Annales Mathematicae et Informaticae
58 (2023) pp. 56–68
DOI: 10.33039/ami.2023.08.005
URL: https://ami.uni-eszterhazy.hu

The Fritz-John Condition System in Interval Branch and Bound method

Mihály Gencsi, Boglárka G.-Tóth

University of Szeged
{gencsi,boglarka}@inf.u-szeged.hu

Abstract. The Interval Branch and Bound (IBB) method is a good choice when a rigorous solution is required. This method handles computational errors in the calculations. Few IBB implementations use the Fritz-John (FJ) optimality condition to eliminate non-optimal boxes in a constrained nonlinear programming problem. Applying the FJ optimality condition implies solving an interval-valued system of equations. In the best case, the solution is an empty set if the interval box does not contain an optimizer point. Solving this system of equations is complicated or unsuccessful in many cases. This problem can be caused by the interval box being too wide, the defined system of equations containing unnecessary constraints, or the solver being unsuccessful. These unsuccessful attempts have a negative outcome and only increase the computation time. In this study, we propose some modifications to reduce the running time and computational requirements of the Interval Branch and Bound method.

Keywords: Global Optimization, Interval Arithmetic, Fritz-John condition, Branch and Bound method, Optimality condition

AMS Subject Classification: 90C26, 65G30, 90C30

1. Introduction

There are many applications in which we are looking for a rigorous solution to a mathematical problem. For example, in physics or chemistry, we look for the stability point of a substance. Sometimes we obtain a stability point, but in this environment, the material is very unstable; it can fall apart even with a small change.

In this study, we focus on solving the constrained nonlinear programming problems with inequality and general bound constraints. We deal with the following

Submitted: July 30, 2023
Accepted: August 7, 2023
Published online: August 8, 2023

n-dimensional nonlinear problem,

$$\begin{array}{ll} \underset{x \in \boldsymbol{y} \subseteq \mathbb{R}^n}{\text{minimize}} & f(x) \\ \text{subject to} & g_i(x) \leq 0, \quad i = 1, \ldots, m, \end{array} \tag{1.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $g_i : \mathbb{R}^n \to \mathbb{R}, i = 1, \ldots, m$ are continuously differentiable nonlinear functions, and the interval box $\boldsymbol{y} = [\underline{y}, \overline{y}]$ denotes a general bound constraint. We search for the global optimum using a guaranteed method, the Interval Branch and Bound (IBB) method. Solving a constrained nonlinear programming problem is, in general, very difficult. Sometimes we can only solve a low-dimensional instance or a smaller subproblem. In the IBB method, we replace the problem with smaller subproblems. We try to discard a subproblem by calculating upper and lower bounds and checking the feasibility or optimality. One of the best ways to rigorously compute the bounds for the subproblem is using interval arithmetic (IA). In IA, the rounding error or imprecision of the parameters is automatically taken into account by replacing the numbers with intervals. Today, several implementations of the IBB can be found in the literature. However, many of them do not use Fritz-John (FJ) or Karush-Kuhn-Tucker (KKT) optimality conditions to discard non-optimal subproblems. These mean solving an interval-valued system of equations. In this study, we use the improved version of IBB, which can solve both the unconstrained and the constrained cases in a reasonable time. We also study the solvability of the interval FJ optimality conditions, which are more general than the interval KKT optimality conditions.

In the following section, we introduce the basic terms and concepts of IA and the solution method for the interval-valued system of equations. We also describe the prototype of the IBB method. In Section 3, we study the FJ Condition System (FJ-CS) and extend it to intervals by defining the Normalized Interval Fritz-John Condition System (NIFJ-CS). In Section 4, we consider four methods to solve NIFJ-CS, analyze them, and present some additional improvements to reduce the running time of the methods. We compare the methods described using computational experiments in Section 5. Finally, in Section 6, we summarize this study and make suggestions for future directions.

2. Interval Branch and Bound method

In this section, we introduce the basic concepts of Interval Arithmetic (IA). We demonstrate the methods for solving interval-valued systems of equations, which we use to solve the NIFJ-CS. We also briefly introduce the prototype of the Interval Branch and Bound (IBB) method.

