“REGULAR-SS12: The Bernstein branch-and-prune algorithm for constrained global optimization of multivariate polynomial MINLPs”

Bhagyesh V. Patil

Cambridge Centre for Advanced Research in Energy Efficiency in Singapore
50 Nanyang Ave, Singapore 639798
bvpatil@ntu.edu.sg

Abstract. This paper address the global optimization problem of polynomial mixed-integer nonlinear programs (MINLPs). A improved branch-and-prune algorithm based on the Bernstein form is proposed to solve such MINLPs. The algorithm use a new pruning feature based on the Bernstein form, called the Bernstein box and Bernstein hull consistency. The proposed algorithm is tested on a set of 16 MINLPs chosen from the literature. The efficacy of the proposed algorithm is brought out via numerical studies with the previously reported Bernstein algorithms and several state-of-the-art MINLP solvers.

1 Introduction

Optimizing a MINLP is a challenging task and has been a point of attraction to many researchers from academia as well as industry. This work present a new solution procedure for such MINLPs. Typically, this work addresses MINLPs of the following form:

$$\begin{aligned} &\min_{x} \quad f(x) \\ &\text{subject to} \\ &g_i(x) \leq 0, \quad i = 1, 2, \ldots, m \\ &h_j(x) = 0, \quad j = 1, 2, \ldots, n \\ &x_k \in \mathbf{x} \subseteq \mathbb{R}, \quad k = 1, 2, \ldots, l_d \\ &x_k \in \{0, 1, \ldots, q\} \subset \mathbb{Z}, \quad k = l_d + 1, \ldots, l \end{aligned} \tag{1}$$

where $f : \mathbb{R}^l \mapsto \mathbb{R}$ is the (possibly nonlinear polynomial) objective function, $g_i : \mathbb{R}^l \mapsto \mathbb{R}$ $(i = 1, 2, \ldots, m)$, and $h_j : \mathbb{R}^l \mapsto \mathbb{R}$ $(j = 1, 2, \ldots, n)$ are the (possibly nonlinear polynomial) inequality and equality constraint functions. Further, $\mathbf{x} := [\underline{x}, \overline{x}]$ is an interval in $\mathbb{R}$, x_k $(k = 1, 2, \ldots, l_d)$ are continuous decision variables, and the rest of x_k $(k = l_d + 1, \ldots, l)$ are integer decision variables with values 0 to q, $q \in \mathbb{Z}$.

Several techniques exist in literature to solve MINLP problems. Most of these techniques either decompose and reformulate the original problem (1) into a series of mixed-integer linear programs (MILPs) and nonlinear programs (NLPs), or they attempt to solve a NLP relaxation in a branch-and-bound framework. The interested reader can refer [2] and references therein for more specific details about these techniques.

Recently, global optimization algorithms based on the Bernstein polynomial approach has been proposed (see [11], [12]), and found to be very effective in solving small to medium dimensional polynomial MINLPs of the form (1). The current scope of the work involve systematic extension of the above proposed Bernstein global optimization algorithms to form a new improved algorithm. The improved algorithm is of a branch-and-prune type and use consistency techniques (constraint propagation) based on the Bernstein form. The consistency techniques prune regions from a solution search space that surely do not contain the global minimizer(s) [5], [6], hence this improved algorithm is defined as the Bernstein branch-and-prune algorithm for the MINLPs (that is, BBPMINLP). The algorithm BBPMINLP has some new features: (a) the consistency techniques are framed in a context of the Bernstein form, namely Bernstein box consistency and Bernstein hull consistency. (b) a new form of domain contraction step based on the application of Bernstein box and Bernstein hull consistency to a constraint $f(x) \leq \tilde{f}$ is introduced. The main feature of the algorithm BBPMINLP is, all operations (branching and pruning) are done using the Bernstein coefficients.

The performance of the algorithm BBPMINLP is compared with the earlier reported Bernstein algorithms BMIO [12] and IBBBC [11], as well as with several state-of-the-art MINLP solvers on a collection of 16 test problems chosen from the literature. The performance comparison is made on the basis of the number of boxes processed (between the algorithms BMIO, IBBBC, and BBPMINLP), and ability to locate a correct global minimum (between state-of-the-art MINLP solvers and the algorithm BBPMINLP). The findings are reported at the end of the paper.

The rest of the paper is organized as follows. In Section 2, the reader is introduced to some background of the Bernstein form. In Section 3, the consistency techniques are introduced. In sequel, Bernstein box and Bernstein hull consistency techniques are also presented. In Section 4, the main global optimization algorithm BBPMINLP to solve the MINLP problems is presented. Finally, some conclusions based on the present work are presented in the Section 5.

2 Background

This section briefly presents some notions about the Bernstein form. Due to the space limitation, a simple univariate Bernstein form is introduced. A comprehensive background and mathematical treatment for a multivariate case can be found in [12].

We can write a univariate l-degree polynomial p over an interval $\mathbf{x}$ in the form

$$p(x) = \sum_{i=0}^{l} a_i x^i, \qquad a_i \in \mathbb{R}\ . \tag{2}$$

Now the polynomial p can be expanded into the Bernstein polynomials of the same degree as below

$$p\left(x\right) = \sum_{i=0}^{l} b_i\left(\mathbf{x}\right) B_i^l\left(x\right)\ . \tag{3}$$

where B_i^l are the Bernstein basis polynomials and $b_i(\mathbf{x})$ are the Bernstein coefficients give as below

$$B_i^l = \binom{l}{i} x^i (1-x)^{1-i}\ . \tag{4}$$

$$b_i\left(\mathbf{x}\right) = \sum_{j=0}^{i} \frac{\binom{i}{j}}{\binom{l}{j}} a_j, \quad i = 0, \ldots, l. \tag{5}$$

Equation (3) is referred as the Bernstein form of a polynomial.

