Quadratic Reformulation of Nonlinear Pseudo-Boolean Functions via the Constraint Composite Graph

Ka Wa Yip(✉), Hong Xu(✉), Sven Koenig, and T. K. Satish Kumar

University of Southern California, Los Angeles, CA 90089, United States of America
{kawayip, hongx, skoenig}@usc.edu, tkskwork@gmail.com

Abstract. *Nonlinear pseudo-Boolean optimization* (nonlinear PBO) is the minimization problem on *nonlinear pseudo-Boolean functions* (nonlinear PBFs). One promising approach to nonlinear PBO is to first use a *quadratization algorithm* to reduce the PBF to a quadratic PBF by introducing intelligently chosen auxiliary variables and then solve it using a quadratic PBO solver. In this paper, we develop a new quadratization algorithm based on the idea of the *constraint composite graph* (CCG). We demonstrate its theoretical advantages over state-of-the-art quadratization algorithms. We experimentally demonstrate that our CCG-based quadratization algorithm outperforms the state-of-the-art algorithms in terms of both effectiveness and efficiency on randomly generated instances and a novel reformulation of the uncapacitated facility location problem.

1 Introduction

Nonlinear pseudo-Boolean optimization (nonlinear PBO) refers to the minimization problem on *nonlinear pseudo-Boolean functions* (nonlinear PBFs). Formally, a PBF is a mapping $f : \mathbb{B}^n \to \mathbb{R}$ that maps each assignment of values to a set of Boolean variables to a real number. The Boolean variables are restricted to take a value in $\mathbb{B} = \{0, 1\}$. A PBF is nonlinear iff it cannot be reformulated as a linear combination of the Boolean variables. (Nonlinear) PBO asks for an assignment of values to the Boolean variables that minimizes the value of a (nonlinear) PBF, i.e., it is the task of computing $\arg\min_{\boldsymbol{x}\in\mathbb{B}^n} f(\boldsymbol{x})$. Nonlinear PBO is known to be NP-hard and subsumes many classic optimization problems, such as MAX-SAT and MAX-CUT [10]. It has been used in many real-world applications, such as computer vision [26], operations research and traffic planning [15, 18, 37], chip design [11], evolutionary computation [38], and spin-glass models [25].

While a lot of research has concentrated on linear PBO (with linear constraints) [1, 8, 13, 23, 31, 33, 35], nonlinear PBO has not been very well studied. There only exist a handful of techniques dedicated to nonlinear PBO, such as reformulation to 0/1 *integer linear programming* (ILP) [16], constraint integer

The research at the University of Southern California was supported by NSF under grant numbers 1724392, 1409987, 1817189, and 1837779.

programming [6], constraint logic programming [7], and graph cuts [26]. Among these techniques, the most viable approach involves *quadratization algorithms*, i.e., reformulating the PBF as a *quadratic* PBF [3, 21]. A PBF is quadratic iff it is a sum of monomials in which each monomial is a product of at most two Boolean variables. Several authors have pointed out the benefits of quadratization algorithms over algorithms based on linearization and ILP [4, 10, 17].

Quadratization algorithms are not only the most viable approach to PBO, but also the only viable approach that serves some fundamental purposes. For example, quantum annealers—such as the D-Wave chips—can solve only quadratic unconstrained Boolean optimization problems [22]. Therefore, a quadratization algorithm is indispensable for solving nonlinear PBO instances on quantum annealers. In addition, no existing weighted MAX-SAT or ILP solver can solve nonlinear PBO instances with arbitrary lengths of monomials. They, including BiqMac [36], are only applicable to unconstrained binary quadratic programs. Quadratization algorithms are therefore required for reformulating nonlinear PBO instances to make them amenable to such solvers. In general, quadratization algorithms are useful due to the existence of more efficient algorithms that are dedicated to minimizing quadratic PBFs, i.e., to *quadratic PBO* (QPBO). For example, a QPBO solver can make use of the peculiar properties of quadratic PBFs, such as their *roof duality* [24] and the existence of polynomial-time algorithms for finding partial solutions even if the PBFs are not submodular [21].