2.1. Interval arithmetic

Interval arithmetic is the basis of the Interval Branch and Bound method, in which numbers are replaced by a range of numbers called an *interval*. In this

way, rounding and measurement errors are avoided by enclosing the number in intervals. Following the basic notation used in the literature [8], the *intervals* are denoted by $\boldsymbol{x} = [\underline{x}, \overline{x}]$, where $\underline{x}$ and $\overline{x}$ describe the lower and upper bounds of the interval, respectively. Therefore, we can define n-dimensional interval vectors as $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \in \mathbb{I}^n = \mathbb{I} \times \cdots \times \mathbb{I}$, which can be called *intervalbox* or *box*, where $\mathbb{I}$ is the set of intervals. In addition, we define some properties of the intervals. For example, the *midpoint* of an interval $\boldsymbol{x}$ is denoted by $\mathrm{mid}(\boldsymbol{x}) = \frac{1}{2}(\overline{x} + \underline{x})$ or the *width* of the interval $\boldsymbol{x}$ by $\mathrm{wid}(\boldsymbol{x}) = \overline{x} - \underline{x}$. We can extend this to boxes as well, as follows. The midpoint of a n-dimensional box $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)^T$ is given by $\mathrm{mid}(\boldsymbol{x}) = (\mathrm{mid}(\boldsymbol{x}_1), \ldots, \mathrm{mid}(\boldsymbol{x}_n))^T$ and the width of the box by $\mathrm{wid}(\boldsymbol{x}) = \max\{\mathrm{wid}(\boldsymbol{x}_i) : i = 1, \ldots, n\}$.

Operations such as addition, multiplication, subtraction, and division can be extended to intervals. The *interval arithmetic operations* are defined by $\boldsymbol{x} \odot \boldsymbol{y} = \{x \odot y : x \in \boldsymbol{x}, y \in \boldsymbol{y}\}$ for $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{I}$, where $\odot \in \{+, -, \cdot, /\}$ and $\boldsymbol{x}/\boldsymbol{y}$ is defined only if $0 \notin \boldsymbol{y}$.

Furthermore, $\mathbf{f} : \mathbb{I}^n \to \mathbb{I}$ is an *inclusion function* for $f : \mathbb{R}^n \to \mathbb{R}$ if it satisfies $\{f(x) : x \in \boldsymbol{x}\} \subseteq \mathbf{f}(\boldsymbol{x})$ for all interval boxes $\boldsymbol{x} \subset \mathbb{I}^n$ within the domain of f. In many cases, the inclusion function is wider than the image of the function because it is overestimated. Note that if $\mathbf{f}$ is an inclusion function, we can obtain the lower and upper bounds on f taking $\underline{\mathbf{f}}(\boldsymbol{x})$ and $\overline{\mathbf{f}}(\boldsymbol{x})$, respectively.

Elementary functions (such as sin, cos, exp, etc.) are easily extended to intervals. The simplest inclusion function is called *natural interval extension*, which means that we replace the numbers x by the box $\boldsymbol{x}$ at each occurrence in the function f and compute the inclusion function in interval terms. One of the most commonly used inclusion functions is the *centered form*, which is a better approximation of the inclusion function than the natural interval extension but takes more time to compute. We compute the centred form using equation $\mathbf{f}_c(\boldsymbol{x}) = \mathbf{f}(c) + \nabla\mathbf{f}(\boldsymbol{x}) \cdot (\boldsymbol{x} - c)$, where $c \in \boldsymbol{x}$ is usually the centre of the box and $\nabla\mathbf{f}(\boldsymbol{x})$ is the inclusion of the gradient of f over $\boldsymbol{x}$. We use Automatic Differentiation (AD) to compute the inclusion of gradients [12]. In this method, only the derivative rules are needed for the calculation, and we calculate the function value and the derivative at the same time. The interested readers can find more information about Interval Arithmetic methods in [3, 5].

2.2. Solvers for interval-valued system of equations

By solving an interval-valued system of equations, we want to find an enclosure of all possible solutions within a starting box. However, if the enclosures of the coefficients are too wide, we cannot remove any part of the box. Sometimes we can improve the solvability of the methods by using preconditioners. This simply means that we transform the system of equations to be more suitable for the solver by using a transformation matrix. We use the midpoint preconditioner, which can be found in [7], along with other preconditioners. To solve the NIFJ-CS we use an iterative method, the Interval Gauss-Seidel method (IGS), and a direct method, the Interval LU decomposition (ILUD). These two methods are straightforward

extensions of methods to solve the real system of equations. The interested reader can find more information about these two methods and their implementations in [1, 13].

2.3. The prototype of the Interval Branch and Bound method

Branch and Bound is a framework for solving optimization problems in which the main problem is divided into subproblems and an attempt is made to discard the subproblems that do not contain the optimal point using some discarding rules. IBB is based on Interval Arithmetic and Branch and Bound concepts. All steps are extended to intervals. These steps must be specified for a particular implementation, and their choice can have a large impact on the efficiency of the method. Since we will only focus on one discarding test (the Fritz-John optimality test), the remaining steps are done in the usual way.

In an IBB method [6, 10], there are five main steps: selection, bounding, discarding, division, and termination. The prototype algorithm is shown in Algorithm 1.

