Polynomial Approximation for Binary Nonlinear Programming

Luca Mencarelli[1], Sourour Elloumi[2,3]

[1] Dipartimento di Informatica, Università di Pisa, I-56127 Pisa, Italy
luca.mencarelli@unipi.it
[2] UMA, ENSTA Paris, Institut Polytechnique de Paris, F-91120 Palaiseau, France
sourour.elloumi@ensta-paris.fr
[3] CEDRIC-Cnam, 292 rue saint Martin, F-75141 Paris Cedex 03, France

Abstract

In this paper, we present a simple heuristic approach based on polynomial approximation for binary nonlinear unconstrained optimization: we consider binary nonlinear programs, we replace the objective functions with a Lagrange multivariate polynomial interpolation computed at the Smolyak grid points, we solve the corresponding polynomial program via standard linearization, and we compute the original objective value at the minimum of the polynomial approximation. We present preliminary promising computational results by comparing the proposed algorithm against two exact MINLP solvers, namely Couenne and Scip, on several synthetic instances.

Keywords : *Binary programming, polynomial approximation, standard linearization.*

1 Introduction

We consider the binary nonlinear (possibly non-convex) unconstrained optimization problem:

$$\min_{x\in\{0,1\}^n} f(x)\,. \tag{BNP}$$

We suppose that computing $f(x)$ for $x \in \{0,1\}^n$ is relatively easy. BNP is at least **NP**-hard since it is a generalization of Binary Quadratic Optimization Problems (BQPs): BQPs are BNPs with quadratic objective function. Moreover, when the objective function of BNP is polynomial, we obtain unconstrained 0-1 polynomial programs.

2 PA4BNP Algorithm

The pseudo-code of heuristic algorithm we propose is reported in Alg. 1. Initially, we build a multivariate polynomial interpolation of the function $f(x)$ over an initial set of binary points S_1 (Step 3). Then, we solve the polynomial approximation (Step 4) and obtain a set of feasible solutions, which we eventually add to the set of approximating point (Step 7). Starting from these latter points, we optionally perform a `LocalSearch` to improve the quality of the solutions by exploring local neighbourhoods of each solution (Steps 9-12): one way to implement this procedure is reported in Alg. 2. In the next subsections we give few more details about the main steps of `PA4BNP` algorithm.

2.1 Building the polynomial approximation (Step 3)

Initially, we build a multivariate Lagrange polynomial interpolation $PI(x)$ of the function $f(x)$ [3] over the Smolyak (sparse) grid, such that

Algorithm 1 PA4BNP Algorithm

Input: an instance P of BNP, a set S of binary points, and parameters K and q
Output: a feasible solution x^* for P

initialization: $S_1 := S$ and $x_1^* := \arg\min_{x \in S_1} f(x)$, $t := 1$
while $t \leq T$ **do**
  build the polynomial interpolation $PI(x)$ of $f(x)$ with respect to set S_t
  minimize $PI(x)$ over $x \in \{0,1\}^n$ and obtain K feasible solutions $\tilde{x}_1, \ldots, \tilde{x}_K$
  $S_{t+1} := S_t$ and $x_{t+1}^* := x_t^*$
  for $k = 1, \ldots, K$ **do**
    if $\tilde{x}_k \notin S_{t+1}$, then $S_{t+1} := S_{t+1} \cup \{\tilde{x}_k\}$
    if $f(\tilde{x}_k) < f(x_{t+1}^*)$, then $x_{t+1}^* := \tilde{x}_k$
    obtain q candidate points $x_1^{LS}, \ldots, x_q^{LS}$, starting from $\tilde{x}_k$
    for $i = 1, \ldots, q$ **do**
      if $x_i^{LS} \notin S_{t+1}$, then $S_{t+1} := S_{t+1} \cup \{x_i^{LS}\}$
      if $f(x_i^{LS}) < f(x_{t+1}^*)$, then $x_{t+1}^* := x_i^{LS}$
  if $x_{t+1}^* = x_t^*$, then **break**
  $t := t + 1$
return x_{t+1}^*

- the interpolation suffers less from curse-of-dimensionality than uniform grid points, such as Latin hypercube, and
- the error bound between the true optimum and the optimum estimated by corresponding polynomial approximation has a weaker dependence on dimensionality n.