Formally, a *quadratization* of a PBF $f(\boldsymbol{x})$ is a quadratic PBF $g(\boldsymbol{x}, \boldsymbol{y})$ such that

$$f(\boldsymbol{x}) = \min_{\boldsymbol{y} \in \mathbb{B}^m} g(\boldsymbol{x}, \boldsymbol{y}) \quad \forall \boldsymbol{x} \in \mathbb{B}^n, \tag{1}$$

where $\boldsymbol{y}$ is a set of m auxiliary Boolean variables. Since minimizing a quadratic PBF is also NP-hard, quadratization algorithms should preferably be achieved in polynomial time using a small number of auxiliary variables. Since the number of variables largely determines the size of the search space, existing algorithms focus on minimizing the number of auxiliary variables [3, 21].

In this paper, we develop a new polynomial-time quadratization algorithm based on the *constraint composite graph* (CCG) [28–30]. We show that our CCG-based quadratization algorithm has a theoretical advantage over the state-of-the-art algorithms proposed in [21]. We also experimentally demonstrate that our CCG-based quadratization algorithm outperforms the state-of-the-art quadratization algorithms in terms of the required number of auxiliary variables, the number of terms in the quadratization, and the runtime. We conduct experiments on both randomly generated instances and a novel reformulation of the uncapacitated facility location problem.

2 Preliminaries

In this section, we give a brief background on quadratizations and the CCG.

2.1 Quadratizations

A PBF is a function that maps n Boolean variables to a real number. As proved in [18], any PBF f of n Boolean variables $\boldsymbol{x} = \{x_1, \ldots, x_n\}$ can be uniquely represented as a polynomial of the form

$$f(\boldsymbol{x}) = \sum_{S \subseteq \boldsymbol{x}} c_S \prod_{x \in S} x, \tag{2}$$

where $c_S \in \mathbb{R}$. Throughout this paper, we specify all PBFs in this form. We let d denote the degree of a PBF, i.e., the maximum degree of all its monomials.

A quadratization of a PBF $f(\cdot)$ is a quadratic PBF $g(\cdot)$ that satisfies Eq. (1). For any given PBF, its quadratizations exist but are not necessarily unique. Since a quadratization algorithm can be seen as a preprocessing algorithm, its effectiveness can be evaluated using two metrics: the number of auxiliary variables and the number of terms in $g(\cdot)$, which usually are good indicators of the time required to solve the resulting quadratic PBF using a QPBO solver.

In terms of the number of auxiliary variables in $g(\cdot)$, some current state-of-the-art algorithms are given in [21]. The first algorithm is called *polynomial expansion* and is polynomial-time. It first *quadratizes*, i.e. finds a quadratization of, each monomial in $f(\cdot)$ individually and then combines all like quadratic terms. Polynomial expansion quadratizes a monomial $ax_1 \ldots x_d$ of degree $d > 2$ to

$$\begin{cases} \min\limits_{w \in \mathbb{B}} aw\left(S_1 - (d-1)\right) & \text{if } a < 0 \\ \min\limits_{\{w_1, \ldots, w_{n_d}\} \in \mathbb{B}^{n_d}} \sum\limits_{i=1}^{n_d} w_i \left(c_{i,d}(-S_1 + 2i) - 1\right) + aS_2 & \text{if } a > 0, \end{cases} \tag{3}$$

where

$$S_1 = \sum_{i=1}^{d} x_i \qquad S_2 = \sum_{i=1}^{d-1} \sum_{j=i+1}^{d} x_i x_j = \frac{S_1(S_1-1)}{2}$$

$$n_d = \left\lfloor \frac{d-1}{2} \right\rfloor \qquad c_{i,d} = \begin{cases} 1 & \text{if } i = n_d \text{ and } d \text{ is odd} \\ 2 & \text{otherwise.} \end{cases}$$

Therefore, if $a < 0$, quadratizing this monomial requires 1 auxiliary variable; if $a > 0$, it requires n_d auxiliary variables[1].

