Finding All Solution Sets of Piecewise-Linear Interval Equations Using Integer Programming*

Kiyotaka Yamamura and Suguru Ishiguro
Department of Electrical, Electronic, and Communication Engineering, Chuo University, 1-13-27 Kasuga, Bunkyo-ku, Tokyo 112-8551, Japan
yamamura@elect.chuo-u.ac.jp

Abstract

This paper presents an efficient method for finding all solution sets of piecewise-linear interval equations using integer programming. In this method, the problem of finding all solution sets is formulated as a mixed integer programming problem, and it is solved by a high-performance integer programming solver such as CPLEX. It is shown that the proposed method can be implemented easily without writing complicated programs, and that all solution sets are obtained by solving mixed integer programming problems several times. Numerical examples are given to confirm the effectiveness of the proposed method.

Keywords: nonlinear equation, finding all solutions, integer programming
AMS subject classifications: 65H10, 65G40, 90C10

1 Introduction

The concept of set-valued function is useful in problems such as numerical computation with guaranteed accuracy and in tolerance analysis of electronic circuits. We consider systems of nonlinear equations whose nonlinear terms are described by set-valued functions as shown in Figure 1(a). Such functions are called piecewise-trapezoidal functions in [20], piecewise-linear enclosures of nonlinear functions in [17], and piecewise-linear interval functions in [8]. In this paper, we use the term *piecewise-linear interval (PLI) functions*. On each piece (interval of x_i), a PLI function is represented by a trapezoid with two parallel vertical sides, as shown in Figure 1(b) (or by a triangle if $b_{ij}^L = b_{ij}^U$). For simplicity, nonlinear equations containing PLI functions will be called PLI equations, and set-valued functions as represented in Figure 1(b) will be called trapezoidal functions. Since any set-valued functions of one variable can be approximated by a PLI function with many trapezoids, the concept of PLI functions is useful in many applications.

A system of PLI equations often has several unconnected sets of solutions, which will be called solution sets. In [20], an efficient algorithm is proposed for finding all

*Submitted: September 30, 2015; Revised: January 27, 2016; Accepted: February 8, 2016.

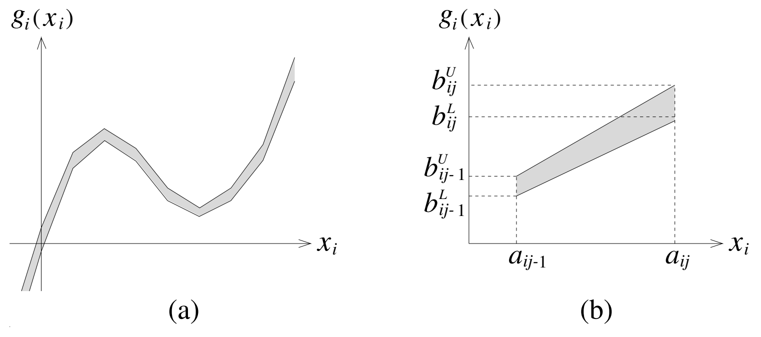


Figure 1: Piecewise-linear interval function.

solution sets of systems of PLI equations approximately using linear programming. Here, the term *approximately* means that small boxes which enclose the solution sets are obtained.[1] This algorithm works well in many cases. However, the implementation of this algorithm is generally difficult, at least for non-experts or beginners. To find all solution sets easily, it is necessary to develop a simple method which can be implemented easily without writing complex programs.

By the way, in the field of mathematical programming, the study of integer programming has made a rapid progress, and excellent commercial and non-commercial integer programming solvers such as CPLEX [5] and SCIP [15] have been developed. These solvers provide high-performance optimizers for solving very large, real-world mixed integer programming problems. The commercial software CPLEX also provides an academic free version.

Motivated by the remarkable progress of this field, we proposed an efficient method for finding all solutions of separable systems of piecewise-linear (PL) equations using integer programming [24]. In this method, the problem of finding all solutions is formulated as a mixed integer programming problem, and it is solved by a high-performance integer programming solver such as CPLEX. It was shown that the proposed method can be implemented easily without writing complicated programs, and that all solutions are obtained by solving a single mixed integer programming problem. It was also shown that the proposed method could find all solutions of systems of PL equations in several hundred variables in practical computation time.

In this paper, we propose an efficient and easily implementable method for finding all solution sets of systems of PLI equations by extending the method proposed in [24]. It is shown that all solution sets can be obtained by solving several mixed integer programming problems.

2 Basic Idea

In this section, we first summarize the basic idea proposed in [24]. As important books of this field, see, e.g., [1, 9, 12, 13, 14]. See also [2, 4, 7, 10, 11, 16, 17, 18] as related studies.

[1] Henceforth, we will omit the term *approximately*.

Consider the problem of finding all solutions to a system of n PL equations with a separable mapping:

$$F(x) = 0 \tag{1}$$

contained in a box $D = ([l_1, u_1], \ldots, [l_n, u_n])^T \subseteq \mathbb{R}^n$, where $x = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^n$ is a variable vector, and $F = (f_1, f_2, \ldots, f_n)^T : \mathbb{R}^n \to \mathbb{R}^n$ is a PL function which can be written as

$$F(x) = F^1(x_1) + F^2(x_2) + \cdots + F^n(x_n). \tag{2}$$

Note that any nonseparable function of many variables can be represented by a set of separable functions; i.e., additions of functions of one variable [19, 21]. In this paper, we assume that we have preliminarily applied the algorithm proposed in [19] and have made the system of equations to be solved separable.

For the simplicity of notation, and without loss of generality, in this paper we assume that (1) can be represented as

$$F(x) \overset{\triangle}{=} PG(x) + Qx + r = 0, \tag{3}$$

as assumed in [20, 21, 23, 24, 25], where $G(x) = [g_1(x_1), g_2(x_2), \ldots, g_n(x_n)]^T : \mathbb{R}^n \to \mathbb{R}^n$ is a PL function with component functions $g_i(x_i) : \mathbb{R}^1 \to \mathbb{R}^1$ $(i = 1, 2, \ldots, n)$, $P \in \mathbb{R}^{n\times n}$ and $Q \in \mathbb{R}^{n\times n}$ are constant matrices, and $r = (r_1, r_2, \ldots, r_n)^T \in \mathbb{R}^n$ is a constant vector. Namely, we divide the system into the nonlinear term $PG(x)$, the linear term Qx, and the constant term r.

Assume that we have chosen a partitioning $l_i = a_{i0} < a_{i1} < \cdots < a_{iK} = u_i$ of the interval $[l_i, u_i]$ and have defined a PL function $g_i(x_i)$ which is linear on the interval $[a_{ij-1}, a_{ij}]$ for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, K$ (see Figure 2). For simplicity of notation, we assume that the number of partitioning K is the same for all x_i-directions. We denote the function value at the points a_{ij} by

$$b_{ij} = g_i(a_{ij}), \qquad i = 1, 2, \ldots, n; \quad j = 0, 1, \ldots, K. \tag{4}$$

Introducing the auxiliary variables δ_{ij} $(i = 1, 2, \ldots, n;\ j = 1, 2, \ldots, K)$, every real x_i in $[a_{ij-1}, a_{ij}]$ can be written as

$$x_i = a_{ij-1} + \delta_{ij}, \tag{5}$$

where $0 \le \delta_{ij} \le a_{ij} - a_{ij-1}$, and the PL function $g_i(x_i)$ can be written as

$$g_i(x_i) = b_{ij-1} + \frac{b_{ij} - b_{ij-1}}{a_{ij} - a_{ij-1}} \delta_{ij} \tag{6}$$

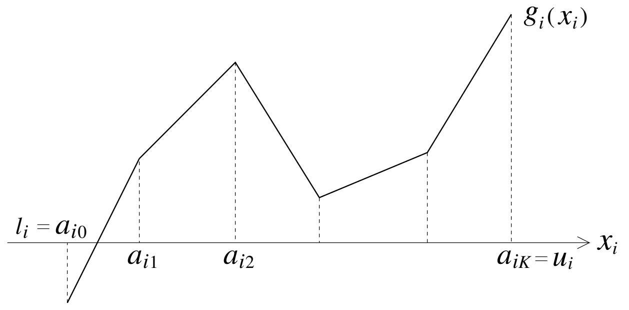


Figure 2: PL function.