Algorithm 1 Prototype Interval Branch and Bound method

$L_{work} \leftarrow \{\boldsymbol{x}\}; L_{result} \leftarrow \{\};$
while $L_{work} \neq \emptyset$ **do**
  Select a box $\boldsymbol{x}$ from L_{work} ▷ Selection Rule
  Compute bounds for $\mathbf{f}(\boldsymbol{x}), \mathbf{g}_i(\boldsymbol{x}), \forall i \in M$ ▷ Bounding Rule
  if $\boldsymbol{x}$ cannot be discarded **then** ▷ Discarding Tests
    Divide $\boldsymbol{x}$ into subboxes $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_r$ ▷ Division Rule
    for i = 1 to r **do**
      if TerminationCriterion$(\boldsymbol{x}_i, \varepsilon)$ **then** ▷ Termination Rule
        Store $\boldsymbol{x}_i$ in L_{result}
      else
        Store $\boldsymbol{x}_i$ in L_{work}
      end if
    end for
  end if
end while
return L_{result}

First, we initialize the working list (L_{work}) with the bound constraint and the result list (L_{result}) with an empty set. We stop the method when the working list is an empty set. In each iteration, we select a box $\boldsymbol{x}_{selected}$ from the working list with some selection rules. The selection rules can be LIFO, FIFO, or the lowest lower bound. In our implementation, we use the lowest lower bound selection rule. In the next step, we bound the objective of the box $\boldsymbol{x}_{selected}$ by computing the natural interval extension or the centered form. Sometimes, in the unconstrained case, we can discard the selected box, because it is monotone, concave, or does not contain

an optimal point (Interval Newton test), by using higher-order information, e.g. gradient. When we solve a constrained nonlinear problem, we can add two more tests. The first is the feasibility test, where we investigate whether the selected box is a subset of the feasible area by computing the bounds on the constraints. In this test, there are three possible cases: the undetermined case when some computed bounds contain zero ($\exists i \in 1,\dots,m$: $0 \in \mathbf{g}_i(\boldsymbol{x}_{selected})$), the infeasible case when one of the lower bounds of the computed bounds is greater than zero ($\exists i \in 1,\dots,m$: $\underline{\mathbf{g}_i}(\boldsymbol{x}_{selected}) > 0$), and the strictly feasible case, when all the upper bounds of the constraints are less than zero ($\overline{\mathbf{g}_i}(\boldsymbol{x}_{selected}) < 0, \forall i = 1,\dots,m$). To examine the optimality of the box, we can use the FJ or KKT optimality tests. If the selected box is not discarded and satisfies the termination rule, we can move it to the result list. The termination rule usually examines the width of the interval or the width of the inclusion function. Otherwise, we divide the selected box $\boldsymbol{x}_{selected}$ into subboxes $\boldsymbol{x}_1,\dots,\boldsymbol{x}_r$ using the division rule. The division rule can be bisection, trisection, multisection, etc. In this work, we use bisection as a division rule, dividing the box into two subboxes along the widest dimension.

In the next section, we will discuss the interval version of the FJ optimality conditions in more detail. We will examine and compare four possible solution methods.

3. The Interval Fritz-John Condition System

The Fritz John conditions are necessary conditions for a solution to be optimal in nonlinear programming. For problem (1.1), the FJ optimality conditions [9] for a given point x are the equations

$$\mu_0 \nabla f(x) + \sum_{i \in M_b} \mu_i \nabla p_i(x) + \sum_{j \in M_c} \mu_j \nabla g_j(x) = 0 \tag{3.1}$$

$$\mu_i p_i(x) = 0, \quad i \in M_b \tag{3.2}$$

$$\mu_j g_j(x) = 0, \quad j \in M_c \tag{3.3}$$

$$\mu_i \geq 0, \quad i \in M_b \cup M_c \cup \{0\}, \tag{3.4}$$

where μ_i are the Lagrange multipliers, M_b and M_c is the set of the bound constraints and the general constraints, respectively. Thus, μ_0 is the Lagrange multiplier of the objective function. If the system can be solved, we confirm that x can be the optimal solution. Note that we can easily formulate a bound constraint for x_i as $p_{i_u}(x) = x_i - \overline{y_i}$ and $p_{i_l}(x) = \underline{y_i} - x_i$, where $\overline{y_i}$ and $\underline{y_i}$ are the upper and lower bounds of the general bound constraint, respectively.