Theorem 1 *(Range enclosure property) Let p be a polynomial of degree l, and let $\overline{p}(\mathbf{x})$ denote the range of p on a given interval $\mathbf{x}$. Then,*

$$\overline{p}(\mathbf{x}) \subseteq B(\mathbf{x}) := \left[\min\ \left(b_i(\mathbf{x})\right),\ \max\ \left(b_i(\mathbf{x})\right)\right]. \tag{6}$$

Proof: See [4].

Remark 1 The above theorem says that the minimum and maximum coefficients of the array $(b_i(\mathbf{x}))$ provide lower and upper bounds for the range. This forms the Bernstein range enclosure, defined by $B(\mathbf{x})$ in equation (6). The Bernstein range enclosure can successively be sharpened by the continuous domain subdivision procedure [4].

3 Consistency techniques

The consistency techniques are used for pruning (deleting) unwanted regions that surely do not contain the global minimizer(s) from the solution search space. This pruning is achieved by assessing consistency of the algebraic equations (in our case inequality and equality constraints) over a given box $\mathbf{x}$.

This section now describe algorithms based on the consistency ideas borrowed from [5], and expanded in context of the Bernstein form. Henceforth, these algorithms are called as Bernstein box consistency (BBC) and Bernstein hull consistency (BHC). These algorithms work as a pruning operator in the main global optimization algorithm BBPMINLP (reported in the Section 4).

3.1 Bernstein box consistency

A Bernstein box consistency (BBC) technique is used to contract the bounds on a variable domain. The implementation of a BBC involve the application of a one-dimensional Bernstein Newton contractor [9] to solve a single equation for a single variable.

Consider an equality constraint polynomial $g(\mathbf{x}) = 0$, and let $(b(\mathbf{x}))$ be the Bernstein coefficients array of $g(\mathbf{x})$. Consider any component direction, say the first, with $\mathbf{x}_1 = [a, b]$. In the BBC technique, typically an attempt is made to increase the value of a and decrease the value of b, thus effectively reducing the width of $\mathbf{x}_1$.

To increase the value of a, first find all those Bernstein coefficients of $(b(\mathbf{x}))$ corresponding to $x_1 = a$. The minimum to maximum of these coefficients gives an interval denoted by $\mathbf{g}(a)$. If $0 \notin \mathbf{g}(a)$, then the constraint is infeasible at this endpoint a, and we search starting from a, along $x_1 = [a, b]$ for the first point at which constraint becomes just feasible, that is, we try to find a zero of $g(x)$. Let us denote this zero as a'. Clearly, $g(x)$ is infeasible over $[a, a')$, and so it can discarded to get a contracted interval $[a', b]$. On the other hand, if $0 \in \mathbf{g}(a)$ then we abandon the process to increase a and instead switch over to the other endpoint b and make an attempt to decrease it in the same way as we did to increase a.

To find a zero of $\mathbf{g}$ in $[a, b]$, one iteration of the univariate version of the Bernstein Newton contractor given in [9] is used. It is as follows

$$\mathbf{N}(\mathbf{x}_1) = a - (\mathbf{g}(a)/\mathbf{g}'_{x_1}),$$
$$\mathbf{x}'_1 = \mathbf{x}_1 \cap \mathbf{N}(\mathbf{x}_1),$$

where, $\mathbf{g}(a)$ is the minimum to maximum of the Bernstein coefficients array $(b(\mathbf{x}))$ at $x_1 = a$, $\mathbf{g}'_{x_1}$ denotes an interval enclosure for the derivative of $\mathbf{g}$ on $\mathbf{x}_1$, and $\mathbf{x}'_1$ gives a new contracted interval. A similar process is carried out from the other endpoint b, and if desired, the whole process can be repeated over all other component directions to a get contracted box $\mathbf{x}'$

The algorithm for the BBC which can be applied to both equality and inequality constraints is as follows.

Algorithm Bernstein box consistency: $\mathbf{x}' = \mathrm{BBC}((b_g(\mathbf{x})), \mathbf{x}, r, x_{status,r}, eq\_type)$

Inputs: The Bernstein coefficient array $(b_g(\mathbf{x}))$ of a given constraint polynomial $g(x)$, the l-dimensional box $\mathbf{x}$, the direction r (decision variable) for which the bounds are to be contracted, flag $x_{status,r}$ to indicate whether r^{th} direction (decision variable) is continuous ($x_{status,r} = 0$) or integer ($x_{status,r} = 1$), and flag *eq_type* to indicate whether $g(x)$ is equality constraint ($eq\_type = 0$) or inequality constraint ($eq\_type = 1$).

Outputs: A box $\mathbf{x}'$ that is contracted using Bernstein box consistency technique for a given constraint polynomial $g(x)$.

BEGIN Algorithm

1. Set $a = \inf \mathbf{x}_r$, $b = \sup \mathbf{x}_r$.
2. From the Bernstein coefficient array $(b_g(\mathbf{x}))$, compute the derivative enclosure $\mathbf{g}_{x_r}^{'}$ in the direction x_r.
3. (Consider left endpoint of $\mathbf{x}_r$). Obtain the Bernstein range enclosure $\mathbf{g}(a)$ as the minimum to maximum from the Bernstein coefficient array of $(b_g(\mathbf{x}))$ for $x_r = a$.
4. If $eq\_type = 1$, then modify $\mathbf{g}(a)$ as $\mathbf{g}(a) = [\min\ \mathbf{g}(a), \inf]$.
5. If $0 \in \mathbf{g}(a)$, then we cannot increase a. Go to step 8 and try from the right endpoint b of the interval $\mathbf{x}_r$.
6. Do one iteration of the univariate Bernstein Newton contractor

$$\mathbf{N}(\mathbf{x}_r) = a - (\mathbf{g}(a)/\mathbf{g}'_{x_r}).$$
$$\mathbf{x}'_{r_a} = \mathbf{x}_r \cap \mathbf{N}(\mathbf{x}_r).$$