The second algorithm is called γ *flipping*. Let $\boldsymbol{\gamma} = \{\gamma_1, \ldots, \gamma_n\} \in \mathbb{B}^n$ and $x_i^{(\boldsymbol{\gamma})} = \gamma_i x_i + \bar{\gamma}_i \bar{x}_i = \begin{cases} x_i & \text{if } \gamma_i = 1 \\ \bar{x}_i & \text{if } \gamma_i = 0. \end{cases}$ Then, we have

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{\gamma} \in \mathbb{B}^n} f(\boldsymbol{\gamma}) x_1^{(\boldsymbol{\gamma})} \ldots x_n^{(\boldsymbol{\gamma})} = \lambda + \sum_{\boldsymbol{\gamma} \in \mathbb{B}^n} (f(\boldsymbol{\gamma}) - \lambda) x_1^{(\boldsymbol{\gamma})} \ldots x_n^{(\boldsymbol{\gamma})}. \tag{4}$$

When using

$$\lambda = \max_{\boldsymbol{\gamma} \in \mathbb{B}^n} f(\boldsymbol{\gamma}), \tag{5}$$

[1] For a single positive monomial, the smallest possible number of auxiliary variables achievable is $\lceil \log d \rceil - 1$, as proven in [9].

every monomial in Eq. (4) is non-positive. Then, by following the first case in Eq. (3), γ flipping requires exactly $2^n - 1$ auxiliary variables. This is superpolynomial with respect to the size of input if the number of terms in Eq. (2) is, for example, polynomial with respect to n.

The computation of Eq. (5) is the bottleneck. Its time complexity is superpolynomial with respect to n. Hence, γ flipping is superpolynomial-time if the number of terms in Eq. (2) is, for example, polynomial with respect to n.

2.2 Constraint Composite Graph

The CCG [28–30] is a combinatorial structure associated with an optimization problem posed as the *weighted constraint satisfaction problem* (WCSP). It simultaneously represents the graphical structure of the variable interactions in the WCSP and the numerical structure of the constraints in it. The task of solving the WCSP can be reformulated as the task of finding a *minimum weighted vertex cover* (MWVC) (called the MWVC problem) on its associated CCG. CCGs can be constructed in polynomial time and are always tripartite. A subclass of the WCSP has instances with bipartite CCGs. This subclass is tractable since an MWVC can be found in polynomial time on bipartite graphs using a maxflow algorithm [27]. The CCG also facilitates kernelization, message passing [14, 40], and an efficient encoding of the WCSP as an integer linear program [39].

Given an undirected graph $G = \langle V, E \rangle$, a vertex cover of G is a set of vertices $S \subseteq V$ such that every edge in E has at least one of its endpoint vertices in S. A *minimum vertex cover* (MVC) of G is a vertex cover of minimum cardinality. When G is vertex-weighted—i.e., each vertex $v_i \in V$ has a non-negative weight w_i associated with it—its MWVC is defined as a vertex cover of minimum total weight of its vertices. The MWVC problem is the task of computing an MWVC on a given vertex-weighted undirected graph.

For a given graph G, the concept of the MWVC problem can be extended to the notion of projecting MWVCs onto a given *independent set* (IS) $U \subseteq V$. (An IS of G is a set of vertices in which no two vertices are adjacent to each other.) The input to such a projection is the graph G as well as an IS $U = \{u_1, u_2, \ldots, u_k\}$. The output is a table of 2^k numbers. Each entry in this table corresponds to a k-bit vector. We say that a k-bit vector t imposes the following restrictions: (i) if the i^{th} bit t_i is 0, the vertex u_i has to be excluded from the MWVC; and (ii) if the i^{th} bit t_i is 1, the vertex u_i has to be included in the MWVC. The projection of the MWVC problem onto the IS U is then defined to be a table with entries corresponding to each of the 2^k possible k-bit vectors $t^{(1)}, t^{(2)}, \ldots, t^{(2^k)}$. The value of the entry corresponding to $t^{(j)}$ is equal to the weight of the MWVC conditioned on the restrictions imposed by $t^{(j)}$. [28, Fig. 2] illustrates this projection.

The table of numbers produced above can be viewed as a weighted constraint over $|U|$ Boolean variables. Conversely, given a (Boolean) weighted constraint, we design a lifted representation for it so as to be able to view it as the projection of an MWVC onto an IS of some intelligently constructed vertex-weighted undirected graph [28, 29]. The benefit of constructing these representations for

individual constraints lies in the fact that the lifted representation for the entire WCSP, i.e., the CCG of the WCSP, can be obtained simply by "merging" them.