The straightforward extension of the Fritz John optimality conditions (3.1)–(3.4) for a given box $\boldsymbol{x}$ are the interval-valued system of equations

$$\boldsymbol{\mu}_0 \nabla \mathbf{f}(\boldsymbol{x}) + \sum_{i \in M_b} \boldsymbol{\mu}_i \nabla \mathbf{p}_i(\boldsymbol{x}) + \sum_{j \in M_c} \boldsymbol{\mu}_j \nabla \mathbf{g}_j(\boldsymbol{x}) = 0 \tag{3.5}$$

$$\boldsymbol{\mu}_i \mathbf{p}_i(\boldsymbol{x}) = 0, \quad i \in M_b \tag{3.6}$$

$$\boldsymbol{\mu}_j \mathbf{g}_j(\boldsymbol{x}) = 0, \quad j \in M_c \tag{3.7}$$

$$\boldsymbol{\mu}_i \geq 0, \quad i \in M_b \cup M_c \cup \{0\}, \tag{3.8}$$

where $\mathbf{f}(\boldsymbol{x})$, $\mathbf{p}_i(\boldsymbol{x})$, $\mathbf{g}_j(\boldsymbol{x})$ are the inclusion functions, $\nabla\mathbf{f}(\boldsymbol{x})$, $\nabla\mathbf{p}_i(\boldsymbol{x})$, $\nabla\mathbf{g}_j(\boldsymbol{x})$ are the inclusions of the gradients of $f(x), p_i(x), g_j(x)$, respectively. Note that we can reduce the number of equations in the system by considering only the active constraints. We consider a constraint active if the inclusion of the constraint, $\mathbf{p}_i(\boldsymbol{x})$, $\mathbf{g}_j(\boldsymbol{x})$, contains zero. Let B and C be the set of active bound constraints and active constraints, respectively. We can formalize the equations (3.5)–(3.8) for active constraints by replacing M_b with B and M_c with C.

3.1. The normalization of the Lagrange multipliers

The Interval FJ-CS usually does not include a normalization condition. One possible way to normalize Lagrange multipliers, following [2], is to use the equation $\mu_0 + \sum_{i \in B \cup C} \mu_i = 1$. We can easily formulate it as an interval-valued function,

$$\mathbf{r}(\boldsymbol{\mu}) = \boldsymbol{\mu}_0 + \sum_{i \in B \cup C} \boldsymbol{\mu}_i - 1 = 0. \tag{3.9}$$

Moreover, adding this condition to the FJ-CS does not remove any solution, and we can replace the interval $\boldsymbol{\mu}_i \geq 0$ with $[0, 1]$ for all Lagrange multipliers, which improves the success rate of the IGS.

3.2. The Normalized Interval Fritz-John Condition System

As before, B and C are the set of active bound constraints and active constraints, respectively. Let $N = 1 + n + |B| + |C|$ be the dimension of the system, where n is the dimension of the problem. Set the Lagrange multipliers, $\boldsymbol{\mu}_i$ $i \in B \cup C \cup \{0\}$, to the interval $[0, 1]$. When formalizing the system of equations, we consider the normalization function $\mathbf{r}(\boldsymbol{\mu})$ extended to the intervals defined in (3.9). Denote all the variables by $\boldsymbol{t} = [\boldsymbol{x}, \boldsymbol{\mu}]^T$, which is N-dimensional. Thus, for the box $\boldsymbol{t}$, we formalize the Normalized Interval Fritz-John Condition System (NIFJ-CS) as

$$\boldsymbol{\phi}(\boldsymbol{t}) = \begin{bmatrix} \mathbf{r}(\boldsymbol{\mu}) \\ \boldsymbol{\mu}_0 \cdot \nabla\mathbf{f}(\boldsymbol{x}) + \sum_{i \in B} \boldsymbol{\mu}_i \cdot \nabla\mathbf{p}_i(\boldsymbol{x}) + \sum_{j \in C} \boldsymbol{\mu}_j \cdot \nabla\mathbf{g}_j(\boldsymbol{x}) \\ \boldsymbol{\mu}_i \cdot \mathbf{p}_i(\boldsymbol{x}) \quad i \in B \\ \boldsymbol{\mu}_j \cdot \mathbf{g}_j(\boldsymbol{x}) \quad j \in C \end{bmatrix} = 0. \tag{3.10}$$

Note that this NIFJ-CS is equivalent to the (3.9), (3.5)–(3.7) interval-valued system of equations.

4. Solving the Normalized Interval Fritz-John Condition System

When we solve an interval-valued optimality condition system, we can discard the box if the solution is an empty set. However, if the enclosures of the gradients are too wide, we cannot remove any part of the box. One possible way to solve the system of equations $\boldsymbol{\phi}(\boldsymbol{t}) = 0$ is to apply the Newton method. Applying the Newton method as in [2], we obtain the system of equations

$$\mathbf{J}(\boldsymbol{t}) \cdot (\boldsymbol{t} - t_0) = -\boldsymbol{\phi}(t_0), \tag{4.1}$$

where $\mathbf{J}(\boldsymbol{t})$ is the Jacobian matrix for $\boldsymbol{\phi}(\boldsymbol{t})$, i.e. $\mathbf{J}_{ij}(\boldsymbol{t}) = \frac{\partial}{\partial \boldsymbol{t}_j}\boldsymbol{\phi}_i(\boldsymbol{t})$ $i, j = 1, \ldots, N$, and t_0 is an interior point of the box $\boldsymbol{t}$. Note that for t_0 we use the midpoint of the interval $\boldsymbol{t}$. In addition, we also use the midpoint preconditioner matrix, with which we transform (4.1) to the system of equations