on $[a_{ij-1}, a_{ij}]$. Extending these over the interval $[l_i, u_i]$, x_i and $g_i(x_i)$ can be expressed as

$$x_i = a_{i0} + \sum_{j=1}^{K} \delta_{ij}, \qquad i = 1, 2, \ldots, n \tag{7}$$

and

$$g_i(x_i) = b_{i0} + \sum_{j=1}^{K} \frac{b_{ij} - b_{ij-1}}{a_{ij} - a_{ij-1}} \delta_{ij}, \qquad i = 1, 2, \ldots, n, \tag{8}$$

respectively. Let $\Delta_{ij} \triangleq a_{ij} - a_{ij-1}$. In (7) and (8), it is necessary that if $0 < \delta_{ik} < \Delta_{ik}$ $(1 \leq k \leq K-1)$, then $\delta_{ij} = \Delta_{ij}$ for $1 \leq j \leq k-1$ and $\delta_{ij} = 0$ for $k+1 \leq j \leq K$. To formulate this in linear inequalities using integer variables, we introduce 0–1 variables μ_{ij} $(i = 1, 2, \ldots, n;\ j = 1, 2, \ldots, K-1)$ and consider the mixed 0–1 model:

$$\begin{array}{rcl} \Delta_{i1}\mu_{i1} \leq & \delta_{i1} & \leq \Delta_{i1}\mu_{i0} \\ & \vdots & \\ \Delta_{ij-1}\mu_{ij-1} \leq & \delta_{ij-1} & \leq \Delta_{ij-1}\mu_{ij-2} \\ \Delta_{ij}\mu_{ij} \leq & \delta_{ij} & \leq \Delta_{ij}\mu_{ij-1} \\ \Delta_{ij+1}\mu_{ij+1} \leq & \delta_{ij+1} & \leq \Delta_{ij+1}\mu_{ij} \\ & \vdots & \\ \Delta_{iK}\mu_{iK} \leq & \delta_{iK} & \leq \Delta_{iK}\mu_{iK-1} \\ & \mu_{i0} = 1, \quad \mu_{iK} = 0. & \end{array} \tag{9}$$

The correctness of (9) follows by induction of K. Moreover, by dividing the jth equation of (9) by Δ_{ij}, it is easily seen that μ_{ij} satisfies

$$1 = \mu_{i0} \geq \mu_{i1} \geq \mu_{i2} \geq \cdots \geq \mu_{iK-2} \geq \mu_{iK-1} \geq \mu_{iK} = 0, \qquad i = 1, 2, \ldots, n. \tag{10}$$

Thus, we can represent the PL function $G(x)$ by (7)–(9).

We further introduce auxiliary variables $y = (y_1, y_2, \ldots, y_n)^T \in \mathbb{R}^n$ and put $y_i = g_i(x_i)$. Then, Eq. (3) and $x \in D$ are represented by Eqs. (7) and (9), and

$$\begin{array}{l} Py + Qx + r = 0 \\ y_i = b_{i0} + \displaystyle\sum_{j=1}^{K} \frac{b_{ij} - b_{ij-1}}{a_{ij} - a_{ij-1}} \delta_{ij}, \qquad i = 1, 2, \ldots, n. \end{array} \tag{11}$$

Now we consider the mixed integer programming problem:

$$\begin{aligned}
&\max \ \text{(arbitrary constant)}\\
&\text{subject to}\\
&\quad Py + Qx + r = 0\\
&\quad x_i = a_{i0} + \sum_{j=1}^{K} \delta_{ij}\\
&\quad y_i = b_{i0} + \sum_{j=1}^{K} \frac{b_{ij} - b_{ij-1}}{a_{ij} - a_{ij-1}} \delta_{ij}\\
&\quad \Delta_{i1}\mu_{i1} \le \delta_{i1} \le \Delta_{i1}\\
&\qquad\qquad \vdots\\
&\quad \Delta_{ij-1}\mu_{ij-1} \le \delta_{ij-1} \le \Delta_{ij-1}\mu_{ij-2}\\
&\quad \Delta_{ij}\mu_{ij} \le \delta_{ij} \le \Delta_{ij}\mu_{ij-1}\\
&\quad \Delta_{ij+1}\mu_{ij+1} \le \delta_{ij+1} \le \Delta_{ij+1}\mu_{ij}\\
&\qquad\qquad \vdots\\
&\quad 0 \le \delta_{iK} \le \Delta_{iK}\mu_{iK-1}, \qquad i = 1, 2, \ldots, n,
\end{aligned} \tag{12}$$

where the objective function is an arbitrary constant and μ_{ij} ($i = 1, 2, \ldots, n$; $j = 1, 2, \ldots, K-1$) are 0–1 variables. Since the constraints of (12) are equivalent to (3) and $x \in D$, the feasible region of (12) is the set of all solutions of (3) in D. Hence, by solving (12), we can obtain a solution of (3) contained in D.

In [24], all solutions of (12) are found by applying integer programming solvers to (12). There are many software packages for solving mixed integer programming problems. Among them, IBM ILOG CPLEX [5] (often informally referred to simply as CPLEX) provides one of the most efficient solvers for mixed integer programming problems at the present time. In this paper, we use CPLEX for solving integer programming problems. One advantage of CPLEX is that it has the *solution pool* feature, which generates and stores multiple solutions to a mixed integer programming problem.[2] Using this feature, we can find all solutions of (3) by solving (12) only once.

3 Algorithm for the Solution Pool of CPLEX

The solution pool feature of CPLEX uses the algorithm proposed in [3] (see [5, p.283]). Algorithms 1–3 show the outlines of the standard branch-and-bound algorithm and the proposed algorithm (termed the one-tree algorithm) written in [3]. In this section, we summarize the one-tree algorithm. We will use the same symbols for variables and sets as those used in [3]. Some symbols are duplicated, but there is little confusion.

Given a mixed integer programming problem $P = \min_{x \in X} c^T x$, where $X = \{x \in \mathbb{R}^d : Ax \le b,\ x_i \in \mathbb{Z},\ \forall i \in I \subseteq \{1, 2, \ldots, d\}\}$, for which an optimal solution is x^*, let us consider a problem of generating p different feasible solutions $x(1), x(2), \ldots, x(p)$ for P within q% of the optimum, i.e., such that $c^T x(i) \le c^T x^* + q|c^T x^*|/100$ for $i = 1, 2, \ldots, p$.

The standard branch-and-bound algorithm for solving integer programming problems aims at progressively reducing the search space as quickly and as much as possible

[2] The non-commercial software SCIP also has a feature similar to the solution pool.

Algorithm 1 Outline of standard branch-and-bound algorithm

```
1: Preprocessing
2: Set of open nodes : $N_{\text{open}} \leftarrow \{rootnode\}$
3: Objective value of the incumbent: $z^* \leftarrow +\infty$
4: while $N_{\text{open}} \neq \emptyset$ do
5:    Choose a node $n$ from $N_{\text{open}}$
6:    Solve LP at node $n$. Solution is $x(n)$ with objective $z(n)$.
7:    if $z(n) \geq z^*$ then
8:       Fathom the node: $N_{\text{open}} \leftarrow N_{\text{open}} \setminus \{n\}$
9:    else
10:      if $x(n)$ is integer-valued then
11:         $x(n)$ becomes new incumbent: $x^* \leftarrow x(n)$; $z^* \leftarrow z(n)$
12:         Do reduced cost fixing
13:         Fathom the node: $N_{\text{open}} \leftarrow N_{\text{open}} \setminus \{n\}$
14:      else
15:         Choose branching variable $i$ such that $x_i(n)$ is fractional
16:         Build children nodes $n_1 = n \bigcap \{x_i \leq \lfloor x_i(n) \rfloor\}$ and
            $n_2 = n \bigcap \{x_i \geq \lfloor x_i(n) \rfloor + 1\}$
17:         $N_{\text{open}} \leftarrow N_{\text{open}} \bigcup \{n_1, n_2\} \setminus \{n\}$
18:      end if
19:   end if
20: end while
```

so that it is easier both to find the optimal solution and to prove that it is optimal. However, when the aim is to generate multiple solutions, the perspective needs to be different: if the search space is reduced too much, it will not contain enough solutions. The one-tree algorithm is adapted from the standard branch-and-bound algorithm (outlined in Algorithm 1) for this purpose. It proceeds in two phases. During the first phase (outlined in Algorithm 2), the branch-and-bound tree is constructed and explored to find the optimal solution, and its nodes are kept for the second phase. During the second phase (outlined in Algorithm 3), the tree built in the first phase is reused and explored in a different way to yield multiple solutions. The differences with the standard branch-and-bound algorithm relate to storing integer solutions, fathoming nodes, and branching.