7. If $\mathbf{x}'_{r_a} = \emptyset$, then there is no zero of $\mathbf{g}$ on entire interval $\mathbf{x}_r$ and hence the constraint g is infeasible over box $\mathbf{x}$. EXIT the algorithm in this case with $\mathbf{x}' = \emptyset$.
8. (Consider right endpoint of $\mathbf{x}_r$). Obtain the Bernstein range enclosure $\mathbf{g}(b)$ as the minimum to maximum from the Bernstein coefficient array of $(b(\mathbf{x}))$ for $x_r = b$.
9. If $eq\_type = 1$, then modify $\mathbf{g}(b)$ as $\mathbf{g}(b) = [\min\ \mathbf{g}(b), \inf]$.
10. If $0 \in \mathbf{g}(b)$, then we cannot decrease b. Go to step 13
11. Do one iteration of the univariate Bernstein Newton contractor

$$\mathbf{N}(\mathbf{x}_r) = b - (\mathbf{g}(b)/\mathbf{g}'_{x_r}).$$
$$\mathbf{x}'_{r_b} = \mathbf{x}_r \cap \mathbf{N}(\mathbf{x}_r).$$

12. If $\mathbf{x}'_{r_b} = \emptyset$, EXIT the algorithm with $\mathbf{x}' = \emptyset$.
13. Compute $\mathbf{x}'_r$ as follows:
    (a) $\mathbf{x}'_r = \mathbf{x}'_{r_a} \cap \mathbf{x}'_{r_b}$, if both $\mathbf{x}'_{r_a}$ and $\mathbf{x}'_{r_b}$ are computed.
    (b) $\mathbf{x}'_r = \mathbf{x}'_{r_a}$ or $\mathbf{x}'_{r_b}$, which ever is computed.
    (c) $\mathbf{x}'_r = \mathbf{x}_r$ (both $\mathbf{x}'_{r_a}$ and $\mathbf{x}'_{r_b}$ are not computed).
14. for $k = 1, 2$ if $x_{status,r} = 1$ then
    (a) if $\mathbf{x}(r,k)$ and $\mathbf{x}'_r(r,k)$ are equal then go to substep (e).
    (b) Set $t_a = \mathbf{x}(r,k)$, and $t_b = \mathbf{x}'_r(r,k)$.
    (c) if $t_a > t_b$ then set $\mathbf{x}'_r(r,k) = \lfloor \mathbf{x}'_r(r,k) \rfloor$.
    (d) if $t_a < t_b$ then set $\mathbf{x}'_r(r,k) = \lceil \mathbf{x}'_r(r,k) \rceil$.
    (e) end (of k-loop).
15. Return $\mathbf{x}' = \mathbf{x}'_r$.

END Algorithm

3.2 Algorithm Bernstein box consistency for a set of constraints

A single application of the proposed algorithm BBC in the section 3.1 can contract only one variable domain. For a multivariate constraint, in turn, we can apply BBC to each variable separately. Below algorithm, called as BBC2SET applies BBC to all the variables present in a constraint, and if there are multiple constraints, BBC2SET applies BBC to all of them simultaneously.

Algorithm BBC for a set of constraints: $\mathbf{x}' = \mathrm{BBC2SET}(B, k, C, \mathbf{x}, x_{status})$

Inputs: A cell structure B containing Bernstein coefficient arrays of all the constraint polynomials with first k Bernstein coefficient arrays are for the equality constraints, the total number of constraints C, the $l-$dimensional box $\mathbf{x}$, and a column vector x_{status} describing the status (continuous or integer) of the each variable x_i $(i = 1, 2, \ldots, l)$.

Outputs: A contracted box $\mathbf{x}'$.

BEGIN Algorithm

1. Set $r = 0$.
2. (a) for $i = 1, 2, \ldots, l$
   (b) for $j = 1, 2, \ldots, C$
      (i) Set $r = r + 1$, and $x_{status,r} = x_{status}(r)$. if $r > l$ then $r = 1$.
      (ii) if $j < k$ then $\mathbf{x}_1$=BBC$(B\{j\}, \mathbf{x}, r, x_{status,r}, 0)$.
      (iii) if $j > k$ then $\mathbf{x}_1$=BBC$(B\{j\}, \mathbf{x}, r, x_{status,r}, 1)$.
      (iv) Update $\mathbf{x} = \mathbf{x} \cap \mathbf{x}_1$.
      (v) if $\mathbf{x} = \emptyset$, then set $\mathbf{x}' = \emptyset$ and EXIT the algorithm.
   (c) end (of $i-$loop).
   (d) end (of $j-$loop).
3. Return $\mathbf{x}' = \mathbf{x}$.

END Algorithm

3.3 Bernstein hull consistency

Similar to a BBC, a Bernstein hull consistency (BHC) technique contract bounds on a variable domain. The typical BHC procedure is as below.

Consider a multivariate equality constraint $h(x) = 0$. To apply BHC to a selected term of $h(x) = 0$, we need to keep the selected term on the left hand side and remaining all other terms need to be taken on the right hand side, that is, we write the constraint in the form $a_I x^I = h_1(x)$ where, $x = (x_1, x_2, \ldots, x_l)$ and $I = (i_1, i_2, \ldots, i_l)$. The new contracted interval for the variable $\mathbf{x}_r$ (in r^{th} direction) can be obtained as

$$\mathbf{x}'_r = \left(\frac{\mathbf{h}'}{a_I \prod \mathbf{x}_k^{i_k}} \right)^{1/i_r} \bigcap \ \mathbf{x}_r \ , \quad r = 1, 2, \ldots, l. \tag{7}$$

Here to compute $\mathbf{h}'$ we compute the Bernstein coefficients of the monomial term $a_I x^I$ and from them subtract the Bernstein coefficients of the constraint polynomial $h(x)$. The minimum to maximum of this subtracted Bernstein coefficients will give $\mathbf{h}'$. For a given constraint all the terms can be solved or only selected terms can be solved.

The algorithm for the BHC that can be applied for both equality and inequality constraints is as follows.