$$P \cdot \mathbf{J}(\boldsymbol{t}) \cdot (\boldsymbol{t} - t_{mid}) = -P \cdot \boldsymbol{\phi}(t_{mid}), \tag{4.2}$$

where $P = \operatorname{mid}(\mathbf{J}(\boldsymbol{t}))^{-1}$. We solve the system of equations only if one bound constraint is active in each dimension. Otherwise, we skip solving NIFJ-CS, reducing the probability of an unsuccessful Newton method, as the system otherwise becomes too large. To solve (4.2), we use IGS, because this method is more efficient than the ILUD method. However, for IGS we need an initial box $\boldsymbol{t}$, but $\boldsymbol{x}$ is given, and $\boldsymbol{\mu}_i$ can be set to $[0, 1]$ for all Lagrange multipliers. Note that since an unsuccessful IGS step significantly increases the runtime of the IBB method, we apply the Newton method only once.

4.1. Estimating the Lagrange multipliers

In many cases, IGS cannot reduce or discard the box $\boldsymbol{x}$ because the initial bound of $\boldsymbol{t}$ is too large, that is, it contains many possible solutions. One way to reduce this problem is to estimate the Lagrange multipliers before applying the IGS method, and initialize $\boldsymbol{\mu}$ with the estimated bounds. The Lagrange multipliers can be approximated by solving the interval equations

$$\begin{gathered} \mathbf{r}(\boldsymbol{\mu}) = 1 \\ \boldsymbol{\mu}_0 \cdot \nabla \mathbf{f}(\boldsymbol{x}) + \sum_{i \in B} \boldsymbol{\mu}_i \cdot \nabla \mathbf{p}_i(\boldsymbol{x}) + \sum_{j \in C} \boldsymbol{\mu}_j \cdot \nabla \mathbf{g}_j(\boldsymbol{x}) = 0 \end{gathered} \tag{4.3}$$

Note that (4.3) are the first two equations of (3.10). To solve these equations, we use the ILUD method mentioned in Section 2.2.

4.2. Methods

To solve the NIFJ-CS (4.2), we investigate four methods based on the Newton method, where we modify one or more steps for each variant.

4.2.1. The naive NIFJ-CS method

In the naive NIFJ-CS method, we formalize the Newton step directly as a system of equations (4.1) and solve with IGS. We initialize the first n component of $\boldsymbol{t}$ with the examined box $\boldsymbol{x}$ and the remaining components with the interval $[0, 1]$ (initial bounds for the Lagrange multipliers). In the best case, we obtain an empty set; that is, there is no solution in the examined box $\boldsymbol{x}$. Thus, we can discard the box $\boldsymbol{x}$. Sometimes we obtain as a solution a tighter box within $\boldsymbol{x}$. In this case, we reduce the box $\boldsymbol{x}$ to it. In many cases, however, it is not possible to reduce the size of the box because it is overestimated. In this case, we obtain many boxes as solutions, as IGS splits each component where the diagonal coefficient interval contains zero. In general, we only divide the components $\boldsymbol{\mu}$ into subboxes and leave the box $\boldsymbol{x}$ unchanged. If the box $\boldsymbol{x}$ is unchanged, the Newton step was unsuccessful. If we obtain subboxes divided in the part of the box $\boldsymbol{x}$, we exchange $\boldsymbol{x}$ to these subboxes. Note that we do not store the estimated Lagrange multipliers because we do not want to increase the required memory for the IBB. We want to point out that in most cases the box $\boldsymbol{x}$ cannot be reduced or discarded.

4.2.2. Lagrange estimator method

In the Lagrange estimator method, we solve (4.3) using the ILUD method. In this case, we do not need initial bounds on the Lagrange multipliers. We solve it if the system is independent, but not underdetermined. We want to use these bounds to decide whether or not to discard the examined box. We can use the ILUD method only if the system is squared. In an overdetermined case, we solve the system using only the first $1+|B|+|C|$ equations. First, we check if there are upper bounds less than 0. If so, the box is discarded because some estimated Lagrange multipliers are negative. If we have more equations than variables, after the first check, we solve the remaining equations with the obtained $\boldsymbol{\mu}$. If the Lagrange multipliers do not satisfy all the equations, we discard the examined box.

4.2.3. Lagrange estimator + NIFJ-CS method

In this method, we first estimate the Lagrange multipliers using the method in Section 4.2.2. We might discard the box by the Lagrange estimator method. If we cannot discard the box, the estimated bounds on the Lagrange multipliers are truncated with $[0, 1]$. We solve the system of equations with the calculated multipliers as described in Section 4.2.1. As a result, we can discard the box $\boldsymbol{x}$ or reduce it by the Newton method.