In standard branch-and-bound, an integer solution is stored only if it improves on the incumbent. When generating solutions in the second phase, we store in the set S all integer solutions which are within $q\%$ of the optimum value. In standard branch-and-bound, a node is fathomed when the sub-problem it defines cannot yield any improving integer solution, i.e., when its LP solution is integer-valued or has an objective value worse than the incumbent. In the first phase of the one-tree algorithm, nodes are fathomed by the same criterion, but instead of being discarded, they are stored for further examination during the second phase (see lines 9 and 13 of Algorithm 2). During the second phase, a node is fathomed if it cannot yield any additional integer solution within $q\%$ of the optimum value, i.e., if its LP solution is integer-valued and all integer variables have been fixed by the local bounds of the node, or if the objective value of its LP solution is strictly more than $q\%$ worse than the optimum value (see

Algorithm 2 Outline of one-tree algorithm: phase I

```
1: Preprocessing with only primal reductions
2: Set of open nodes : $N_{\text{open}} \leftarrow \{rootnode\}$
3: Set of stored nodes : $N_{\text{stored}} \leftarrow \emptyset$
4: Objective value of the incumbent: $z^* \leftarrow +\infty$
5: while $N_{\text{open}} \neq \emptyset$ do
6:    Choose a node $n$ from $N_{\text{open}}$
7:    Solve LP at node $n$. Solution is $x(n)$ with objective $z(n)$.
8:    if $z(n) \geq z^*$ then
9:       Fathom the node and keep it for phase II: $N_{\text{open}} \leftarrow N_{\text{open}} \setminus \{n\}$;
         $N_{\text{stored}} \leftarrow N_{\text{stored}} \bigcup \{n\}$
10:   else
11:      if $x(n)$ is integer-valued then
12:         $x(n)$ becomes new incumbent: $x^* \leftarrow x(n)$; $z^* \leftarrow z(n)$
13:         Fathom the node and keep it for phase II: $N_{\text{open}} \leftarrow N_{\text{open}} \setminus \{n\}$;
            $N_{\text{stored}} \leftarrow N_{\text{stored}} \bigcup \{n\}$
14:      else
15:         Choose branching variable $i$ such that $x_i(n)$ is fractional
16:         Build children nodes $n_1 = n \bigcap \{x_i \leq \lfloor x_i(n) \rfloor\}$ and
            $n_2 = n \bigcap \{x_i \geq \lfloor x_i(n) \rfloor + 1\}$
17:         $N_{\text{open}} \leftarrow N_{\text{open}} \bigcup \{n_1, n_2\} \setminus \{n\}$
18:      end if
19:   end if
20: end while
```

Algorithm 3 Outline of one-tree algorithm: phase II

```
1: Reuse tree from phase I: $N_{\text{open}} \leftarrow N_{\text{stored}}$
2: Reuse incumbent from phase I: Set of solutions: $S \leftarrow \{x^*\}$
3: while $N_{\text{open}} \neq \emptyset$ do
4:    Choose a node $n$ from $N_{\text{open}}$
5:    Solve LP at node $n$. Solution is $x(n)$ with objective $z(n)$.
6:    if $z(n) > z^* + q|z^*|/100$ then
7:       Fathom the node: $N_{\text{open}} \leftarrow N_{\text{open}} \setminus \{n\}$
8:    else
9:       if $x(n)$ is integer-valued then
10:         $x(n)$ is added to the pool of solutions if it is not a duplicate:
            if $x(n) \notin S$, then $S \leftarrow S \bigcup \{x(n)\}$
11:      end if
12:      Choose branching variable $i$ such that it is not fixed by the local bounds
         of node $n$: $lb_i(n) < ub_i(n)$
13:      Build children nodes $n_1 = n \bigcap \{x_i \leq \lfloor x_i(n) \rfloor\}$ and
         $n_2 = n \bigcap \{x_i \geq \lfloor x_i(n) \rfloor + 1\}$
14:      $N_{\text{open}} \leftarrow N_{\text{open}} \bigcup \{n_1, n_2\} \setminus \{n\}$
15:   end if
16: end while
```

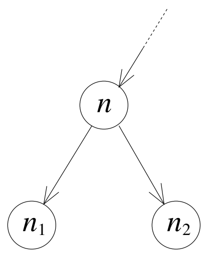


Figure 3: A part of the search tree where the nodes n_1 and n_2 correspond to two adjacent trapezoidal boxes, at least one of which contains the solution set.

lines 6 and 7 of Algorithm 3).

In standard branch-and-bound, only variables which are fractional in the LP solution at the node are branched on, but in the second phase of the one-tree algorithm, we also branch on variables which are integral in the LP solution at the node if they are not fixed by local bounds (see line 12 of Algorithm 3). The one-tree algorithm stops in both phases when the set of open nodes N_{open} becomes empty (or because other stopping criteria intervene).

In CPLEX, the relative tolerance on the objective value (q on line 6 of Algorithm 3) is determined by the solution pool relative gap parameter `SolnPoolGap` [6, p.139]. Solutions which are worse than the objective value of the incumbent by this measure are not kept in the solution pool. For example, if we set this parameter to 0.01, then solutions worse than the incumbent by 1% or more will be discarded. Hence, we can collect solutions within a given percentage of the optimal solution.

In the general usage of CPLEX, `SolnPoolGap` is set to a small value so that we can find near-optimal solutions. If we set `SolnPoolGap` to a very large value, then a feasible node is not discarded in both phases because the condition on line 6 of Algorithm 3 is always false. In this paper, we use this property.

4 Finding All Solution Sets of PLI Equations Using Integer Programming

In this section, we propose an efficient method for finding all solution sets of a system of PLI equations:

$$F(x) \overset{\triangle}{=} PG(x) + Qx + r = 0 \tag{13}$$

contained in a box $D = ([l_1, u_1], \ldots, [l_n, u_n])^T \subset \mathbb{R}^n$, where $x = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^n$ is a variable vector, $r = (r_1, r_2, \ldots, r_n)^T \in \mathbb{R}^n$ is a constant vector, $P \in \mathbb{R}^{n\times n}$ and $Q \in \mathbb{R}^{n\times n}$ are constant matrices, and $G(x) = [g_1(x_1), g_2(x_2), \ldots, g_n(x_n)]^T$ is a PLI function. For simplicity, a box on which $F(x)$ becomes a trapezoidal function will be called a trapezoidal box. Note that there is a one-to-one correspondence between a trapezoidal box and a set of 0–1 variables μ_{ij} ($i = 1, 2, \ldots, n$; $j = 1, 2, \ldots, K-1$) which satisfies (10).