Algorithm Bernstein hull consistency: $\mathbf{x}' = \mathrm{BHC}((b_g(\mathbf{x})), a_I, I, \mathbf{x}, x_{status}, eq\_type)$

Inputs: The Bernstein coefficient array $(b_g(\mathbf{x}))$ of a given constraint polynomial $g\,(x)$, coefficient a_I of the selected term t, power I of the each variable in term t, the $l-$dimensional box $\mathbf{x}$, a column vector $x_{status,r}$ describing the status (if continuous, then $x_{status,r} = 0$; if integer, then $x_{status,r} = 1$) of the each variable x_r $(r = 1, 2, \ldots, l)$, and flag *eq_type* to indicate whether $g(x)$ is equality constraint ($eq\_type = 0$) or inequality constraint($eq\_type = 1$).

Outputs: A box $\mathbf{x}'$ that is contracted using Bernstein hull consistency technique applied to a given constraint polynomial $g\,(x)$ and selected term t.

BEGIN Algorithm

1. Compute the Bernstein coefficient array of the selected term t as $(b_t\,(\mathbf{x}))$.
2. Obtain the Bernstein coefficients of the constraint inverse polynomial by subtracting $(b_g(\mathbf{x}))$ from $(b_t\,(\mathbf{x}))$, and then obtain its Bernstein range enclosure as the minimum to maximum of these Bernstein coefficients. Denote it as $\mathbf{h}'$.
3. if $eq\_type = 1$ then
   (a) Compute an interval $\mathbf{y}$ as $\mathbf{y} = [-\infty, 0] \cap [\min\,(b_g(x)), \max\,(b_g(x))]$.
   (b) if $\mathbf{y} = \emptyset$ then set $\mathbf{x}' = \emptyset$, and EXIT the algorithm. Else modify $\mathbf{h}'$ as $\mathbf{h}' = \mathbf{h}' + \mathbf{y}$.
4. (a) for $r = 1, 2, \ldots, l$
   (b) Compute $\mathbf{x}_r' = \left(\frac{\mathbf{h}'}{a_I \prod \mathbf{x}_k^{i_k}}\right)^{1/i_r} \bigcap\ \mathbf{x}_r$
   (c) for $k = 1, 2$ if $x_{status}(r) = 1$ then
      (i) if $\mathbf{x}(r,k)$ and $\mathbf{x}_r'(r,k)$ are equal then go to substep (v).
      (ii) Set $t_a = \mathbf{x}(r,k)$ and $t_b = \mathbf{x}_r'(r,k)$.
      (iii) if $t_a > t_b$ then set $\mathbf{x}_r'(r,k) = \lfloor \mathbf{x}_r'(r,k) \rfloor$.
      (iv) if $t_a < t_b$ then set $\mathbf{x}_r'(r,k) = \lceil \mathbf{x}_r'(r,k) \rceil$.
      (v) end (of $k-$loop).
   (d) end (of $r-$loop).
5. Return $\mathbf{x}'$.

END Algorithm

3.4 Algorithm Bernstein hull consistency for a set of constraints

A single application of BHC algorithm can be made only to a single term of the selected constraint. However, in practice, we may want to apply BHC to more terms, or if there is more than one constraint, we may want to call BHC several times.

Below algorithm BHC2SET applies BHC to the multiple terms and to the multiple constraints. This algorithm will call BHC several times. Our criteria for term selection is as follows. In a given constraint, if a term contains maximum power for any of the variable, then it is selected. If the term contains maximum power for two variables, then it is solved two times and so on. This criteria is inspired from the ideas about interval hull consistency reported in [5].

Algorithm BHC for a set of constraints: $\mathbf{x}' = \text{BHC2SET}(A, B, k, C, \mathbf{x}, x_{status})$

Inputs: The cell structure A containing the coefficient arrays of all constraint polynomials with first k coefficient arrays are for the equality constraints, a cell structure B containing Bernstein coefficient arrays of all the constraint polynomials, where first k Bernstein coefficient arrays are for the equality constraints, the total number of constraints C, the $l-$dimensional box $\mathbf{x}$, and a column vector $x_{status,r}$ describing the status (if continuous, then $x_{status,r} = 0$; if integer, then $x_{status,r} = 1$) of the each variable x_r $(r = 1, 2, \ldots, l)$.

Outputs: A contracted box $\mathbf{x}'$.

BEGIN Algorithm

1. Set $r = 0$.
2. (a) for $i = 1, 2, \ldots, l$
   (b) for $j = 1, 2, \ldots, C$
      (i) Set $r = r + 1$. if $r > l$ then $r = 1$
      (ii) Select the term having the maximum power for r in the constraint j, and obtain the coefficient a_I of the selected term and I containing the power of each variable in the selected term (this shall be obtained from A).
      (iii) if $j < k$ then $\mathbf{x}_1$=BHC($B\{j\}, a_I, I, \mathbf{x}, x_{status}, 0$).
      (iv) if $j > k$ then $\mathbf{x}_1$=BHC($B\{j\}, a_I, I, \mathbf{x}, x_{status}, 1$).
      (v) Update $\mathbf{x} = \mathbf{x} \cap \mathbf{x}_1$.
      (vi) if $\mathbf{x} = \emptyset$ then set $\mathbf{x}' = \emptyset$, and EXIT the algorithm.
   (c) end (of $j-$loop).
   (d) end (of $i-$loop).
3. Return $\mathbf{x}' = \mathbf{x}$.

END Algorithm

4 Main algorithm BBPMINLP

This section presents the main algorithm for constrained global optimization of the MINLPs of a form (1). The working of the algorithm is similar to a interval branch-and-bound procedure, but with following enhancements.

- This algorithm use the Bernstein form as a inclusion function for the global optimization.
- Unlike classical subdivision procedure, the algorithm use a modified subdivision procedure from [11].
- Similarly, this algorithm use a efficient cut-off test, called as a vectorized Bernstein cut-off test (VBCT) from [11].
- Further, this algorithm use the efficient Bernstein box and Bernstein hull consistency techniques. These techniques serve as a pruning operator in the algorithm, thereby speeding up the convergence of the algorithm.