[28, Fig. 5] shows an example WCSP instance over 3 Boolean variables to illustrate the construction of the CCG. Here, there are 3 unary and 3 binary weighted constraints. Their lifted representations are shown next to them. The figure also illustrates how the CCG is obtained from the lifted representations of the weighted constraints: In the CCG, vertices that represent the same variable are simply "merged"—along with their edges—and every "composite" vertex is given a weight equal to the sum of the individual weights of the merged vertices. Computing the MWVC for the CCG yields a solution for the WCSP instance; namely, if X_i is in the MWVC, then it is assigned value 1 in the WCSP instance, otherwise it is assigned value 0 in the WCSP instance.

3 The CCG-Based Quadratization Algorithm

PBO is a special case of the WCSP and is therefore equivalent to solving the MWVC problem on its associated CCG. In turn, we show that the MWVC problem itself can be reformulated as QPBO. This leads to the CCG-based quadratization algorithm presented in this section.

Given a vertex-weighted graph $G = \langle V, E, w\rangle$ and one of its independent sets T, the projection of the MWVC problem onto T is a table of weights of MWVCs with all combinations of vertices in T imposed to be included in or excluded from the MWVC [28]. More formally:

Definition 1. *Let $T_+ \cup T_- = T$ and $T_+ \cap T_- = \emptyset$. S is a para-vertex cover on $\langle G, T_+, T_-\rangle$ iff S is a vertex cover on G, $T_+ \subseteq S$, and $T_- \cap S = \emptyset$. S is a para-MWVC on $\langle G, T_+, T_-\rangle$ iff S is a para-vertex cover on $\langle G, T_+, T_-\rangle$ and the sum of weights of all vertices in S is no greater than that of any other para-vertex cover on $\langle G, T_+, T_-\rangle$. The projection of the MWVC problem onto T (on G) is a function that maps $\langle T_+, T_-\rangle$ to the weight of a para-MWVC on $\langle G, T_+, T_-\rangle$.*

The following theorem is inspired by [12, Theorem 3].

Theorem 1. *Let us consider the finite graph $G = \langle V, E, w\rangle$ and an independent set $T = T_+ \cup T_-$ on it. Let $\boldsymbol{x} = (x_r : r \in V)$ and*

$$\mathcal{C}(\boldsymbol{x}) = \sum_{p\in V} w_p x_p + \sum_{(p,q)\in E} J_{pq}(1 - x_p)(1 - x_q). \tag{6}$$

(i) If $\forall (p, q) \in E : J_{pq} \geq \max\{w_p, w_q\}$, then the projection of the MWVC problem onto an independent set T equals the function

$$h(\langle T_+, T_-\rangle) = \min_{x_j\in\mathbb{B}: j\in V\setminus T} \mathcal{C}(\boldsymbol{x})\big|_{\substack{x_i=1 \quad \text{if } i\in T_+ \\ x_i=0 \quad \text{if } i\in T_-}}. \tag{7}$$

(ii) If further $\forall (p, q) \in E : J_{pq} > \max\{w_p, w_q\}$, then any $S^* \subset V$ that satisfies

$$T_+ \subseteq S^* \tag{8}$$

$$T_- \cap S^* = \emptyset \tag{9}$$

$$\mathscr{C}(S^*) = h(\langle T_+, T_-\rangle) \tag{10}$$

is a para-MWVC on $\langle G, T_+, T_- \rangle$*, with* $\mathscr{C}(\cdot)$ *defined as* $\mathscr{C}(S) = \mathcal{C}(\boldsymbol{x})\big|_{\substack{x_i=1 \quad \text{if } i\in S \\ x_i=0 \quad \text{if } i\notin S}}$.

Proof. Let us consider a given $\langle T_+, T_- \rangle$. We first prove (ii), then (i).

For (ii): We first prove by contradiction that, if $J_{pq} > \max\{w_p, w_q\}$, then S^* is a vertex cover. Let $x_i^* = 1$ if $i \in S^*$ and $x_i^* = 0$ if $i \in V \backslash S^*$, and $\boldsymbol{x}^* = (x_i^* : i \in V)$. We assume that there exists an edge (a, b) such that $x_a^* = x_b^* = 0$. Neither a nor b can be in T_+ because $T_+ \subseteq S^*$. Since T is an independent set, a and b cannot be both in T_-. If we hold either of the rest cases, i.e.,

- if only one of a and b is in T_- (without loss of generality, we let $a \in T_-$), or
- if neither a nor b is in T_-,

then $a \notin S^*$ and $\mathscr{C}(S^*) - \mathscr{C}(S^* \cup \{b\}) = \sum_{k\notin S^*:(b,k)\in E} J_{bk} - w_b \geq J_{ab} - w_b > 0$, which contradicts Eq. (10).