4.2.4. Taylor expansion of the NIFJ-CS

This method is very similar to the naive NIFJ-CS method in Section 4.2.1. However, in this case, we replace some intervals in the Jacobian matrix with their midpoints. We use the Jacobian of $\boldsymbol{\phi}(\boldsymbol{x})$ as $\mathbf{J}_{ij}(\boldsymbol{t}) = \frac{\partial}{\partial \boldsymbol{t}_j}\boldsymbol{\phi}_i(\boldsymbol{t}_1, \ldots, \boldsymbol{t}_j, t_{j+1}, \ldots, t_N)$ $i, j = 1, \ldots, N$, where the $N - j$ components are real numbers. This reduction is

valid, since the Taylor expansion of $\boldsymbol{\phi}(\boldsymbol{t})$ is done for each variable one by one. It might produce a tighter inclusion; however, the computational cost of the Jacobian is much higher. The interested reader can find more information on the background of this method in [3].

4.3. Additional improvements

As a further development, we examine the success of the methods based on the order of the equations and the variables. Ordering the variables does not have a huge effect on the solvability of the NIFJ-CS. It only increases the required computation time. Sorting the equations in descending order by

$$o_i = \frac{\overline{\mathbf{g}_i(\boldsymbol{x})}}{\overline{\mathbf{g}_i(\boldsymbol{x})} - \underline{\mathbf{g}_i(\boldsymbol{x})}},$$

we can improve the success rate of the methods described in Sections 4.2.1 and 4.2.3 without significantly increasing the computation time.

5. Computational experiments

In this section, we compare the four methods together with the IBB method without NIFJ-CS. We implemented the IBB method with the following discarding tests: Newton, midpoint, cutoff, concavity, monotonicity, feasibility, and FJ test.

In detail, we use the bisection method as the division step. The termination criterion for any box $\boldsymbol{x}$ is $\operatorname{wid}(\boldsymbol{x}) < \varepsilon$ or $\operatorname{wid}(\mathbf{f}(\boldsymbol{x})) < \varepsilon$, where $\varepsilon = 10^{-6}$. We sort the working list in ascending order by the lower bound and delete any box whose lower bound is greater than the current upper bound (cutoff test). We select the first element from the working list. We stop the algorithm when the working list is empty or when we reach the maximum running time (7 200 seconds).

We implement the IBB method in Matlab R2020a version 8 [4] and use Intlab 11 [11]. From Intlab, we used only the IA, AD, and ILUD methods. We implement other tests, methods, and functions. The complete project can be found on GitLab[1]. The abbreviations of the five methods can be seen in Table 1.

Table 1. Abbreviations of the methods.

IBBWO	IBB without FJ optimality conditions
NFJ	IBB with the naive NIFJ-CS method (see Section 4.2.1)
Lag	IBB with the Lagrange estimator method (see Section 4.2.2)
Lag + NFJ	IBB with the Lagrange estimator + NIFJ-CS (see Section 4.2.3)
Tay + NFJ	IBB with Taylor expansion of the NIFJ-CS (see Section 4.2.4)

[1]The IBB with Optimality condition - Intlab: https://gitlab.com/gencsimiska27/the-ibb-with-optimality-condition-intlab

Table 2. Computation time.

Test name	IBBWO	NFJ	Lag	Lag + NFJ	Tay + NFJ
circle	21.4	30.6	13.7	**12.7**	61.1
ex14_1_1	755.5	4 321.1	774.2	**750.5**	5 565.4
ex14_1_4	266.2	764.3	218.9	**201.5**	836.7
ex14_1_8	**41.9**	84.6	51.1	48.8	181.7
ex2_1_2	496.3	597.9	**55**	58.3	4 331.3
ex2_1_4	🕒	🕒	🕒	🕒	🕒
ex2_1_6	**346.9**	1 762.7	361.8	355.7	2 463.7
ex3_1_2	680.8	1 543.9	87.4	**80.5**	2 240
ex3_1_3	🕒	🕒	225.4	**201.8**	🕒
ex3_1_4	620.3	563.4	181.8	**142.5**	374
ex4_1_9	🕒	6 843.4	5 032.5	**4 501.1**	5 836.1
ex7_2_3	🕒	🕒	🕒	🕒	🕒
ex7_2_4	🕒	🕒	🕒	🕒	🕒
ex7_3_2	62	133.0	26.3	**14.3**	296.2
Gomez Levy	**3.2**	3.8	3.9	4.5	8.8
Mishra's Bird	0.7	0.9	0.6	**0.6**	0.7
Rbrock (disk)	**0.8**	1.5	0.9	0.9	1.7
Rbrock (cube)	**0.7**	1.4	0.9	1	2.4
Simionescu	64.4	10.2	**4.2**	5	132.8
Hansen Test	**0.5**	1.7	0.8	0.9	4.3
Avg. Comp. Time	1 968.2	2 273.4	1 432	**1 399.1**	2 556.9