Multivariate Lagrange polynomial interpolation $PI(x)$ is defined as the linear combination of the Lagrange basis polynomials, composed by multivariate polynomials of degree $\leq d$ (we set $d := 4$ in computational experiments), so that $PI(x)$ has degree at most d. We have as many Lagrange polynomials as there are interpolating points, *i.e.*, $|S_t|$. With the previous approach we obtain quite sparse approximating polynomials.

In order to avoid the so-called Runge's phenomenon, the nodes are computed using Chebyshev-Gauss-Lobatto and Clenshaw-Curtis equidistant nodes. This approach is already been successfully exploited in the contest of black-box optimization for Smolyak grid built on Chebyshev extrema, see the seminal paper [1].

In computational experiments, we use the Julia package `SmolyakApprox.jl` [5], and for the multivariate Lagrange polynomial interpolation we use the `lagrange_nd` C++ package developed by John Burkardt [2].

2.2 Solving the polynomial approximation (Step 4)

We deal with the polynomial approximated problem, by applying standard linearization and solving the corresponding BLP. In particular, we consider the 0-1 (unconstrained) polynomial optimization problems:

$$\min_{x \in \{0,1\}^n} PI(x), \quad \text{with } PI(x) = \sum_{p \leq m} a_p \prod_{i \in \mathcal{M}_p} x_i \,, \tag{1}$$

where m is the number of monomials, each of them containing the variables whose indexes belong to $\mathcal{M}_p$ ($p \leq m$). In order to obtain a BLP, we replace each monomial with an additional variable y_p such that:

$$y_p \leq x_i \quad \forall i \in \mathcal{M}_p \tag{2}$$
$$y_p \geq \textstyle\sum_{i \in \mathcal{M}_p} x_i - (|\mathcal{M}_p| - 1) \tag{3}$$
$$y_p \geq 0 \,. \tag{4}$$

Note that if $a_p < 0$ it is enough to add constraints (2) and (4) to problem (1), while if $a_p > 0$ we add only constraints (3) and (4). In our case, standard linearization technique is enough to solving problem (1), since we are dealing with sparse polynomials. In the implementation, we maintain a solution pool of the best K solutions found by the algorithm used to solve problem (1) (see Section 3).

2.3 LocalSearch (Steps 9-12)

We implement also several LocalSearch variants, by differently exploring the neighbourhoods of an initial solution obtained by flipping each binary component of the given solution:

- `LS1` (see Alg. 2) which flips independently each single component of the initial solution and take the best solution in terms of objective function,
- `LS2` (see Alg. 3), which progressively flips the components of the initial solution and take the first solution improving the objective function, and
- `LS3` (see Alg. 4), which generates candidate solutions as in the `LS2`, but continues exploring the neighbourhoods of the initial solution while improving objective function value.

Algorithm 2 LS1 Algorithm

Input: an instance P of BNP and a binary solution $\overline{x}$
Output: a (possible improved) feasible solution x^* for P

initialization: $x^* := \overline{x}$
for $i = 1, \ldots, n$ **do**
    define a candidate solution x_i^{LS} by flipping the i-th component of $\overline{x}$
    if $f(x_i^{LS}) < f(x^*)$, then $x^* := x_i^{LS}$
return x^*

Algorithm 3 LS2 Algorithm

Input: an instance P of BNP and a binary solution $\overline{x}$
Output: a (possible improved) feasible solution x^* for P

initialization: $x^* := \overline{x}$
$x_0^{LS} := \overline{x}$
for $i = 1, \ldots, n$ **do**
    define a candidate solution x_i^{LS} by flipping the i-th component of x_{i-1}^{LS}
    if $f(x_i^{LS}) < f(x^*)$, then $x^* := x_i^{LS}$ and **break**
return x^*

Algorithm 4 LS3 Algorithm

Input: an instance P of BNP and a binary solution $\overline{x}$
Output: a (possible improved) feasible solution x^* for P

initialization: $x^* := \overline{x}$, and $x_0^{LS} := \overline{x}$
$x_0^{LS} := \overline{x}$
for $i = 1, \ldots, n$ **do**
    define a candidate solution x_i^{LS} by flipping the i-th component of x_{i-1}^{LS}
    if $f(x_i^{LS}) < f(x^*)$, then $x^* := x_i^{LS}$
    if $f(x_i^{LS}) > f(x^*)$, then **break**
return x^*