Let the plane where we want to draw the solution sets be the (x_p, x_q)-plane. In the proposed method, we consider the mixed integer programming problems:

$$
\begin{aligned}
&\max/\min\ x_p \quad \text{and} \quad \max/\min\ x_q\\
&\text{subject to}\\
&\quad Py + Qx + r = 0\\
&\quad x_i = a_{i0} + \sum_{j=1}^{K} \delta_{ij}\\
&\quad y_i \geq b_{i0}^{L} + \sum_{j=1}^{K} \frac{b_{ij}^{L} - b_{ij-1}^{L}}{a_{ij} - a_{ij-1}} \delta_{ij}\\
&\quad y_i \leq b_{i0}^{U} + \sum_{j=1}^{K} \frac{b_{ij}^{U} - b_{ij-1}^{U}}{a_{ij} - a_{ij-1}} \delta_{ij}\\
&\quad \Delta_{i1}\mu_{i1} \leq \delta_{i1} \leq \Delta_{i1}\\
&\qquad\quad \vdots\\
&\quad \Delta_{ij-1}\mu_{ij-1} \leq \delta_{ij-1} \leq \Delta_{ij-1}\mu_{ij-2}\\
&\quad \Delta_{ij}\mu_{ij} \leq \delta_{ij} \leq \Delta_{ij}\mu_{ij-1}\\
&\quad \Delta_{ij+1}\mu_{ij+1} \leq \delta_{ij+1} \leq \Delta_{ij+1}\mu_{ij}\\
&\qquad\quad \vdots\\
&\quad 0 \leq \delta_{iK} \leq \Delta_{iK}\mu_{iK-1}, \qquad i = 1, 2, \ldots, n,
\end{aligned}
\tag{14}
$$

where the superscripts L and U denote the lower bounds and the upper bounds of the trapezoidal functions as shown in Figure 1(b). In (14), we maximize and minimize x_p and x_q. It is seen easily that the constraints of (14) are equivalent to (13) and $x \in D$. Then, we solve (14) by CPLEX using the solution pool feature, where we set the solution pool relative gap parameter `SolnPoolGap` to `default` (1.0e+75). We also set the solution pool intensity parameter `SolnPoolIntensity` to 4 (highest value), the populate limit parameter `PopulateLim` (which denotes the maximum number of solutions generated for the solution pool) to a sufficiently large value (for example, 2 100 000 000 [5, p.289]), and other parameters to `default`. Then, we call `populate`.

If we apply CPLEX to (14) using the above parameter values, then eventually the LP relaxations of (14) are solved on all trapezoidal boxes containing the solution sets. Suppose that Figure 3 shows a part of the search tree where n_1 and n_2 correspond to two adjacent trapezoidal boxes (i.e., all 0–1 variables are fixed and only one 0–1 variable has a different value at the nodes n_1 and n_2), at least one of which contains the solution set, and n is a feasible node corresponding to a union of the two trapezoidal boxes. As noted in Section 3, a feasible node where at least one integer variable has not been fixed yet is not fathomed in Algorithm 3 if `SolnPoolGap` is set to a very large value such as `default` (1.0e+75). Since the ancestor nodes of n_1 and n_2 are feasible, they are not fathomed. Hence, we reach the node n.

Then, n is branched on and its children nodes n_1 and n_2 are created on line 13 of Algorithm 3. These nodes are added to the set N_{open} on line 14, taken out from N_{open} on line 4, and the LP relaxations are solved at the nodes n_1 and n_2 on line 5. If the node is feasible, then the solution is added to the pool of solutions on line 10. Thus, eventually the LP relaxations of (14) are solved on all trapezoidal boxes containing the solution sets, and the solutions are stored in the set S.

On a trapezoidal box, (14) becomes a linear programming problem. Therefore,

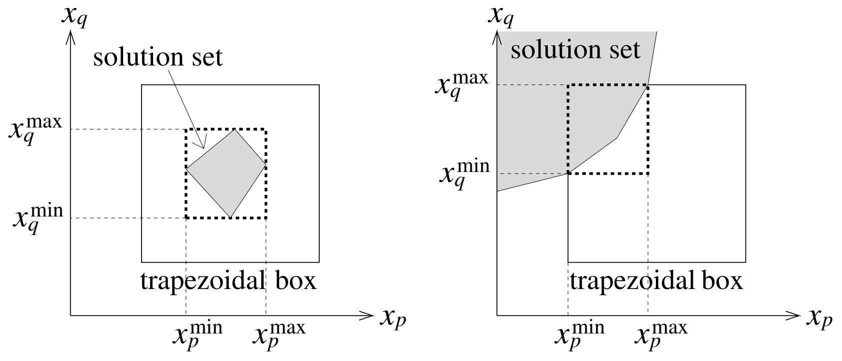


Figure 4: The smallest boxes containing the solution sets.

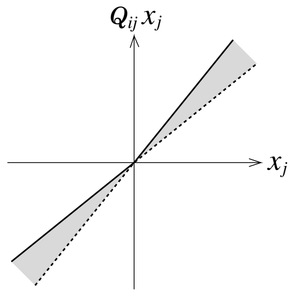


Figure 5: Set-valued function $[\underline{Q}_{ij}, \overline{Q}_{ij}]x_j$.

by solving (14), we can obtain the smallest box containing the solution set for each trapezoidal box as shown in Figure 4 by dashed lines, together with the values of μ_{ij}'s which specify the trapezoidal box. Thus, we can find all solution sets of the system of PLI equations (13).

If we want to see the solution sets from various directions, then we change the objective function of (14) to

$$\max/\min\ x_1, \quad \max/\min\ x_2,\ \ldots,\ \max/\min\ x_n.$$

Namely, we solve the mixed integer programming problems $2n$ times. Then, we can draw the solution sets projected onto any (x_i, x_j)-plane (see Example 4 of Section 5).[3]

The proposed method can be extended to the case where the components Q_{ij} of the matrix Q are given by intervals $[\underline{Q}_{ij}, \overline{Q}_{ij}]$, and therefore each component $Q_{ij}x_j$ of the linear term Qx is represented by a set-valued function $[\underline{Q}_{ij}, \overline{Q}_{ij}]x_j$. In this case, we consider $[\underline{Q}_{ij}, \overline{Q}_{ij}]x_j$ to be a kind of a PLI function which is defined by two PL functions as represented in Figure 5 by solid lines and dashed lines.

[3] This can also be done by solving the mixed integer programming problem only once (by which all trapezoidal boxes containing the solution sets are obtained) and then by solving linear programming problems $2n$ times on all of the trapezoidal boxes (see [20, p.244]).

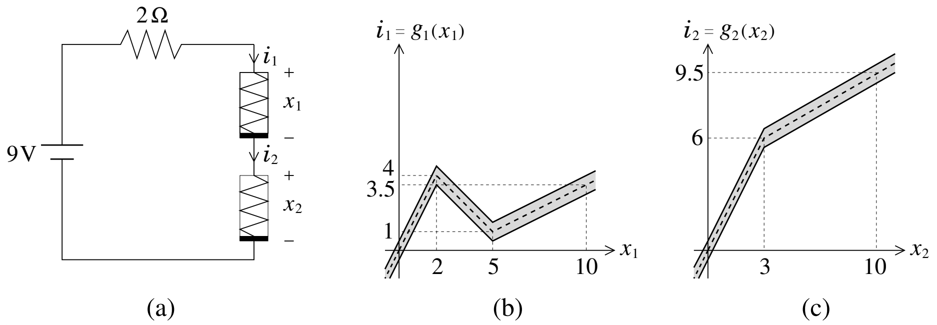


Figure 6: Circuit studied in Example 1 and the voltage-current characteristics of the PL resistors.

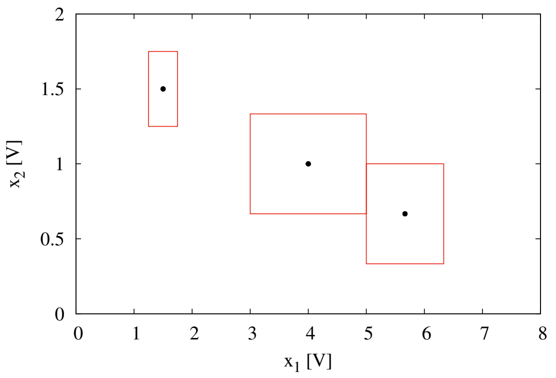


Figure 7: Solution sets (Example 1).

5 Numerical Examples

We implemented the proposed method using CPLEX 12.6.2 on a Hewlett-Packard Z820 (CPU: Intel Xeon Processor E5-2697 v2 2.70GHz). In this section, we show some numerical examples. We mainly consider the problems of the tolerance analysis of electronic circuits, where the characteristics of electronic devices fluctuate because of aging, environmental effects such as temperature, and manufacturing variations.