Algorithm Bernstein branch-and-prune constrained optimization:
$[\widetilde{y}, \widetilde{p}, U]$=BBPMINLP$(N, a_I, \mathbf{x}, x_{status}, \epsilon_p, \epsilon_x, \epsilon_{zero})$

Inputs: Degree N of the variables occurring in the objective and constraint polynomials, the coefficients a_I of the objective and constraint polynomials in the power form, the initial search domain $\mathbf{x}$, a column vector $x_{status,r}$ describing the status (if continuous, then $x_{status,r} = 0$; if integer, then $x_{status,r} = 1$) of a each variable x_r $(r = 1, 2, \ldots, l)$, the tolerance parameters ϵ_p and ϵ_x on the global minimum and global minimizer(s), and the tolerance parameter ϵ_{zero} to which the equality constraints are to be satisfied.

Outputs: A lower bound $\widetilde{y}$ and an upper bound $\widetilde{p}$ on the global minimum f^*, along with a set U containing all the global minimizer(s) $\mathbf{x}^{(i)}$.

BEGIN Algorithm

1. Set $\mathbf{y} := \mathbf{x}$ and $y_{status,r} := x_{status,r}$.
2. From a_I, compute the Bernstein coefficient arrays of the objective and constraint polynomials on the box $\mathbf{y}$ respectively as $(b_o(\mathbf{y})), (b_{gi}(\mathbf{y})), (b_{hj}(\mathbf{y}))$, $i = 1, 2, ..., m,\ j = 1, 2, ..., n$.
3. Set $\widetilde{p} := \infty$ and $y := \min(b_o(\mathbf{y}))$.
4. Set $R = (R_1, \ldots, R_m, R_{m+1}, \ldots, R_{m+n}) := (0, \ldots, 0)$.
5. Initialize list $\mathcal{L} := \{(\mathbf{y}, y, R, (b_o(\mathbf{y})), (b_{gi}(\mathbf{y})), (b_{hj}(\mathbf{y})))\}$, $\mathcal{L}^{sol} := \{\}$.
6. If $\mathcal{L}$ is empty then go to step 22. Otherwise, pick the first item $(\mathbf{y}, y, R, (b_o(\mathbf{y})), (b_{gi}(\mathbf{y})), (b_{hj}(\mathbf{y})))$ from $\mathcal{L}$, and delete its entry from $\mathcal{L}$.
7. Apply the Bernstein hull consistency algorithm to the relation $f(\mathbf{y}) \le \widetilde{p}$. If the result is empty, then delete item $(\mathbf{y}, y, R, (b_o(\mathbf{y})), (b_{gi}(\mathbf{y})), (b_{hj}(\mathbf{y})))$ and go to step 6.
$$\mathbf{y}' = \text{BHC}((b_o(\mathbf{y})), a_I, I, \mathbf{y}, y_{status,r}, 1)$$
8. Set $\mathbf{y} := \mathbf{y}'$ and compute the Bernstein coefficient arrays of the objective and constraint polynomials on the box $\mathbf{y}$, respectively as $(b_o(\mathbf{y})), (b_{gi}(\mathbf{y})), (b_{hj}(\mathbf{y}))$, $i = 1, 2, ..., m,\ j = 1, 2, ..., n$. Also set $y := \min(b_o(\mathbf{y}))$.

9. Apply the Bernstein box consistency algorithm to the $f(\mathbf{y}) \leq \widetilde{p}$. If the result is empty, then delete item $(\mathbf{y}, y, R, (b_o(\mathbf{y})), (b_{gi}(\mathbf{y})), (b_{hj}(\mathbf{y})))$ and go to step 6.

$$\mathbf{y}' = \mathrm{BBC}((b_o(\mathbf{y})), \mathbf{y}, r, y_{status,r}, 1)$$

   where bound contraction will be applied in the r^{th} direction.
10. Set $\mathbf{y} := \mathbf{y}'$ and compute the Bernstein coefficient arrays of the objective and constraint polynomials on the box $\mathbf{y}$, respectively as $(b_o(\mathbf{y})), (b_{gi}(\mathbf{y})), (b_{hj}(\mathbf{y}))$, $i = 1, 2, ..., m$, $j = 1, 2, ..., n$. Also set $y := \min(b_o(\mathbf{y}))$.
11. {Contract domain box by applying Bernstein hull consistency to all the constraints} Apply the algorithm BHC2SET to all the constraints

$$\mathbf{y}' = \mathrm{BHC2SET}(A_c, B_c, k, C, \mathbf{y}, y_{status,r})$$

   Here A_c is a cell structure containing the coefficient arrays of the all constraints, where the first k coefficient arrays are for the equality constraints, B_c is a cell structure containing the Bernstein coefficient arrays of the all constraints, where the first k Bernstein coefficient arrays are for the equality constraints, C is the total number of constraints, $\mathbf{y}$ is a domain box, and $\mathbf{y}'$ is the new contracted box.
12. Set $\mathbf{y} := \mathbf{y}'$ and compute the Bernstein coefficient arrays of the objective and constraint polynomials on the box $\mathbf{y}$, respectively as $(b_o(\mathbf{y})), (b_{gi}(\mathbf{y})), (b_{hj}(\mathbf{y}))$, $i = 1, 2, ..., m$, $j = 1, 2, ..., n$. Also set $y := \min(b_o(\mathbf{y}))$.
13. {Contract domain box by applying Bernstein box consistency to all the constraints} Apply the algorithm BBC2SET to all the constraints