3 Computational experiments

All codes have been developed in `Julia 1.5.3`. We compare the `PA4BNP` using `Gurobi 9.1.2` to solve the BLPs obtained by standard linearization, against two exact solver for non-convex MINLPs, namely `Couenne 0.5.8` and `SCIP.jl 0.9.8` on a set of synthetic instances, generated according to [4], *i.e.*,

$$f(x) = \frac{\sum_{i \leq I} f_i \prod_{j \neq i} \|x - z_j\|^{\alpha_j}}{\sum_{i \leq I} \prod_{j \neq i} \|x - z_j\|^{\alpha_j}}, \tag{5}$$

with parameters: $I \in \mathbb{N}$, $z_j \in [0,1]^n$ for all $j \leq I$, $f_i \in \mathbb{R}$ for all $i \leq I$, and $\alpha_j \in \mathbb{R}^+$ for all $j \leq I$. If $\alpha_j > 1$, then $\lim_{x \mapsto z_j} \nabla f(x) = 0$.

We set a time limit of 300 seconds for all the algorithms, and $K := 1$, $q := 1$, and $\alpha_j := 2$ for $j \leq I$. We randomly generate z_j for $j \leq I$ and the values f_i's in $[-1000, 1000]$. We consider two sets of instances: the first one with $z_j \in \{0,1\}^n$ for all $j \leq I$, and the second one with $z_j \in [0,1]^n$ for all $j \leq I$. For the first set of instances, the optimal solution is known *a priori*, namely $f(x^*) = \min_{i \leq I} f_i$. The instance names are `n_I_b` and `n_I_c` for the first and second sets, respectively. `PA4BNP` algorithm obtains preliminary promising computational results (see Table 1).

	Couenne		SCIP		PA4BNP+LS1		PA4BNP+LS2		PA4BNP+LS3	
instance	obj	CPU	obj	CPU	obj	CPU	obj	CPU	obj	CPU
20_20_b	-959.25	T.L.	fail	0.15	-984.18	3.00	-984.18	2.00	-984.18	3.00
20_50_b	–	T.L.	–	T.L.	-959.25	3.01	-959.25	6.03	-959.25	2.01
20_80_b	–	T.L.	–	T.L.	-69.96	3.01	-71.32	4.01	-69.963	3.01
20_100_b	–	T.L.	–	T.L.	-710.87	3.01	-710.87	4.01	-710.87	4.01
40_20_b	-984.18	T.L.	fail	0.42	-984.18	75.01	-984.18	206.03	-984.181	76.01
40_50_b	–	T.L.	fail	2.34	-959.25	120.02	-959.25	235.09	-959.25	129.02
40_80_b	–	T.L.	–	T.L.	-71.63	72.01	-71.29	112.01	-71.63	72.01
40_100_b	–	T.L.	–	T.L.	-972.92	127.02	-972.92	127.03	-972.92	128.03
20_20_c	-11.71	T.L.	–	T.L.	-165.87	4.00	-165.87	6.01	-165.87	2.00
20_50_c	-73.76	T.L.	–	T.L.	-149.74	6.01	-149.74	4.01	-149.74	6.01
20_80_c	–	T.L.	–	T.L.	-65.72	3.01	-65.72	4.02	-65.72	4.02
20_100_c	–	T.L.	–	T.L.	-38.14	2.00	-35.50	5.02	-38.14	4.00
40_20_c	-42.35	T.L.	fail	86.99	-133.43	78.00	-133.43	118.01	-133.43	76.00
40_50_c	–	T.L.	–	T.L.	-131.25	121.01	-131.25	167.03	-131.25	120.01
40_80_c	–	T.L.	–	T.L.	-58.09	111.01	-58.09	114.02	-58.09	110.01
40_100_c	–	T.L.	–	T.L.	-34.89	200.03	-34.89	110.03	-34.89	198.03

TAB. 1: Objective values and CPU times in sec. T.L. time limit. "–" no feasible solution within T.L.

References

[1] C.A. Kieslich, F. Boukouvala, and C.A. Floudas. Optimization of black-box problems using Smolyak grids and polynomial approximations. *Journal of Global Optimization*, 71:845–869, 2018.

[2] lagrange_nd. `https://github.com/johannesgerer/jburkardt-cpp/tree/master/lagrange_nd`.

[3] P.J. Olver. On multivariate interpolation. *Studies in Applied Mathematics*, 116(4):201–240, 2006.

[4] F. Schoen. A wide class of test functions for global optimization. *Journal of Global Optimization*, 3:133–137, 1993.

[5] SmolyakApprox.jl. `https://github.com/RJDennis/SmolyakApprox.jl`.