is possible, produce such a representation. In this paper, we describe a feasible algorithm for such checking.

Comment. In particular, this algorithm can help in estimating the ranges of expression with duplicated variables, such as $f(x_1, x_2, x_3) = x_1 \cdot x_3 + x_2 \cdot x_3$, if they can be reduced to a form that allows a reduction to fewer-variables problems.

2 Case When We Know Which Variables $x_1, \ldots, x_k$ Enter Only Via Some Combination $a(x_1, \ldots, x_k)$

Description of the case. Let us start with a simple case, when we know which variables $x_1, \ldots, x_k$ affect the value $f(x_1, \ldots, x_n)$ only via some combination $a(x_1, \ldots, x_k)$.

Main assumptions. Our first assumption is that all three functions $f(x_1, \ldots, x_n)$, $a(x_1, \ldots, x_k)$ and $F(a, x_{k+1}, \ldots, x_n)$ are differentiable, and that for each algorithmically given function, we can compute all its partial derivatives in feasible time. This assumption is in line with the existence of many feasible algorithms for automatic differentiation; see, e.g., [4].

Our second assumption is that, given two algorithmic functions $f(x_1, \ldots, x_n)$ and $g(x_1, \ldots, x_n)$, we can feasibly check whether the corresponding functions always produce the same results, i.e., whether it is true that

$$\forall x_1 \ \ldots \ \forall x_n \, (f(x_1, \ldots, x_n) = g(x_1, \ldots, x_n)).$$

This assumption is not true in the most general case, since, in general, the problem of checking whether the two functions always produce the same results is equivalent to the non-decidable "Is this expression always equal to zero" problem; see, e.g., [8].

In many cases, however, this assumption is, for all practical purposes, satisfied. For example, for analytical functions, we can simply check whether their values are equal to a randomly generated tuple $(\xi_1, \ldots, \xi_n)$. If $f(\xi_1, \ldots, \xi_n) \neq g(\xi_1, \ldots, \xi_n)$, then, of course, the functions $f(x_1, \ldots, x_n)$ and $g(x_1, \ldots, x_n)$ are different.

Vice versa, if the functions are different, then the set of all the tuples $(x_1, \ldots, x_n)$ for which the difference is equal to 0 (i.e., for which $f(x_1, \ldots, x_n) - g(x_1, \ldots, x_n) = 0$) is small – it is of lower dimension than n. Thus, the probability that a random tuple will land in this set is 0 – or, if we take into account that we perform computations with finite accuracy, very small. Moreover, this probability can be made as small as possible by repeatedly generating random tuples and checking whether the two given functions attain the same value on all these tuples. Thus, if $f(\xi_1, \ldots, \xi_n) = g(\xi_1, \ldots, \xi_n)$, then with probability close to 1, we can conclude that the values of the functions $f(x_1, \ldots, x_n)$ and $g(x_1, \ldots, x_n)$ are equal for all the tuples $(x_1, \ldots, x_n)$.

In our algorithm, we will count each call to one of these auxiliary algorithms – of differentiation or of checking function equality – as one step.

Analysis of the problem. Let us first show how, based only on the knowledge that the function $f(x_1, \ldots, x_n)$ can be represented in the form (1), we can actually extract the corresponding functions $a(x_1, \ldots, x_k)$ and $F(a, x_{k+1}, \ldots, x_n)$.

The selection of a and F is not unique: instead of the original function $a(x_1, \ldots, x_k)$, we can use a function $a' = g(a(x_1, \ldots, x_k))$ for some invertible $g(x)$; then

$$f(x_1, \ldots, x_n) = F'(a'(x_1, \ldots, x_k), x_{k+1}, \ldots, x_n),$$

where

$$F'(a, x_{k+1}, \ldots, x_n) \stackrel{\text{def}}{=} F(g^{-1}(a), x_{k+1}, \ldots, x_n),$$

and $g^{-1}(y)$ denotes an inverse function, for which $x = g^{-1}(y)$ if and only if $y = g(x)$.