$$\mathbf{y}' = \mathrm{BBC2SET}(B_c, k, C, \mathbf{y}, y_{status,r})$$

   Here B_c is a cell structure containing the Bernstein coefficient arrays of all the constraints, where the first k Bernstein coefficient arrays are for the equality constraints, C is the total number of constraints, $\mathbf{y}$ is a domain box, and $\mathbf{y}'$ is a new contracted box.
14. Set $\mathbf{y} := \mathbf{y}'$ and compute the Bernstein coefficient arrays of the objective and constraint polynomials on the box $\mathbf{y}$, respectively as $(b_o(\mathbf{y})), (b_{gi}(\mathbf{y})), (b_{hj}(\mathbf{y}))$, $i = 1, 2, ..., m$, $j = 1, 2, ..., n$. Also set $y := \min(b_o(\mathbf{y}))$.
15. {Branching}
    (a) If $w(\mathbf{y}_i) = 0$ for all $i = l_d + 1, \ldots, l$ (that is, all the integer variables has been fixed to some integer values from there respective domains) then go to substep (c).
    (b) Choose a coordinate direction λ parallel to which $\mathbf{y}_{l_d+1} \times \cdots \times \mathbf{y}_l$ has an edge of maximum length, that is $\lambda \in \{i : w(\mathbf{y}) := w(\mathbf{y}_i), i = l_d+1, \ldots, l\}$. Go to step 16.
    (c) Choose a coordinate direction λ parallel to which $\mathbf{y}_1 \times \cdots \times \mathbf{y}_{l_d}$ has an edge of maximum length, that is $\lambda \in \{i : w(\mathbf{y}) := w(\mathbf{y}_i), i = 1, \ldots, l_d\}$.
16. Bisect $\mathbf{y}$ normal to direction λ, getting boxes $\mathbf{v}_1$, $\mathbf{v}_2$ such that $\mathbf{y} = \mathbf{v}_1 \cup \mathbf{v}_2$. The modified subdivision procedure from [11] is used.
17. for $k = 1, 2$

(a) Set $R^k = (R_1^k, \ldots, R_m^k, R_{m+1}^k, \ldots, R_{m+n}^k) := R$.
(b) Find the Bernstein coefficient array and the corresponding Bernstein range enclosure of the objective function (f) over $\mathbf{v}_k$ as $(b_0(\mathbf{v}_k))$ and $B_0(\mathbf{v}_k)$, respectively.
(c) Set $d_k := \min B_o(\mathbf{v}_k)$.
(d) If $\widetilde{p} < d_k$ then go to substep (j).
(e) for $i = 1, 2, \ldots, m$ if $R_i = 0$ then
    (i) Find the Bernstein coefficient array and the corresponding Bernstein range enclosure of the inequality constraint polynomial (g_i) over $\mathbf{v}_k$ as $(b_{gi}(\mathbf{v}_k))$ and $B_{gi}(\mathbf{v}_k)$, respectively.
    (ii) If $B_{gi}(\mathbf{v}_k) > 0$ then go to substep (j).
    (iii) If $B_{gi}(\mathbf{v}_k) \leq 0$ then set $R_i^k := 1$.
(f) for $j = 1, 2, \ldots, n$ if $R_{m+j} = 0$ then
    (i) Find the Bernstein coefficient array and the corresponding Bernstein range enclosure of the equality constraint polynomial (h_j) over $\mathbf{v}_k$ as $(b_{hj}(\mathbf{v}_k))$ and $B_{hj}(\mathbf{v}_k)$, respectively.
    (ii) If $0 \notin B_{hj}(\mathbf{v}_k)$ then go to substep (j).
    (iii) If $B_{hj}(\mathbf{v}_k) \subseteq [-\epsilon_{zero}, \epsilon_{zero}]$ then set $R_{m+j}^k := 1$.
(g) If $R^k = (1, \ldots, 1)$ then set $\widetilde{p} := \min(\widetilde{p}, \max B_o(\mathbf{v}_k))$.
(h) Enter $(\mathbf{v}_k, d_k, R^k)$ into the list $\mathcal{L}$ such that the second members of all items of the list do not decrease.
(j) end (of $k-$loop).

18. {Cut-off test} Discard all items $(\mathbf{z}, z, R, (b_o(\mathbf{z})), (b_{gi}(\mathbf{z})), (b_{hj}(\mathbf{z})))$ in the list $\mathcal{L}$ that satisfy $\widetilde{p} < z$. For the remaining items in the list $\mathcal{L}$ apply the vectorized Bernstein cut-off test from [11], and update the current minimum estimate $\widetilde{p}$.
19. Denote the first item of the list $\mathcal{L}$ by $(\mathbf{y}, y, R, (b_o(\mathbf{y})), (b_{gi}(\mathbf{y})), (b_{hj}(\mathbf{y})))$.
20. If $(w(\mathbf{y}) < \epsilon_x)$ & $(\max\ B_o(\mathbf{y}) - \min\ B_o(\mathbf{y})) < \epsilon_p$ then remove the item from the list $\mathcal{L}$ and enter it into the solution list $\mathcal{L}^{sol}$.
21. Go to step 6.
22. {Compute the global minimum} Set the global minimum $\widetilde{y}$ to the minimum of the second entries over all the items in $\mathcal{L}^{sol}$.
23. {Compute the global minimizers} Find all those items in $\mathcal{L}^{sol}$ for which the second entries are equal to $\widetilde{y}$. The first entries of these items contain the global minimizer(s) $\mathbf{x}^{(i)}$.
24. Return the lower bound $\widetilde{y}$ and upper bound $\widetilde{p}$ on the global minimum f^*, along with the set U containing all the global minimizer(s) $\mathbf{x}^{(i)}$.

END Algorithm

5 Numerical studies

This section reports a numerical experimentation with the algorithm BBPMINLP on a set of 16 test problems. These test problems were chosen from [3], [8], [13]. At the outset, the performance of the algorithm BBPMINLP was compared with the Bernstein algorithms BMIO in [12] and IBBBC in [11]. Further, the algorithm BBPMINLP was compared with the four state-of-the-art MINLP solvers, namely AlphaECP, BARON, Bonmin, DICOPT, whose GAMS interface is available through the NEOS server [10], and one MATLAB based open-source solver BNB20 [7].

For all computations, a desktop PC with Pentium IV 2.40 GHz processor with 2 GB RAM was used. The algorithm BBPMINLP was implemented in the MATLAB [1] with an accuracy $\epsilon = 10^{-6}$ for computing the global minimum and global minimizer(s), and a maximum limit on the number of subdivisions to be 500.