In addition, S^* is also a para-MWVC on $\langle G, T_+, T_- \rangle$ because S^* being a vertex cover implies $\sum_{(p,q)\in E} J_{pq}(1 - x_p^*)(1 - x_q^*) = 0$. Therefore, (ii) holds.

For (i): Let $S^{*\prime}$ be a para-MWVC on $\langle G, T_+, T_- \rangle$. Because $S^{*\prime}$ is a vertex cover, the second summation of Eq. (6) in $\mathscr{C}(S^{*\prime})$ equals zero and thus the weight of $S^{*\prime}$ equals $\mathscr{C}(S^{*\prime})$. Therefore, it is sufficient to prove that there exists such an $S^{*\prime}$ that satisfies $\mathscr{C}(S^{*\prime}) = h(\langle T_+, T_- \rangle)$, or, equivalently, $\mathscr{C}(S^{*\prime}) = \mathscr{C}(S^*)$.

If $\mathcal{C}(\cdot)$ is a constant function, then it is obvious that $\mathscr{C}(S^{*\prime}) = h(\langle T_+, T_- \rangle)$. We now consider the case where $\mathcal{C}(\cdot)$ is not a constant function. Let $E' = \{(p, q) \in E : J_{pq} = \max\{w_p, w_q\}\}$. Let

$$\mathcal{C}'(\boldsymbol{x}) = \sum_{p\in V} w_p x_p + \sum_{(p,q)\in E} J'_{pq}(1 - x_p)(1 - x_q). \tag{11}$$

Here, $J'_{pq} = \begin{cases} J_{pq} & \text{if } (p, q) \notin E' \\ J_{pq} + \epsilon_{pq} & \text{if } (p, q) \in E' \end{cases}$, where $\forall (p, q) \in E' : \epsilon_{pq} > 0$ and they satisfy

$$\sum_{(p,q)\in E'} \epsilon_{pq} < \epsilon_0, \tag{12}$$

where ϵ_0 is the smallest positive value that $\mathcal{C}(\boldsymbol{x}) - \mathcal{C}(\boldsymbol{y})$ can be for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{B}^{|V|}$, i.e., $\epsilon_0 = \min\{\mathcal{C}(\boldsymbol{x}) - \mathcal{C}(\boldsymbol{y}) \in \mathbb{R}_{>0} : \boldsymbol{x}, \boldsymbol{y} \in \mathbb{B}^{|V|}\}$. Here, the operand of min cannot be $\emptyset$ because $\mathcal{C}(\cdot)$ is not a constant function.

($\bigstar$) Let $S^{*\prime} \subseteq V$ satisfy Eqs. (8) to (10) except that $\mathscr{C}(\cdot)$ in Eq. (10) is replaced by $\mathscr{C}'(\cdot)$, defined as $\mathscr{C}'(S) = \mathcal{C}'(\boldsymbol{x})\big|_{\substack{x_i=1 \quad \text{if } i\in S \\ x_i=0 \quad \text{if } i\notin S}}$, and all occurrences of $\mathcal{C}(\cdot)$ are replaced by $\mathcal{C}'(\cdot)$. According to (ii), $S^{*\prime}$ is a para-MWVC on $\langle G, T_+, T_- \rangle$. Let $x_i^{*\prime} = 1$ if $i \in S^{*\prime}$ and $x_i^{*\prime} = 0$ if $i \in V \setminus S^{*\prime}$, and $\boldsymbol{x}^{*\prime} = (x_i^{*\prime} : i \in V)$. Now we only need to prove $\mathscr{C}(S^{*\prime}) = \mathscr{C}(S^*)$, or, equivalently, $\mathcal{C}(\boldsymbol{x}^{*\prime}) = \mathcal{C}(\boldsymbol{x}^*)$.