Let us therefore fix a combination of values $v_{k+1}, \ldots, v_n$. Then, the value $f(x_1, \ldots, x_k, v_{k+1}, \ldots, v_n)$ depends only on $a(x_1, \ldots, x_k)$ and therefore can serve as an alternative function $a'(x_1, \ldots, x_k)$. To find the appropriate function $F'(a, x_{k+1}, \ldots, x_n)$, we pick some values $v_2, \ldots, v_k$, then find x_1' for which $a'(x_1', v_2, \ldots, v_k) = a'(x_1, x_2, \ldots, x_k)$, i.e., for which

$$f(x_1', v_2, \ldots, v_k, v_{k+1}, \ldots, v_n) = a' = f(x_1, \ldots, x_k, v_{k+1}, \ldots, v_n).$$

Since the dependence of f on $x_1, \ldots, x_k$ is only through the combination

$$a'(x_1, \ldots, x_k) = f(x_1, \ldots, x_k, v_{k+1}, \ldots, v_n),$$

and we have

$$a'(x_1', v_2, \ldots, v_k) = a'(x_1, x_2, \ldots, x_k),$$

we can therefore conclude that

$$f(x_1, \ldots, x_k, x_{k+1}, \ldots, x_n) = f(x_1', v_2, \ldots, v_k, x_{k+1}, \ldots, x_n).$$

In other words, here

$$F(a, x_{k+1}, \ldots, x_n) = f(x_1'(a'), v_2, \ldots, v_k, x_{k+1}, \ldots, x_n),$$

where $x_1'(a)$ is the value for which $f(x_1', v_2, \ldots, v_k, v_{k+1}, \ldots, v_n) = a'$.

Due to this possibility, it is sufficient to be able to check whether such a representation is possible – since if it is possible, the above text describes the corresponding functions F and a.

To check whether such a representation is possible, let us differentiate the function f. Due to the chain rule, for each i from 1 to k, the i-th partial derivative of the function $f(x_1, \ldots, x_n)$ has the form

$$\frac{\partial f}{\partial x_i} = \frac{\partial F}{\partial a}(a, x_{k+1}, \ldots, x_n) \cdot \frac{\partial a}{\partial x_i}(x_1, \ldots, x_k).$$

Thus, for $i_0 = 1$ and for every index $i \le k$, the ratio

$$r_i \stackrel{\text{def}}{=} \frac{\dfrac{\partial f}{\partial x_i}}{\dfrac{\partial f}{\partial x_{i_0}}} \tag{2}$$

is equal to

$$r_i = \frac{\dfrac{\partial f}{\partial x_i}}{\dfrac{\partial f}{\partial x_{i_0}}} = \frac{\dfrac{\partial a}{\partial x_i}(x_1, \ldots, x_k)}{\dfrac{\partial a}{\partial x_{i_0}}(x_1, \ldots, x_k)}.$$

Therefore, the ratio r_i depends only on the variables $x_1, \ldots, x_k$: $r_i = r_i(x_1, \ldots, x_k)$. Hence, for every $j > k$, we have $d(i, i_0, j) = 0$, where

$$d(i, i_0, j) \stackrel{\text{def}}{=} \frac{\partial}{\partial x_j} \left(\frac{\dfrac{\partial f}{\partial x_i}}{\dfrac{\partial f}{\partial x_{i_0}}} \right). \tag{3}$$

Let us show that, vice versa, if the condition $d(i, i_0, j) = 0$ is satisfied for all $i \le k$ and all $j > k$, then the function $f(x_1, \ldots, x_n)$ can be represented (at least locally) in the desired form (1). Indeed, let us assume that $d(i, i_0, j) = 0$ for all $i \le k$ and for all $j > k$. Let us show that then, the value of $f(x_1, \ldots, x_k, x_{k+1}, \ldots, x_n)$ depends only on

$$a(x_1, \ldots, x_k) = f(x_1, \ldots, x_k, v_{k+1}, \ldots, v_n),$$

i.e., that if

$$f(x_1, \ldots, x_k, v_{k+1}, \ldots, v_n) = f(x'_1, \ldots, x'_k, v_{k+1}, \ldots, v_n),$$

then

$$f(x_1, \ldots, x_k, x_{k+1}, \ldots, x_n) = f(x'_1, \ldots, x'_k, x_{k+1}, \ldots, x_n).$$

Indeed, locally, all pairs of tuples $(x_1, \ldots, x_k)$ and $(x'_1, \ldots, x'_k)$ with the same value of $a(x_1, \ldots, x_k)$ can be connected by a smooth path $(x_1(t), \ldots, x_k(t))$ of such tuples, i.e., tuples for which

$$a(x_1(t), \ldots, x_k(t)) = f(x_1(t), \ldots, x_k(t), v_{k+1}, \ldots, v_n) = \text{const}. \tag{4}$$