Table 1 describes the list of symbols for Table 3. Table 2 reports for the 16 test problems, the total number of boxes processed and the computational time taken in seconds to locate a correct global minimum by the Bernstein algorithms BMIO, IBBBC, and the algorithm BBPMINLP reported in this work. The algorithm BBPMINLP was compared using three different flags described as below:

- **A**: Application of the Bernstein hull consistency to the inequality and equality constraints, that is algorithm BHC2SET (see Section 3.4) is applied to these constraints.
- **B**: Application of the Bernstein box consistency to the inequality and equality constraints, that is algorithm BBC2SET (see Section 3.2) is applied to these constraints.
- **C**: Application of the Bernstein hull and box consistencies to the constraint $f(x) \leq \widetilde{f}$ ($\widetilde{f}$ is the current global minimum estimate). This serves to delete a subbox that bounds a nonoptimal point of $f(x)$.

The findings are as below. It was observed that the algorithm BMIO failed to solve for the four test problems (wester, hmittelman, sep1, tln5) and the algorithm IBBBC is unable to solve one test problem sep1. Similary, the algorithm BBPMINLP with flags A and C is unable to solve one test problem (sep1). This is perhaps the Bernstein hull consistency in this problem was unable to sufficiently prune the search region, and hence may take more time to find the solution. However, for one test problem (tln5) we found the algorithm BBPMINLP with flag A to be more efficient than the others. In contrast, the algorithm BBPMINLP with flag B was able to successfully solve all the test problems. Moreover, it was observed for two test problems (wester, hmittelman) algorithm with flag B performed exceptionally well than the others. Overall, the performance of the algorithm BBPMINLP with flag B was seen to be the best in terms of both the number of boxes processed and the computational time it took to found a global minimum.

Table 3 reports for the 16 test problems the quality of the global minimum obtained with the algorithm BBPMINLP and the state-of-the-art MINLP solvers[1]. The bold values in the table indicate the local minimum value. For these test problems the performance of the state-of-the-art solvers was as follows:

[1] All the solver were executed in their default options for the 16 test problems considered.

- AlphaECP found the local minimum for two test problem (zhu2, tln5), and failed to solve one test problem (hmittelman).
- Bonmin found the local minimum for two test problems (zhu2, tln5).
- BNB20 found the local minimum for four test problems (floudas1, zhu2, hmittelman, sep1), and failed to solve three test problems (wester, st_test3, tln5).
- DICOPT found the local minimum for three test problems (floudas1, zhu2, tln5), and failed solve two test problems (zhu1, hmittelman).

However, the algorithm BBPMINLP was able to found the correct the global minimum value for all the test problems, and compares well with the state-of-the-art solvers in terms of the computational time.

6 Conclusions

In this work the Bernstein algorithm (BBPMINLP) was proposed to solve the polynomial type of MINLPs. This algorithm was composed with the two new solution search space pruning operators, namely the Bernstein box and Bernstein hull consistency. Further, the proposed algorithm also used another pruning operator based on the application of the Bernstein box and hull consistency to a constraint based on the objective function $f(x)$ and a current minimum estimate $\widetilde{f}$. This step along with a cut-off test improves the convergence of the algorithm. The performance of the proposed algorithm BBPMINLP was tested on a collection of 16 test problems. The test problems had dimensions ranging from 2 to 35 and number of constraints varying from 1 to 31. At the outset, the effectiveness of the algorithm BBPMINLP was demonstrated over the previously reported Bernstein algorithms BMIO and IBBBC. The algorithm BBPMINLP was found to be more efficient in the number of boxes processed, resulting an average reduction of $96-99\%$ compared to BMIO and $42-88\%$ compared to IBBBC. Similarly, from the computational perspective BBPMINLP was found to be well competent with the algorithms BMIO and IBBBC.

Lastly, the performance of the algorithm BBPMINLP was compared with the existing state-of-the-art MINLP solvers, such as AlphaECP, BARON, Bonmin, BNB20, and DICOPT. Test results showed the superiority of the proposed algorithm BBPMINLP over state-of-the-art MINLP solvers in terms of the solution quality obtained. Specifically, all solvers (except BARON) located local solution or failed for atleast one problem from a set of 16 test problems considered. On the otherhand, the algorithm BBPMINLP could locate correct global minimum for all the test problems. In terms of the computational time, BBPMINLP was some order of magnitudes slower than the considered MINLP solvers. However, this could be due to the difference in the computing platforms used for the algorithm implementation and testing.

Table 1. Description of symbols for Table 3.

Symbol	Description
l	Total number of the decision variables (binary, integer and continuous)
f^*	Bold values in this row indicates local minimum obtained
*	Indicates that the solver failed giving the message "relaxed NLP is unbounded"
**	Indicates that the solver searched one hour for the solution, still could not find the solution and therefore was terminated
***	Indicates that the solver returned the message "terminated by the solver"
****	Indicates that the solver failed giving the message "infeasible row with only small Jacobian elements"

Table 2. Comparison of the number of boxes processed and computational time (in seconds) taken by the earlier Bernstein algorithms BMIO, IBBBC and the algorithm BBPMINLP.

Example	l	Statistics	BMIO	IBBBC	BBPMINLP		
					A	B	C
floudas1	2	Boxes	1003	33	29	10	31
		Time	0.45	0.08	0.3	0.10	0.18
zhu1	2	Boxes	1166	173	63	61	81
		Time	1.05	0.14	0.5	0.40	0.59
st_testph4	3	Boxes	1870	47	20	15	29
		Time	2.21	0.18	0.15	0.10	0.44
nvs21	3	Boxes	1149	785	125	67	615
		Time	0.81	0.10	0.23	0.31	1.17
gbd	4	Boxes	2201	23	23	5	15
		Time	1.40	0.09	0.11	0.02	0.28
st_e27	4	Boxes	572	21	5	5	13
		Time	0.40	0.08	0.06	0.07	0.21
zhu2	5	Boxes	2571	700	84	81	173
		Time	2.71	1.40	3.35	2.30	4.13
st_test2	6	Boxes	2987	107	17	16	5
		Time	1.63	0.18	0.30	0.12	0.11
wester	6	Boxes	*	1621	1500	4	6003
		Time		5.25	300	0.07	39.83
alan	8	Boxes	4015	1	1	1	1
		Time	3.03	0.01	0.01	0.02	0.01
ex1225	8	Boxes	6869	385	343	85	261
		Time	6.60	0.15	0.7	0.40	3.17
st_test6	10	Boxes	3003	111	18	18	91
		Time	3.57	2.68	1.25	0.70	11.51
st_test3	13	Boxes	3960	340	119	21	261
		Time	48.50	4.32	5.61	4.31	75.40
hmittelman	16	Boxes	*	431	5000	3	191
		Time		61.52	1561	1.35	316.44
sep1	29	Boxes	*	**	**	1034	**
		Time				5.96	
tln5	35	Boxes	*	>10,000	1003	2972	>8003
		Time			68.28	18.96	