5.1 Example 1

We first consider the PL resistive circuit shown in Figure 6(a) [24], which is described by a system of PL equations:

$$2g_1(x_1) + x_1 + x_2 - 9 = 0$$
$$2g_2(x_2) + x_1 + x_2 - 9 = 0,$$

where $g_1(x_1)$ and $g_2(x_2)$ are PL functions as shown in Figures 6(b) and 6(c) by dashed lines, respectively. The initial box we consider is $D = ([0, 10], [0, 10])^T$. This system

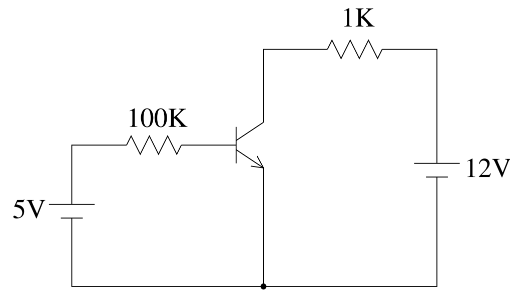


Figure 8: Transistor circuit (Example 2).

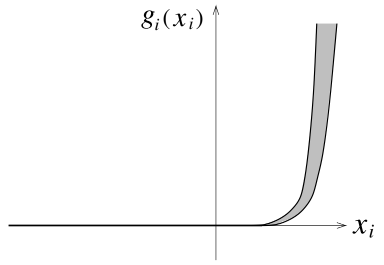


Figure 9: Set-valued function obtained by temperature fluctuation (Example 2).

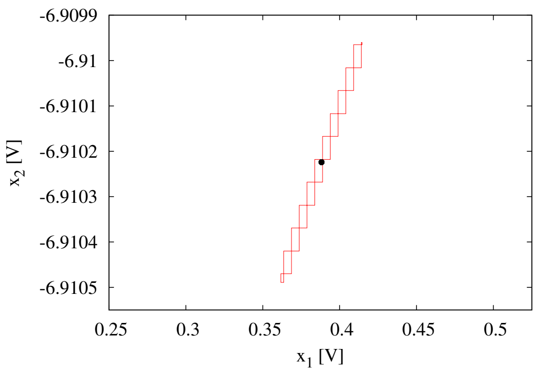


Figure 10: Solution set (Example 2).

of PL equations has three solutions within the box. We moved the PL functions up and down with distance 0.5 to obtain PLI functions as shown in Figures 6(b) and 6(c) by solid lines. We applied the proposed method to the system of PLI equations thus obtained. Figure 7 shows the result of computation, where the smallest boxes containing the solution sets are shown by solid lines, and the solutions of the original system of PL equations are shown by dots. The CPU time was 0.03 s.

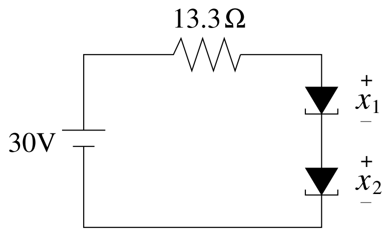


Figure 11: Two-tunnel diode circuit (Example 3).

The LP file of the mixed integer programming problem which was solved in this example is shown in the Appendix.

5.2 Example 2

We next consider the transistor circuit shown in Figure 8, described by

$$\begin{bmatrix} g_1(x_1) \\ g_2(x_2) \end{bmatrix} + \begin{bmatrix} 0.1010 \times 10^{-2} & -0.9901 \times 10^{-3} \\ -0.1980 \times 10^{-6} & 0.1980 \times 10^{-4} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 0.1198 \times 10^{-1} \\ -0.1369 \times 10^{-3} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

where x_1 and x_2 denote the base-emitter voltage and the base-collector voltage of the transistor, respectively, $g_i(x_i) \overset{\triangle}{=} 10^{-9}[\exp(qx_i/kT) - 1]$, $q = 1.602 \times 10^{-19}$, $k = 1.381 \times 10^{-23}$, and T denotes the temperature. The solution of this circuit when $T = 20\,^\circ\mathrm{C}$ is $(0.388232, -6.910224)^T$.

We changed the temperature as $T = 20 \pm 20\,^\circ\mathrm{C}$ and approximated the set-valued function thus obtained (see Figure 9) by a PLI function with 100 trapezoids. We applied the proposed method to the system of PLI equations using the initial box $D = ([-10, 0.5], [-10, 0.5])^T$. Figure 10 shows the result of computation, where the smallest boxes containing the solution set are shown by solid lines, and the solution of the original transistor circuit when $T = 20\,^\circ\mathrm{C}$ is shown by a dot. The CPU time was 0.04 s.

5.3 Example 3

Consider the nonlinear resistive circuit containing two tunnel diodes shown in Figure 11, described by a system of two nonlinear equations [20]. The characteristics of the tunnel diodes are given by

$$g_1(x_1) = 2.5x_1^3 - 10.5x_1^2 + 11.8x_1$$
$$g_2(x_2) = 0.43x_2^3 - 2.69x_2^2 + 4.56x_2.$$

We considered the initial box $D = ([-1, 4], [-1, 4])^T$, and approximated the nonlinear functions by PL functions on $[-1, 4]$, which are linear on K equally spaced intervals. Then, we moved them up and down with distance $w/2$ to obtain PLI functions as shown in Figure 1(a) of width w. We applied the proposed method to the system of PLI equations thus obtained. Figures 12(a)–(d) show the smallest boxes containing

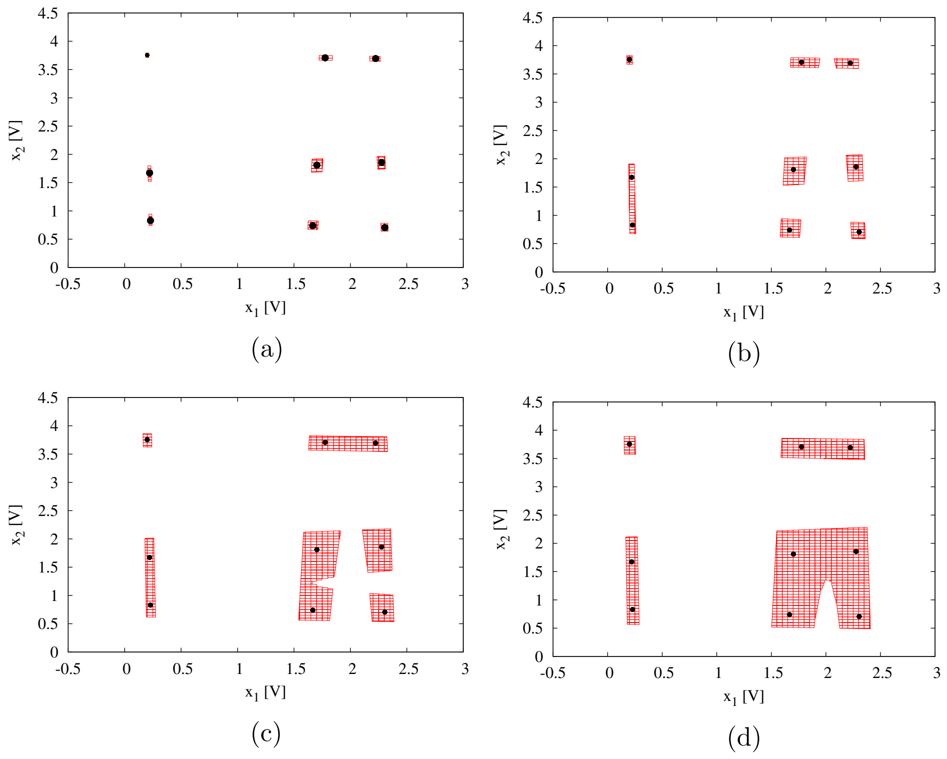


Figure 12: Solution sets (Example 3).

the solution sets when $K = 100$ and $w = 0.2$, 0.4, 0.6, and 0.8, respectively. The original system of PL equations has nine solutions, which are shown in Figure 12 by dots. From these figures, it is seen that the number of connected solution sets decreases as w increases. The total CPU time was 1.2 s.

In [20], the same problem is solved by the algorithm proposed there which uses linear programming. To verify that the proposed method certainly gives all solution sets, we compared the result with that in [20]. Then, we could confirm that the same result was obtained, i.e., the number of narrowed trapezoidal boxes in Figure 12 and the number of trapezoidal boxes in Figure 4 of [20] are the same.