Differentiating the formula (4) by t, we conclude that

$$\sum_{i=1}^{k} \frac{\partial f}{\partial x_i}(x_1(t), \ldots, x_k(t), v_{k+1}, \ldots, v_n) \cdot \frac{dx_i}{dt} = 0. \tag{5}$$

Due to the condition $d(i, i_0, j) = 0$, where $d(i, i_0, j)$ is defined by the formula (3), the ratio (2) does not depend on any of the variables. Thus, this ratio only depends on the variables $x_1, \ldots, x_k$: $r_i = r_i(x_1, \ldots, x_k)$. So, for every i from 1 to k, we have

$$\frac{\partial f}{\partial x_i}(x_1(t), \ldots, x_k(t), x_{k+1}, \ldots, x_n) =$$

$$\frac{\partial f}{\partial x_1}(x_1(t), \ldots, x_k(t), x_{k+1}, \ldots, x_n) \cdot r_i(x_1, \ldots, x_k).$$

Substituting this expression for partial derivative into the formula (5), we conclude that

$$\sum_{i=1}^{k} \frac{\partial f}{\partial x_1}(x_1(t), \ldots, x_k(t), v_{k+1}, \ldots, v_n) \cdot a_i(x_1 \ldots, x_k) \cdot \frac{dx_i}{dt} = 0.$$

Dividing both sides of this equality by $\dfrac{\partial f}{\partial x_1}(x_1(t), \ldots, x_k(t), v_{k+1}, \ldots, v_n)$ and multiplying both sides by $\dfrac{\partial f}{\partial x_1}(x_1(t), \ldots, x_k(t), x_{k+1}, \ldots, x_n)$, we conclude that

$$\sum_{i=1}^{k} \frac{\partial f}{\partial x_1}(x_1(t), \ldots, x_k(t), x_{k+1}, \ldots, x_n) \cdot a_i(x_1 \ldots, x_k) \cdot \frac{dx_i}{dt} = 0,$$

hence

$$\sum_{i=1}^{k} \frac{\partial f}{\partial x_i}(x_1(t), \ldots, x_k(t), x_{k+1}, \ldots, x_n) \cdot \frac{dx_i}{dt} = 0.$$

The left-hand side of this formula is the derivative of the expression $f(x_1(t), \ldots, x_k(t), x_{k+1}, \ldots, x_n)$ with respect to the parameter t. Since the derivative of this expression with respect to t is equal to 0, this expression does not depend on t, and thus, we have

$$f(x_1, \ldots, x_k, x_{k+1}, \ldots, x_n) = f(x_1', \ldots, x_k', x_{k+1}, \ldots, x_n).$$

So, if $a(x_1, \ldots, x_k) = a(x_1', \ldots, x_k')$, then

$$f(x_1, \ldots, x_k, x_{k+1}, \ldots, x_n) = f(x_1', \ldots, x_k', x_{k+1}, \ldots, x_n),$$

i.e., the value of f depends only on the combination $a(x_1, \ldots, x_k)$. The statement is proven.

Resulting algorithm. We consider the situation when we know which variables $x_1, \ldots, x_k$ should be joined together, we are just checking whether the representation (1) is possible – and if it is possible, generating the corresponding functions a and F.

For that, we check, for $i_0 = 1$, whether $d(i, i_0, j) = 0$ for all $1 < i \le k$ and all $j > k$, where $d(i, i_0, j)$ is determined by the formula (2). If this equality holds for all such i and j, then the representation (1) is possible, otherwise such a representation is not possible.

If the representation is possible, then we find the corresponding functions $a(x_1, \ldots, x_k)$ and $F(a, x_{k+1}, \ldots, x_n)$ as follows:

- we fix some values $v_2, \ldots, v_n$;
- we take $a(x_1, \ldots, x_k) \stackrel{\text{def}}{=} f(x_1, \ldots, x_k, v_{k+1}, \ldots, v_n)$, and
- we take $F(a, x_{k+1}, \ldots, x_n) \stackrel{\text{def}}{=} f(x_1(a), v_2, \ldots, v_k, x_{k+1}, \ldots, x_n)$, where $x_1(a)$ is the value for which $f(x_1, v_2, \ldots, v_k, v_{k+1}, \ldots, v_n) = a$.