* indicates that the algorithm returned "out of memory error".
** indicates that the algorithm did not give the result even after one hour and is therefore terminated.

Table 3. Comparison of the global minimum obtained and computational time (in seconds) taken by the algorithm BBPMINLP with state-of-the-art MINLP solvers.

Example	l	Statistics	Solver/Algorithm					
			AlphaECP	BARON	Bonmin	BNB20	DICOPT	BBPMINLP
floudas1	2	f^*	−8.5	−8.5	−8.5	**−5**	**−4**	−8.5
		Time	1.04	0.25	0.14	0.01	0.21	0.1
zhu1	2	f^*	−3.9374E+10	−3.9374E+10	−3.9374E+10	−3.9374E+10	*	−3.9374E+10
		Time	1.36	0.25	0.16	0.07		0.40
st_testph4	3	f^*	−80.5	−80.5	−80.5	−80.5	−80.5	−80.5
		Time	0.89	0.26	0.26	0.22	0.47	0.10
nvs21	3	f^*	−5.68	−5.68	−5.68	−5.68	−5.68	−5.68
		Time	15.54	1.06	0.16	0.29	0.23	0.31
gbd	4	f^*	2.2	2.2	2.2	2.2	2.2	2.2
		Time	0.5	0.25	0.22	0.03	0.22	0.02
st_e27	4	f^*	2	2	2	2	2	2
		Time	0.71	0.26	0.13	0.01	0.22	0.07
zhu2	5	f^*	**0**	−51,568	**0**	**−42, 585**	**0**	−51,568
		Time	1.94	0.25	0.16	1.38	0.23	2.30
st_test2	6	f^*	−9.25	−9.25	−9.25	−9.25	−9.25	−9.25
		Time	1.83	0.32	0.31	0.29	0.84	0.12
wester	6	f^*	112,235	112,235	112,235	**	1,12,235	112,235
		Time	6.66	0.37	0.08		0.82	0.07
alan	8	f^*	2.92	2.92	2.92	2.92	2.92	2.92
		Time	0.61	0.23	0.20	0.14	1.02	0.02
ex1225	8	f^*	31	31	31	31	31	31
		Time	0.72	0.26	0.28	0.28	0.47	0.40
st_test6	10	f^*	471	471	471	471	471	471
		Time	3.56	1.42	1.17	0.82	1.42	0.70
st_test3	13	f^*	−7	−7	−7	**	−7	−7
		Time	0.94	0.27	0.61		0.98	4.31
hmittelman	16	f^*	***	13	13	19	****	13
		Time		0.42	2.62	0.09		1.35
sep1	29	f^*	−510.08	−510.08	−510.08	−50	−510.08	−510.08
		Time	7.91	0.06	0.04	0.14	0.001	5.96
tln5	35	f^*	**10.6**	10.3	**10.6**	**	**13.7**	10.3
		Time	12.23	0.53	52.74		0.002	18.96

Acknowledgement

This work was funded by the Singapore National Research Foundation (NRF) under its Campus for Research Excellence And Technological Enterprise (CREATE) programme and the Cambridge Centre for Advanced Research in Energy Efficiency in Singapore (CARES).

References

1. The Mathworks Inc., MATLAB version 7.1 (R14) (Natick, MA, 2005)
2. D'Ambrosio, C., Lodi, A.: Mixed integer nonlinear programming tools: an updated practical overview. Annals of Operations Research 204(1), 301–320 (2013)
3. Floudas, C.A.: Nonlinear and mixed-integer optimization: fundamentals and applications. Oxford University Press, New York (1995)
4. Garloff, J.: The Bernstein algorithm. Interval Computations 2, 154–168 (1993)
5. Hansen, E.R., Walster, G.W.: Global optimization using interval analysis, 2^{nd} edition. Marcel Dekker, New York (2005)
6. Hooker, J.: Logic-based methods for optimization: combining optimization and constraint satisfaction. John Wiley, New York (2000)
7. Kuipers, K.: Branch-and-bound solver for mixed-integer nonlinear optimization problems. MATLAB Central for File Exchange. (Retrieved 18 December 2009)
8. GAMS MINLP Model Library: Available at http://www.gamsworld.org/minlp/minlplib/minlpstat.htm. (Accessed 20 March 2015)
9. Nataraj, P.S.V., Arounassalame, M.: An interval Newton method based on the Bernstein form for bounding the zeros of polynomial systems. Reliable Computing 15(2), 109–119 (2011)
10. NEOS server for optimization.: Available at http://www.neos-server.org/neos/solvers/index.html. (Accessed 20 March 2015)
11. Patil, B.V., Nataraj, P.S.V.: An improved Bernstein global optimization algorithm for MINLP problems with application in process industry. Mathematics in Computer Science 8(3-4), 357–377 (2014)
12. Patil, B.V., Nataraj, P.S.V., Bhartiya, S.: Global optimization of mixed-integer nonlinear (polynomial) programming problems: the Bernstein polynomial approach. Computing 94(2-4), 325–343 (2012)
13. Zhu, W.: A provable better branch and bound method for a nonconvex integer quadratic programming problem. Journal of Computer and System Sciences 70(1), 107–117 (2005)