We used two benchmarks to compare the methods. The first benchmark is the GLOBALLib[2], from which we chose 14 constrained nonlinear programming test cases having only inequality constraints. The second benchmark is called Test Functions for Constrained Optimization from the Wikipedia website[3], which contains five two-dimensional test cases. In some cases, general bound constraints were not given. Thus, we used a large interval, $[-10\,000, 10\,000]$, which encloses the optimum points in these cases. All runs were performed on an AMD Ryzen 7 3800X 8-Core Processor with 32 GB RAM. In addition, we use an additional test case from [2], named the Hansen test.

The running time for each test case and method can be seen in Table 2. The symbol 🕒 means that we cannot reach the required accuracy in 7 200 seconds, but we still have possible solutions in the result list. We can see that we can reduce the

[2]GLOBALLib: http://www.gamsworld.org/global/globallib.htm

[3]Test Functions for Constrained Optimization: https://en.wikipedia.org/wiki/Test_functions_for_optimization

required computation time for the IBB methods by using the optimality conditions in most cases. Only a few small problems cannot be improved with them. The NFJ method takes equal to or more time to compute than the other methods. It is because we try to solve an interval-valued system of equations with overestimated variables (initial bounds for Lagrange multipliers, overestimated enclosures of the gradients) and in many cases we are not able to reduce or discard the studied box. The calculation time of the Tay + NFJ method is sometimes long but almost always better than NFJ due to the sharper gradient enclosures. However, it takes more time to calculate the gradient with different input boxes.

In average time, the best method is the Lag + NFJ method, which takes 1 399.1 seconds on average for 20 test cases, but the Lag method comes very close. The small difference is due to the fact that the NIFJ-CS method is solved with estimated bounds on the Lagrange multipliers.

Table 3. Relative deviations of the Weighted Function Evaluations from the best results (in bold).

Test name	IBBWO	NFJ	Lag	Lag + NFJ	Tay + NFJ
circle	142%	384%	**26 464**	327%	2 185%
ex14_1_1	**1 057 048**	784%	100%	101%	981%
ex14_1_4	**165 154**	618%	132%	194%	1 036%
ex14_1_8	**37 518**	693%	121%	180%	761%
ex2_1_2	1 807%	4 282%	**19 719**	118%	10 127%
ex2_1_4	**2 519 349**	150%	124%	122%	239%
ex2_1_6	**116 566**	2 038%	100%	101%	2 809%
ex3_1_2	728%	3 993%	**69 215**	185%	7 905%
ex3_1_3	525%	2885%	**164 486**	155%	3 299%
ex3_1_4	482%	473%	**153 782**	141%	260%
ex4_1_9	171%	219%	**4 719 013**	126%	114%
ex7_2_3	**3 835 776**	536%	122%	130%	695%
ex7_2_4	**1 311 026**	185%	144%	153%	651%
ex7_3_2	209%	1 613%	**51 862**	273%	3 552%
Gomez Levy	**1 416**	374%	110%	116%	216%
Mishra's Bird	179%	138%	102%	103%	120%
Rbrock (disk)	**613**	497%	117%	122%	210%
Rbrock (cube)	**888**	336%	129%	158%	360%
Simionescu	1 270%	140%	**2 826**	107%	2274%
Hansen Test	**1 276**	161%	113%	141%	660%
Avg. WFE	121%	361%	**817 505**	113%	477%

In Table 3, we compare the weighted function evaluations (WFE), computed

as WFE = Feval + $n \cdot$ Geval + $\frac{n^2}{2} \cdot$ Heval, where n is the dimension of the problem, Feval, Geval, Heval are the number of function, gradient and Hessian evaluations, respectively. We report the best results in bold, and the relative deviation for the rest. One can see that, on average, the Lag method requires the least number of function evaluations. The evaluation for IBBWO, Lag, and Lag + NFJ is very close, and these three methods solve the problems with the least function evaluations. The other two methods require more evaluations, because many times solving NIFJ-CS is unsuccessful.

The relative deviations of the averages for the methods can be seen in Table 4. We compare the five methods in five aspects: computation time, weighted number of function evaluations (WFE), number of result boxes, number of iterations, and number of NIFJ-CS tests, respectively. Furthermore, we study the success rate of IBB methods that include NIFJ-CS. We can see that the Lag + NFJ method is the best in terms of computation time (1 399.1 s), number of results boxes (4), and FJ test (15 551). The success rate of this method is 49%, which is very high compared to the rest. IBBWO is worst in all aspects compared to the best results. The NFJ method is sometimes worse than the IBBWO method, which is caused by the high number of unsuccessful NIFJ-CS. The Lag method requires the least function evaluation because we do not solve the NIFJ-CS. The Tay + NFJ method requires the least number of iterations, but the number of function evaluations is significantly higher, increasing its computation time.