5.4 Example 4

Consider the nonlinear resistive circuit containing three tunnel diodes shown in Figure 13, described by a system of three nonlinear equations. The characteristics of the tunnel diodes are given by

$$\begin{aligned} g_1(x_1) &= 0.43x_1^3 - 2.69x_1^2 + 4.56x_1 \\ g_2(x_2) &= 2.5x_2^3 - 10.5x_2^2 + 11.8x_2 \\ g_3(x_3) &= 1.3x_3^3 - 5.4x_3^2 + 6.9x_3. \end{aligned}$$

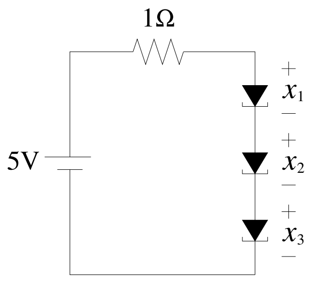


Figure 13: Three-tunnel diode circuit (Example 4).

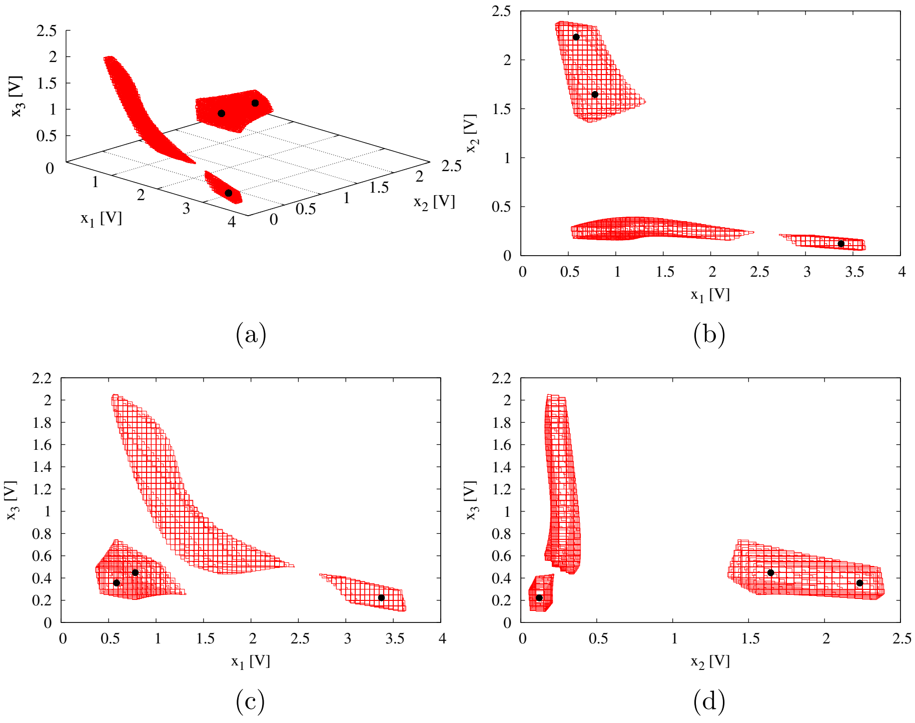


Figure 14: Solution sets (Example 4).

The initial box is $D = ([-1,4],[-1,4],[-1,4])^T$. The original circuit has three solutions within the box. We considered the PLI functions obtained in the same manner as in Example 3. In this example, we also changed the value of the linear terms $\pm d/2$ to obtain a set-valued function as shown in Figure 5. We applied the proposed method to the system of PLI equations thus obtained using $K = 100$, $w = 0.6$, and $d = 0.1$. Figure 14(a) shows the solution sets in the (x_1, x_2, x_3)-space. This figure can be obtained by maximizing and minimizing x_1, x_2, and x_3; i.e., it can be obtained by solving the mixed integer programming problems six times. If we enlarge this figure,

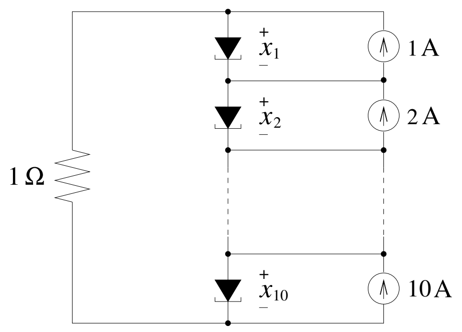
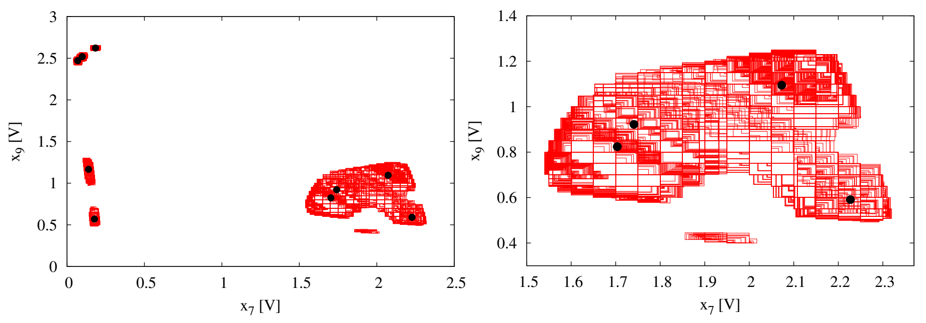


Figure 15: Ten-tunnel diode circuit (Example 5).



(a) (b)

Figure 16: Solution sets (Example 5).

then we can see many three-dimensional boxes which enclose the solution sets. The number of boxes which enclose the solution sets is 2 579, and the CPU time was 2 s. Figures 14(b)–(d) show the solution sets projected onto the (x_1, x_2), (x_1, x_3), and (x_2, x_3)-planes, respectively. It is seen that the shape of the solution sets varies largely according to the direction we are looking.

5.5 Example 5

Consider the nonlinear resistive circuit containing ten tunnel diodes shown in Figure 15 [20, 21, 23, 24, 25], described by a system of ten nonlinear equations:

$$g(x_i) + x_1 + x_2 + \cdots + x_{10} - i = 0, \qquad i = 1, 2, \ldots, 10,$$

where

$$g(x_i) = 2.5x_i^3 - 10.5x_i^2 + 11.8x_i.$$

We consider the initial box $([-1, 4], \ldots, [-1, 4])^T$. The original circuit has nine solutions within the box. We considered the PLI functions obtained in the same manner as in Example 3, and applied the proposed method to the system of PLI equations using $K = 100$ and $w = 0.4$. In this example, the mixed integer programming problems to

Table 1: Result of computation (Example 6).

n	Number of boxes	CPU time (s)
10	5	0.4
20	6	0.9
30	12	2
40	19	3
50	29	4
60	40	7
70	54	9
80	69	13
90	88	17
100	107	26
$\vdots$	$\vdots$	$\vdots$
200	477	349
300	1 019	2 111
400	1 567	14 918
500	2 229	86 496

be solved consist of 2 010 variables including 990 0–1 variables and 2 020 constraints. Figure 16(a) shows the solution sets projected onto the (x_7, x_9)-plane. The number of boxes which enclose the solution sets is 16 272, and the CPU time was 60 s. Figure 16(b) shows the enlarged view of the solution sets in the lower right part of Figure 16(a). It is seen that intricately-shaped solution sets are obtained by the proposed method.

5.6 Example 6

Finally, consider a system of n nonlinear equations [20, 21, 23, 24, 25]

$$x_i - \frac{1}{2n}\left(\sum_{j=1}^{n} x_j^3 + i\right) = 0, \qquad i = 1, 2, \ldots, n.$$

We considered the initial box $([-2.5, 2.5], \ldots, [-2.5, 2.5])^T$. This system has three solutions within the box. We considered the PLI functions obtained in the same manner as in Example 3, and applied the proposed method to the system of PLI equations using $K = 10$ and $w = 0.002$. Table 1 shows the number of boxes which enclose the solution sets and the CPU time when we maximized and minimized two variables (i.e., when we solved the mixed integer programming problems four times) for each n. It is seen that a system of 100 PLI equations could be solved in 26 s, and a system of 500 PLI equations could be solved in about 24 h. In this example, the mixed integer programming problems when $n = 500$ consist of 10 500 variables including 4500 0–1 variables and 11 000 constraints.