According to Eq. (10), $\mathcal{C}(\boldsymbol{x}^*) = h(\langle T_+, T_- \rangle)$. Therefore, $\mathcal{C}(\boldsymbol{x}^{*\prime}) \geq \mathcal{C}(\boldsymbol{x}^*)$. We now only need to prove that $\mathcal{C}(\boldsymbol{x}^{*\prime}) > \mathcal{C}(\boldsymbol{x}^*)$ cannot hold. We prove by contradiction. Assume $\mathcal{C}(\boldsymbol{x}^{*\prime}) > \mathcal{C}(\boldsymbol{x}^*)$. Then, according to the definition of ϵ_0, $\mathcal{C}(\boldsymbol{x}^{*\prime}) - \mathcal{C}(\boldsymbol{x}^*) \geq \epsilon_0$. According to Eqs. (11) and (12), $\mathcal{C}(\boldsymbol{x}^*) - \mathcal{C}'(\boldsymbol{x}^*) \geq -\sum_{(p,q)\in E'} \epsilon_{pq} > -\epsilon_0$. According to Eq. (11), $\mathcal{C}'(\boldsymbol{x}^{*\prime}) - \mathcal{C}(\boldsymbol{x}^{*\prime}) \geq 0$. Adding these three inequalities, we have $\mathcal{C}'(\boldsymbol{x}^{*\prime}) - \mathcal{C}'(\boldsymbol{x}^*) > 0$, and thus $\mathscr{C}'(S^{*\prime}) > \mathscr{C}'(S^*)$. This contradicts Eq. (10) after the replacements in ($\bigstar$). □

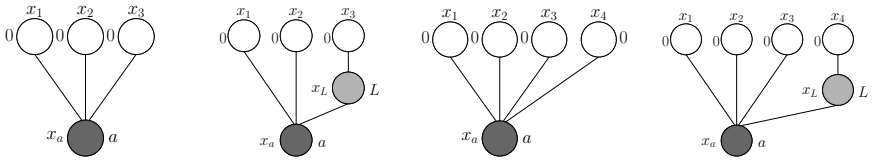


Fig. 1: The graph gadgets for the construction of the CCG. Each vertex is associated with a weight and a label. "x_a" and "x_L" are the labels of the auxiliary variables.

Based on Theorem 1, we outline a quadratization algorithm for a (nonlinear) PBO—or more generally for a WCSP—as follows: (i) Using the polynomial-time algorithm proposed in [28], reformulate the input PBF $f(\boldsymbol{x})$ to the projection of the MWVC problem on its CCG; and (ii) using Theorem 1, convert this projection to a quadratic PBF and output it as its quadratization.

3.1 A Full Example

Consider the PBF

$$P(x_1, x_2, x_3, x_4) = 3x_2x_3 + 5x_1x_2x_3 + 6x_1x_2x_3x_4 - 3x_1x_3x_4\,. \tag{13}$$

The CCG is a composition of graph gadgets, each of which represents a monomial [28]. Each monomial is related to an MWVC of a particular graph gadget (Fig. 1). Assume $a > 0$ (throughout this subsection), for a monomial $-ax_1x_2x_3$, $\mathsf{MWVC}\{\text{Fig. } 1a\} = a - ax_1x_2x_3$, where $\mathsf{MWVC}\{\text{Fig. } 1a\}$ is the weight of the MWVCs of Fig. 1a, i.e.,

$$-ax_1x_2x_3 = \mathsf{MWVC}\{\text{Fig. } 1a\} - a\,. \tag{14}$$

For $ax_1x_2x_3$, $\mathsf{MWVC}\{\text{Fig. } 1b\} = L(1-x_3)+a-a(x_1x_2(1-x_3))$, for a sufficiently large constant L, i.e.,

$$ax_1x_2x_3 = \mathsf{MWVC}\{\text{Fig. } 1b\} - L(1-x_3) - a + ax_1x_2\,. \tag{15}$$

For $ax_1x_2x_3x_4$, $\mathsf{MWVC}\{\text{Fig. } 1d\} = L(1-x_4) + a - a(x_1x_2x_3(1-x_4))$, i.e.,

$$ax_1x_2x_3x_4 = \mathsf{MWVC}\{\text{Fig. } 1d\} - L(1-x_4) - a + ax_1x_2x_3\,, \tag{16}$$

where $ax_1x_2x_3$ can be rewritten as in Eq. (15). Here, all monomials of degree > 2 have been rewritten as quadratic PBFs and weights of MWVCs of graph gadgets.