Computational complexity of this algorithm. Checking each condition $d(i, i_0, j) = 0$ requires three differentiations, one division, and one call to an auxiliary algorithm for checking the equality of two functions – overall, a constant number of steps. We need to check this condition for all pairs (i, j). The total number of such pairs does not exceed $O(n^2)$. For each pair, we need a constant number of steps, so the overall complexity of this algorithm is $O(n^2)$.

Example. Let us illustrate this algorithm on the above example of the function $f(x_1, x_2, x_3) = x_1 \cdot x_3 + x_2 \cdot x_3$, for which we know that the variables x_1 and x_2 go together (i.e., $k = 2$). In this example, $\dfrac{\partial f}{\partial x_1} = x_3$ and $\dfrac{\partial f}{\partial x_2} = x_3$, so the ratio (2) is equal to 1. Thus, the derivative $d(1, 2, 3)$ of this ratio with respect to x_3 is equal to 0.

The equality condition is satisfied, which means that the desired representation is possible. The above algorithm explains how to generate the corresponding functions $a(x_1, x_2)$ and $F(a, x_3)$. for example, if we take $v_2 = v_3 = 1$, then we get $a(x_1, x_2) = f(x_1, x_2, 1) = x_1 \cdot 1 + x_2 \cdot 1 = x_1 + x_2$.

Here, $F(a, x_3) = f(x_1(a), 1, x_3)$, where $a(x_1)$ is the value for which $f(x_1, 1, 1) = x_1 + 1 = a$, i.e., $x_1(a) = a - 1$. Thus,

$$F(a, x_3) = f(a - 1, 1, x_3) = (a - 1) \cdot x_3 + 1 \cdot x_3.$$

One can check that this expression is indeed equal to $a \cdot x_3$.

Comment. If we select different values v_i, we get different expressions for $a(x_1, x_1)$ and $F(a, x_3)$. For example, for $v_2 = v_3 = 2$, we get $a(x_1, x_2) = 2x_1 + 2x_2$ and $F(a, x_3) = \frac{1}{2} \cdot a \cdot x_3$.

3 General Case

Analysis of the problem. In the previous section, we considered the case when we know which variables should be jointly considered, we just want to confirm our guess and to generate the corresponding functions $a(x_1, \ldots, x_k)$ and $F(a, x_{k+1}, \ldots, x_n)$. In practice, we do not know which of n variables can be combined. In principle, it is possible to try all possible subsets of the set of all n variables, but there are 2^n such subsets, and testing them all would requires unfeasible exponential time. It is therefore necessary to devise a feasible algorithm for finding such variables.

Such a feasible algorithm is presented in this section.

Let us assume that there are some variables $x_{i_1}, \ldots, x_{i_k}$ for which the function $f(x_1, \ldots, x_n)$ depends only on some combination $a(x_{i_1}, \ldots, x_{i_k})$. In other words, we assume that if $a(x'_{i_1}, \ldots, x'_{i_k}) = a(x_{i_1}, \ldots, x_{i_k})$ and $x'_j = x_j$ for all other indices j, then $f(x_1, \ldots, x_n) = f(x'_1, \ldots, x'_n)$.

If we pick one of the indices i_0 from the list, then what we need is to find a subset $I = \{i_1, \ldots, i_k\}$ such that $d(i, i_0, j) = 0$ for all $i \in I - \{i_0\}$ and for all $j \notin I$. To check the existence of such a set, let us build a directed graph G in which vertices are indices $1, \ldots, n$ except for i_0, and there is an edge from i to j if $d(i, i_0, j) \neq 0$.

To formulate our result, we need to recall some notions related to directed graphs; see, e.g., [1]. By a *path* from a vertex a to a vertex b, we mean a sequence a vertices $a = a_0, a_1, \ldots, a_p = b$ for which, for every i, there is an edge from a_i to a_{i+1}. We say that two vertices a and b in a directed graph are called *strongly connected* if there is a path from a to b and there is also a path from b to a. One can easily check that strong connectedness is an equivalence relation, so it divides the graph into equivalence classes. These classes are known as *strongly connected components*. There exist algorithms that find the strongly connected components in linear time $O(n)$ [1].