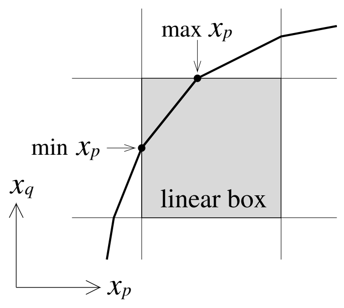


Figure 17: Relation between the solutions of (16) and solution curves.

6 Extension to Finding All Solution Curves

The proposed method can easily be extended to finding all solution curves. Consider the problem of finding all one-dimensional solution curves to a system of n PL equations in $n+1$ variables:

$$F(x) \overset{\triangle}{=} PG(x) + Qx + r = 0 \tag{15}$$

contained in a box $D = ([l_1, u_1], \ldots, [l_{n+1}, u_{n+1}])^T \subset \mathbb{R}^{n+1}$, where $x = (x_1, x_2, \ldots, x_{n+1})^T \in \mathbb{R}^{n+1}$ is a variable vector, $G(x) = [g_1(x_1), g_2(x_2), \ldots, g_{n+1}(x_{n+1})]^T : \mathbb{R}^{n+1} \to \mathbb{R}^{n+1}$ is a PL function with component functions $g_i(x_i) : \mathbb{R}^1 \to \mathbb{R}^1$ $(i = 1, 2, \ldots, n+1)$ given by Figure 2, $P \in \mathbb{R}^{n \times (n+1)}$ and $Q \in \mathbb{R}^{n \times (n+1)}$ are constant matrices, and $r = (r_1, r_2, \ldots, r_n)^T \in \mathbb{R}^n$ is a constant vector. In this section, we do not consider set-valued functions. For simplicity, a box on which $F(x)$ becomes a linear function will be called a linear box.

In this case, we consider the mixed integer programming problems:

$$\begin{aligned}
&\text{max/min } x_p \\
&\text{subject to} \\
&\quad Py + Qx + r = 0 \\
&\quad x_i = a_{i0} + \sum_{j=1}^{K} \delta_{ij} \\
&\quad y_i = b_{i0} + \sum_{j=1}^{K} \frac{b_{ij} - b_{ij-1}}{a_{ij} - a_{ij-1}} \delta_{ij} \\
&\quad \Delta_{i1}\mu_{i1} \le \delta_{i1} \le \Delta_{i1} \\
&\qquad\qquad \vdots \\
&\quad \Delta_{ij-1}\mu_{ij-1} \le \delta_{ij-1} \le \Delta_{ij-1}\mu_{ij-2} \\
&\quad \Delta_{ij}\mu_{ij} \le \delta_{ij} \le \Delta_{ij}\mu_{ij-1} \\
&\quad \Delta_{ij+1}\mu_{ij+1} \le \delta_{ij+1} \le \Delta_{ij+1}\mu_{ij} \\
&\qquad\qquad \vdots \\
&\quad 0 \le \delta_{iK} \le \Delta_{iK}\mu_{iK-1}, \qquad i = 1, 2, \ldots, n+1,
\end{aligned} \tag{16}$$

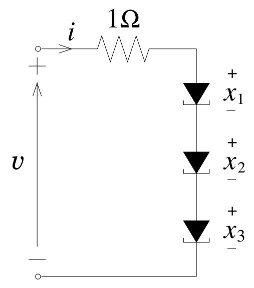


Figure 18: One-port tunnel diode circuit.

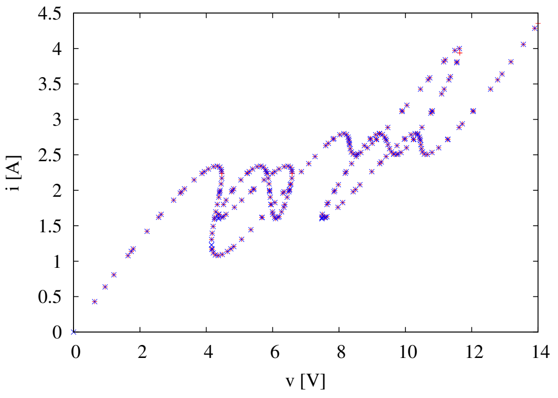


Figure 19: Extreme points of the solution curves for the circuit in Figure 18.

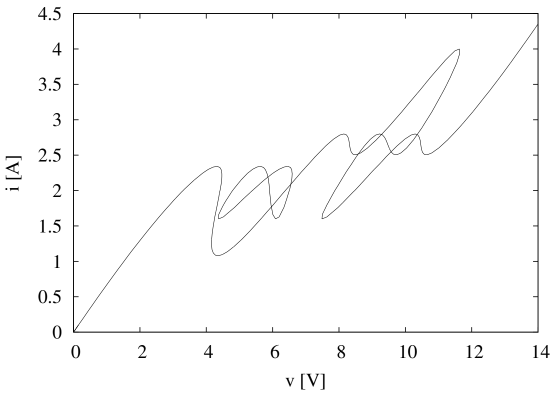


Figure 20: Solution curves for the circuit in Figure 18.

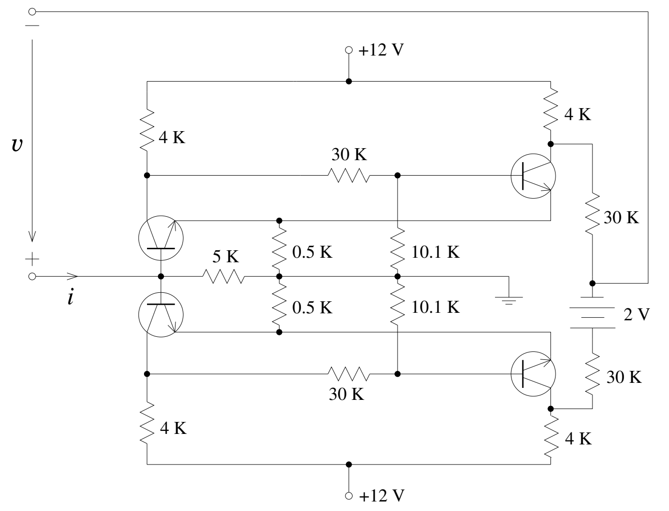


Figure 21: One-port transistor circuit.

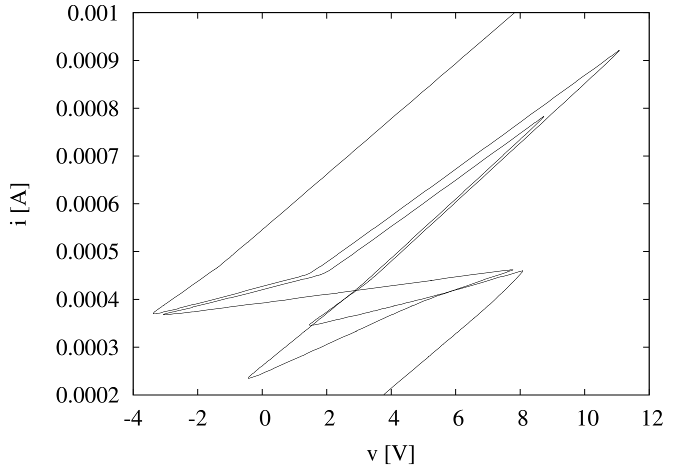


Figure 22: Solution curves for the circuit in Figure 21.

where we maximize and minimize an arbitrary variable x_p. Then, we solve (16) by CPLEX using the solution pool feature, where we set `SolnPoolGap` to `default`, `SolnPoolIntensity` to 4, `PopulateLim` to a sufficiently large value, and other parameters to `default`, and call `populate`. Then, eventually the LP relaxations of (16) are solved on all linear boxes containing the solution curves. On a linear box, (16) becomes a linear programming problem. Hence, by solving (16), we can obtain both extreme points of a segment which is a part of the solution curves as shown in Figure 17, together with the values of μ_{ij}'s which specify the linear box. By connecting the extreme points, we can find all solution curves.

We show two numerical examples. We first consider the one-port circuit shown in Figure 18, where the PL characteristics of the tunnel diodes are given as in Example 4 of Section 5. This circuit is described by a system of four PL equations in five variables (x_1, x_2, x_3, v, i). We applied the proposed method to this system using $K = 50$, and we obtained the extreme points as shown in Figure 19, where marks $+$ (red) denote the extreme points obtained by maximization, and marks $\times$ (blue) denote the extreme points obtained by minimization. By connecting the extreme points, we obtained the solution curves (called driving-point characteristic curves) as shown in Figure 20. The number of segments of the solution curves is 241, and the CPU time was 0.1 s.