Table 4. The relative average deviations.

Method	IBBWO	NFJ	Lag	Lag + NFJ	Tay + NFJ
Computation time	141%	162%	102%	**1 399.1**	183%
WFE	121%	361%	**817 505**	113%	477%
No. result boxes	**4**	**4**	**4**	**4**	**4**
No. iterations	238%	320%	243%	239%	**21 340**
No. FJ Test	-	157%	104%	**15 551**	143%
FJ Success Percentage	-	7%	38%	**49%**	25%

6. Summary

We studied the applicability of optimality conditions in Interval Branch and Bound method to constrained problems. We introduced the NIFJ-CS, which is based on the Fritz-John optimality conditions. We found that the naive NIFJ-CS is difficult to solve. This is due to the overestimation of the Lagrange multipliers and the enclosure of the gradients. We studied four versions (NFJ, Lag, Lag + NFJ, Tay + NFJ) for solving the NIFJ-CS, and compared their efficiency together with the IBB method without optimality conditions in 20 test cases from the literature. We found that the best method for solving NIFJ-CS is the Lag + NFJ method. The average time required for this method is 1 399.1 seconds, and the success rate is 49%.

In the future, we want to improve the success rate of the Lag + NFJ method by introducing a preliminary test before trying to solve NIFJ-CS. The preliminary test should discard the box, which certainly does not contain an optimal point, within a short computation time. In addition, we plan to use constraint propagation to reduce the box as much as possible. We also want to extend the methods to problems with equality constraints.

References

[1] A. Goldsztejn, G. Chabert: *A Generalized Interval LU Decomposition for the Solution of Interval Linear Systems*, in: Aug. 2006, pp. 312–319, ISBN: 978-3-540-70940-4, DOI: 10.1007/978-3-540-70942-8_37.

[2] E. Hansen, G. Walster: *Bounds for Lagrange multipliers and optimal points*, Computers & Mathematics with Applications 25.10 (1993), pp. 59–69, ISSN: 0898-1221, DOI: 10.1016/0898-1221(93)90282-Z.

[3] E. Hansen, G. W. Walster: *Global Optimization Using Interval Analysis: Revised And Expanded*, CRC Press, 2003, p. 728, ISBN: 0824740599, DOI: 10.1201/9780203026922.

[4] T. M. Inc.: *MATLAB version: 9.13.0 (R2022a)*, Natick, Massachusetts, United States, 2022, URL: https://www.mathworks.com.

[5] L. Jaulin, M. Kieffer, O. Didrit, E. Walter: *Applied Interval Analysis with Examples in Parameter and State Estimation, Robust Control and Robotics*, Aug. 2001, ISBN: 1852332190, DOI: 10.1108/k.2002.06731eae.002.

[6] R. B. Kearfott: *An Interval Branch and Bound Algorithm for Bound Constrained Optimization Problems*, Journal of Global Optimization 2 (1992), pp. 259–280, DOI: 10.1007/BF00171829.

[7] R. B. Kearfott: *Preconditioners for the Interval Gauss–Seidel Method*, SIAM Journal on Numerical Analysis 27.3 (1990), pp. 804–822, DOI: 10.1137/0727047.

[8] R. Kearfott, M. Nakao, A. Neumaier, S. Rump, S. Shary, P. Van Hentenryck: *Standardized notation in interval analysis*, Vychislitel'nye Tekhnologii 15 (Jan. 2010).

[9] O. Mangasarian, S. Fromovitz: *The Fritz John necessary optimality conditions in the presence of equality and inequality constraints*, Journal of Mathematical Analysis and Applications 17.1 (1967), pp. 37–47, DOI: 10.1016/0022-247X(67)90163-1.

[10] L. Pál, T. Csendes: *INTLAB implementation of an interval global optimization algorithm*, Optimization Methods and Software 24.4-5 (2009), pp. 749–759, DOI: 10.1080/10556780902753395.

[11] S. Rump: *INTLAB - INTerval LABoratory*, in: Developments in Reliable Computing, ed. by T. Csendes, http://www.tuhh.de/ti3/rump/, Dordrecht: Kluwer Academic Publishers, 1999, pp. 77–104.

[12] H. J. S., A. Griewank, G. F. Corliss: *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, Mathematics of Computation 62.205 (Jan. 1994), p. 434, DOI: 10.2307/2153424.

[13] S. P. Shary: *Interval Gauss-Seidel Method for Generalized Solution Sets to Interval Linear Systems*, Reliable Computing 7.2 (Apr. 2001), pp. 141–155, ISSN: 1573-1340, DOI: 10.1023/A:1011422215157.