Our auxiliary result is that the desired set I exists if and only if there is more than one strongly connected component.

Indeed, if a set I exists, then there is no edge going from I to its complement J. So, if a path starts in the set I, this path cannot reach into J, and thus, vertices from I cannot belong to the same strongly connected component as vertices from the set J.

Vice versa, if there is more than one component, then the relation "there is a path from a vertex in C to a vertex in C'" provides a partial order on the set of these components. There is thus at least one component C which is maximal in this order, i.e., from which no edge is going out. Let us show that this component C satisfies the property of the desired set I. Indeed, since no edge is going from $i \in C$ to other elements $j \notin C$, we have $d(i, i_0, j) = 0$ for all $i \in C$ and $j \notin C$, which is exactly what we wanted.

Resulting algorithm. First, we compute the values $d(i, i_0, j)$ for all i, i_0, and j. Then, for each of n possible indices $i_0 = 1, \ldots, n$:

- we form a graph G whose vertices are indices $1, \ldots, n$, and in which there is an edge from i to j if $d(i, i_0, j) \neq 0$;

- then, we apply the known algorithm to find strongly connected components of this graph.

We stop either when we exhaust all n indices and all the graphs are strongly connected – in this case, a representation (1) is not possible, or when we find an index i_0 for which the corresponding graph has more than one strongly connected components. Once we find such i_0, then out of the corresponding components we select one which does not have any edges going out. We then take the union of this component and the index i_0 as the set $I = \{i_1, \ldots, i_k\}$.

Let $j_1, \ldots, j_{n-k}$ describe all the indices which do not belong to the set I. Then, we find the corresponding functions $a(x_{i_1}, \ldots, x_{i_k})$ and $F(a, x_{j_1}, \ldots, x_{j_{n-k}})$ as follows:

- we select some values $v_1, \ldots, v_n$;
- we take $a(x_{i_1}, \ldots, x_{i_k}) \stackrel{\text{def}}{=} f(x_{i_1}, \ldots, x_{i_k}, v_{j_1}, \ldots, v_{j_{n-k}})$, and
- we take $F(a, x_{j_1}, \ldots, x_{j_{n-k}}) \stackrel{\text{def}}{=} f(x_{i_1}(a), v_{i_2}, \ldots, v_{i_k}, x_{j_1}, \ldots, x_{j_{n-k}})$, where $x_{i_1}(a)$ is the value for which $f(x_{i_1}, v_{i_1}, \ldots, v_{i_k}, v_{j_1}, \ldots, v_{j_{n-k}}) = a$.

Comment. Our use of graphs to describe dependencies is similar to the use of graph coloring to help compute sparse Jacobian and Hessian matrices by using finite differences or automatic differentiation; see, e.g., [6].

Computational complexity of this algorithm. Checking each condition $d(i, i_0, j) = 0$ requires a constant number of steps. We need to check this condition for all triples (i, i_0, j). The total number of such pairs does not exceed $O(n^3)$. For each triple, we need a constant number of steps, so the overall complexity of this part of the algorithm is $O(n^3)$.

After that, we need n times to apply a linear time ($O(n)$) algorithm for computing strongly connected components. The overall complexity of this part is $n \cdot O(n) = O(n^2)$. Thus, the overall complexity of our algorithm is $O(n^3) + O(n^2) = O(n^3)$.

Comment. By definition, an algorithm is called feasible if its computation time on all inputs is bounded by some polynomial of the length of the input. According to this definition, any $O(n^3)$-time algorithm is feasible. Thus, we indeed have a feasible algorithm for finding appropriate variables, and for generating the corresponding functions $a(x_1, \ldots, x_k)$ and $F(a, x_{k+1}, \ldots, x_n)$. In other words, we have a feasible algorithm for deciding when we can reduce a multi-variable range estimation problem to two fewer-variable problems.

Example. Let us consider the example similar to the one we had before, with $f(x_1, x_2, x_3) = x_1 \cdot x_2 + x_1 \cdot x_3$. In this example, $\dfrac{\partial f}{\partial x_1} = x_2 + x_3$, $\dfrac{\partial f}{\partial x_2} = x_1$, and $\dfrac{\partial f}{\partial x_3} = x_1$.