We next consider the one-port transistor circuit shown in Figure 21, which is discussed in [22]. This circuit is described by a system of nine PL equations in ten variables. We applied the proposed method to this system using $K = 30$, and we obtained the solution curves as shown in Figure 22. The number of segments of the solution curves is 545, and the CPU time was 0.9 s.

In [22], the second problem is solved by the algorithm proposed there using linear programming. To verify that the proposed method certainly provides all solution curves, we compared the result with that in [22] and confirmed that the same result was obtained, i.e., the solution curves in Figure 22 and the solution curves in Figure 10 of [22] are the same.

7 Conclusion

In this paper, an efficient method has been proposed for finding all solution sets of systems of PLI equations using integer programming. It has been shown that the proposed method can be implemented easily without writing complicated programs, and that all solution sets are found by solving mixed integer programming problems several times. The proposed method will be useful in the analysis of perturbed systems which have many unconnected solution sets. It will also be useful in the numerical computation with guaranteed accuracy.

In the proposed method, our use of CPLEX is different from the general usage mainly in two points. First, we do not consider optimization but find all feasible solutions which are the solutions of the original continuous systems to be solved. Secondly, in the general usage of CPLEX, the solution pool relative gap parameter `SolnPoolGap` is set to a small value so that we can find near-optimal solutions, but in the proposed method, `SolnPoolGap` is set to a sufficiently large value so that we can find all solution sets. Such a use is enabled by the rapid progress of integer programming in the last decade. Since integer programming will continue to progress also in the future, it will be significant to study further how integer programming solvers can be used in the fields of interval analysis and reliable computing.

Acknowledgements

The authors would like to express their sincere gratitude to Emeritus Professor Hiroshi Konno of Tokyo Institute of Technology for valuable suggestions on integer programming. They are also grateful to the anonymous reviewer for helpful comments.

References

[1] G. Alefeld and J. Herzberger. *Introduction to Interval Computations.* Academic Press, New York, 1983.

[2] A. Baharev, L. Kolev, and E. Rév. Computing multiple steady states in homogeneous azeotropic and ideal two-product distillation. *AIChE Journal*, 57(6):1485–1495, 2011.

[3] E. Danna, M. Fenelon, Z. Gu, and R. Wunderling. Generating multiple solutions for mixed integer programming problems. In M. Fischetti and D. P. Williamson, editors, *Integer Programming and Combinatorial Optimization, 12th International IPCO Conference, Ithaca, NY, USA, June 25-27, 2007, Proceedings*, pages 280–294, Springer-Verlag, Berlin, Heidelberg, 2007.

[4] K. Georg. A new exclusion test. *J. Computational and Applied Mathematics*, 152(1-2):147–160, 2003.

[5] IBM. *IBM ILOG CPLEX Optimization Studio, CPLEX User's Manual, Version 12, Release 6.* `http://www.ibm.com/support/knowledgecenter/SSSA5P_12.6.3/ilog.odms.studio.help/pdf/usrcplex.pdf`.

[6] IBM. *IBM ILOG CPLEX Optimization Studio, CPLEX Parameters Reference, Version 12, Release 6.* `http://www.ibm.com/support/knowledgecenter/SSSA5P_12.6.3/ilog.odms.studio.help/pdf/paramcplex.pdf`.

[7] C. Jansson. Quasiconvex relaxations based on interval arithmetic. *Linear Algebra and its Applications*, 324(1-3):27–53, 2001.

[8] R. Kateja and G. Frehse. Representation of piecewise linear interval functions. Verimag Research Report TR-2012-16, 2012.

[9] R. B. Kearfott and V. Kreinovich, editors. *Applications of Interval Computations.* Applied Optimization, volume 3, Kluwer Academic Publishers, Dordrecht, 1996.

[10] L. V. Kolev. A new method for global solution of systems of non-linear equations. *Reliable Computing*, 4(2):125–146, 1998.

[11] Y. Lebbah, C. Michel, and M. Rueher. Efficient pruning technique based on linear relaxations. In C. Jermann, A. Neumaier, and D. Sam, editors, *Global Constrained Optimization and Constraint Satisfaction*, Lecture Notes in Computer Science, volume 3478, pages 1–14. Springer-Verlag, Berlin, Heidelberg, 2005.

[12] R. E. Moore. *Methods and Applications of Interval Analysis.* SIAM, Philadelphia, Pennsylvania, 1979.

[13] R. E. Moore, R. B. Kearfott, and M. J. Cloud. *Introduction to Interval Analysis.* SIAM, Philadelphia, Pennsylvania, 2009.

[14] A. Neumaier. *Interval Methods for Systems of Equations.* Encyclopedia of Mathematics and its Applications. Cambridge University Press, Cambridge, England, 1990.

[15] SCIP (Solving Constraint Integer Programs). `http://scip.zib.de/`.

[16] M. I. Syam. Nonlinear optimization exclusion tests for finding all solutions of nonlinear equations. *Applied Mathematics and Computation*, 170(2):1104–1116, 2005.

[17] Y. Wang and B. O. Nnaji. Solving interval constraints by linearization in computer-aided design. *Reliable Computing*, 13(2):211–244, 2007.

[18] M. A. Wolfe. On bounding solutions of underdetermined systems. *Reliable Computing*, 7(3):195–207, 2001.

[19] K. Yamamura. An algorithm for representing functions of many variables by superpositions of functions of one variable and addition. *IEEE Trans. Circuits and Systems-I*, 43(4):338–340, 1996.

[20] K. Yamamura. Finding all solution sets of piecewise-trapezoidal equations described by set-valued functions. *Reliable Computing*, 9(3):241–250, 2003.

[21] K. Yamamura, H. Kawata, and A. Tokue. Interval solution of nonlinear equations using linear programming. *BIT Numerical Mathematics*, 38(1):186–199, 1998.

[22] K. Yamamura and T. Ohshima. Finding all solutions of piecewise-linear resistive circuits using linear programming. *IEEE Trans. Circuits and Systems-I*, 45(4):434–445, 1998.

[23] K. Yamamura, K. Suda, and N. Tamura. LP narrowing: A new strategy for finding all solutions of nonlinear equations. *Applied Mathematics and Computation*, 215(1):405–413, 2009.

[24] K. Yamamura and N. Tamura. Finding all solutions of separable systems of piecewise-linear equations using integer programming. *J. Computational and Applied Mathematics*, 236(11):2844–2852, 2012.

[25] K. Yamamura and S. Tanaka. Finding all solutions of systems of nonlinear equations using the dual simplex method. *BIT Numerical Mathematics*, 42(1):214–230, 2002.

Appendix: LP File Used in Example 1

```
Maximize
 obj: x1
Subject To
 c1:  2 y1 + x1 + x2 = 9
 c2:  2 y2 + x1 + x2 = 9
 c3:  x1 - d11 - d12 - d13 = 0
 c4:  x2 - d21 - d22 = 0
 c5:  y1 - 2 d11 + d12 - 0.5 d13 >= -0.5
 c6:  y1 - 2 d11 + d12 - 0.5 d13 <= 0.5
 c7:  y2 - 2 d21 - 0.5 d22 >= -0.5
 c8:  y2 - 2 d21 - 0.5 d22 <= 0.5
 c9:  d11 - 2 u11 >= 0
 c10: d12 - 3 u12 >= 0
 c11: d12 - 3 u11 <= 0
 c12: d13 - 5 u12 <= 0
 c13: d21 - 3 u21 >= 0
 c14: d22 - 7 u21 <= 0
Bounds
 0 <= x1 <= 10
 0 <= x2 <= 10
 -0.5 <= y1 <= 4.5
 -0.5 <= y2 <= 10
 0 <= d11 <= 2
 0 <= d12 <= 3
 0 <= d13 <= 5
 0 <= d21 <= 3
 0 <= d22 <= 7
 0 <= u11 <= 1
 0 <= u12 <= 1
 0 <= u21 <= 1
Binaries
 u11  u12  u21
End
```