We start with $i_0 = 1$. In this case, we have $d(2, 1, 3) \neq 0$ and $d(3, 1, 2) \neq 0$, so we have a graph with vertices 2 and 3 for which there is an edge from 2 to 3 and an edge from 3 to 2. This graphs is clearly strongly connected.

Next, we take $i_0 = 2$. In this case, we have $d(1, 2, 3) \neq 0$ and $d(3, 2, 1) = 0$. Thus, we have a graph with vertices 1 and 3, there is an edge from 1 to 3, but there is no edge from 3 to 1. This graph has two strongly connected components: $\{1\}$ and $\{3\}$. Out of these components, the component $\{1\}$, has an edge going out. The component $C = \{3\}$ has no edge going out, so we take $I = C \cup \{2\} = \{2, 3\}$.

What follows is similar to the example from the previous section: by selecting $v_i = 1$, we find $a(x_2, x_3) = x_2 + x_3$, and $F(a, x_1) = a \cdot x_1$.

4 Future Work

Towards a software implementation of the above algorithm. At present, we only have an algorithm. Our examples indicate that it is desirable to incorporate this algorithm – or, if possible, its improved version – into interval computations software. As a start, it may be desirable to first produce a separate implementation of this algorithm, as a pre-processing tool for transforming a multi-variable range estimation problem to two or more fewer-variables problems (whenever such a transformation is possible).

Once such a software implementation is in place, it is necessary to test it (and thus, to test our algorithm) on problems which are more sophisticated (and more realistic) than our simple examples.

Towards more efficient algorithms. In this paper, we gauge every node in the corresponding computational graph only from the viewpoint of how many input variable it takes. It is desirable to also take into account what function is computed on each of these nodes. For some practical problems, many nodes of the computational graph represent monotonic or convex functions, which admit tight bounds with modest effort. It is desirable to take this information into account, to make the corresponding reduction to fewer-variables problems more efficient.

This information will hopefully be useful, since experience has shown that often, even modest effort at tightening bounds of sub-expression yields impactful improvements in bounds higher in the expression tree[1].

Acknowledgments. This work was supported in part by the National Science Foundation grants HRD-0734825 and HRD-1242122 (Cyber-ShARE Center of Excellence) and DUE-0926721.

The authors are greatly thankful to the anonymous referees for valuable suggestions.

References

[1] Cormen, Th.H., Leiserson, C.E., Rivest, R.L., and Stein, C.: *Introduction to Algorithms*, MIT Press, Cambridge, MA, 2009.

[2] Gaganov, A.A.: *Computational complexity of the range of the polynomial in several variables*, Leningrad University, Math. Department, M.S. Thesis, 1981 (in Russian).

[3] Gaganov, A.A.: *Computational complexity of the range of the polynomial in several variables*, Cybernetics, pp. 418–421, 1985.

[4] Griewank, A., and Walther, A.: *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, SIMA, Philadelphia, Pennsylvania, 2008.

[5] Hansen, E.: *Sharpness in interval computations*, Reliable Computing 3, pp. 7–29, 1997.

[6] Gebremedhin, A.H., Manne, F., and Pothen, A.: *What color is your Jacobian? Graph coloring for computing derivatives*, SIAM Review 47(4), pp. 629–705, 2005.

[7] Jaulin, L., Kieffer, M., Didrit, O., and Walter, E.: *Applied Interval Analysis*, Springer Verlag, London, 2001.

[1] We are thankful to the anonymous referee for these useful suggestions.

[8] Kreinovich, V., Lakeyev, A., Rohn, J., and Kahl, P.: *Computational Complexity and Feasibility of Data Processing and Interval Computations*, Kluwer, Dordrecht, 1998.

[9] Lorkowski, J.: *From single to double use expressions, with applications to parametric interval linear systems: on computational complexity of fuzzy and interval computations*, Proceedings of the 30th Annual Conference of the North American Fuzzy Information Processing Society NAFIPS'2011, El Paso, Texas, March 18–20, 2011.

[10] Moore, R.E., Kearfott, R.B., and Cloud, M.J.: *Introduction to Interval Analysis*. SIAM Press, Philadelphia, Pennsylviania, 2009.

[11] Rabinovich, S.G.: *Measurement Errors and Uncertainty: Theory and Practice*, Springer Verlag, Berlin, 2005.