Applying Theorem 1 and setting $J \geq L > a > 0$, we further express the weights of MWVCs in algebraic quadratic forms as

$$\begin{aligned}\mathsf{MWVC}\{\text{Fig. } 1a\} = \min_{x_a}[&ax_a + J(1-x_1)(1-x_a)\\ &+J(1-x_2)(1-x_a) + J(1-x_3)(1-x_a)]\end{aligned} \tag{17}$$

$$\begin{aligned}\mathsf{MWVC}\,\{\text{Fig. } 1b\} = \min_{x_a, x_L}\,[&ax_a + Lx_L + J(1-x_1)(1-x_a)\\ &+J(1-x_2)(1-x_a)+J(1-x_3)(1-x_L) + J(1-x_L)(1-x_a)]\end{aligned} \tag{18}$$

$$\begin{aligned}\mathsf{MWVC}\,\{\text{Fig. } 1c\} = \min_{x_a}[&ax_a + J(1-x_1)(1-x_a)\\ &+J(1-x_2)(1-x_a)+J(1-x_3)(1-x_a) + J(1-x_4)(1-x_a)]\end{aligned} \tag{19}$$

$$\begin{aligned}\mathsf{MWVC}\,\{\text{Fig. } 1d\} = \min_{x_a, x_L}\,[&ax_a + Lx_L + J(1-x_1)(1-x_a)\\ &+J(1-x_2)(1-x_a) + J(1-x_3)(1-x_a)\\ &+J(1-x_4)(1-x_L) + J(1-x_L)(1-x_a)]\,.\end{aligned} \tag{20}$$

Here, we have quadratized all monomials of degree > 2 using the algebraic expression of the weight of MWVCs on its graph gadget. The auxiliary variables are named uniquely for each graph gadget. For the PBF in Eq. (13), we therefore need 5 auxiliary variables: x_a and x_L for the degree-4 term, $x_{a'}$ and $x_{L'}$ for the degree-3 term with positive coefficient that combines the existing degree-3 term with the degree-3 term that comes from the reduction of the degree-4 term, and $x_{a''}$ for the degree-3 term with negative coefficient.

3.2 Details of the CCG-Based Quadratization Algorithm

The CCG-based quadratization algorithm is an iterative algorithm. Let $f(\boldsymbol{x})$ be the input PBF. It initializes a polynomial $f'(\boldsymbol{x})$ to $f(\boldsymbol{x})$. In each iteration, let d be the degree of $f'(\boldsymbol{x})$. It substitutes each degree-d negative monomial and positive monomial in $f'(\boldsymbol{x})$, respectively, using

$$-ax_1 \dots x_d = \min_{x_a}\left[ax_a + J\sum_{i=1}^{d}(1-x_i)(1-x_a)\right] - a \tag{21}$$

$$\begin{aligned}ax_1 \dots x_d = \min_{x_a, x_L}\Bigg[&ax_a + Lx_L + J\sum_{i=1}^{d-1}(1-x_i)(1-x_a)\\ &+ J(1-x_d)(1-x_L) + J(1-x_L)(1-x_a)\Bigg]\\ &-L(1-x_d) - a + ax_1 \dots x_{d-1},\end{aligned} \tag{22}$$

where $J \geq L > a > 0$. It then combines all like terms in $f'(\boldsymbol{x})$. Because the right-hand sides of Eqs. (21) and (22) are of degrees that are lower than the left-hand sides, the degree of $f'(\boldsymbol{x})$ decreases by at least 1 after each iteration. The iterating procedure terminates until the degree of $f'(\boldsymbol{x})$ is no larger than 2. Finally, the algorithm outputs $f'(\boldsymbol{x})$ as the quadratization.

4 Evaluation

In this section, we evaluate our CCG-based quadratization algorithm both theoretically and experimentally, and illustrate its uses and advantages on a real-world problem.

4.1 Theoretical Results

In this subsection, we theoretically compare our CCG-based quadratization algorithm with the state-of-the-art algorithms in [21] in terms of the number of auxiliary variables. For any PBF of degree d on n variables, the maximum number of non-zero monomials of each degree $i \leq d$ is $\binom{n}{i}$. For polynomial expansion, the worst case occurs when all coefficients of non-zero monomials are positive[2]. From Eq. (3), each positive monomial of degree $i \geq 3$ generates $\left\lfloor \frac{i-1}{2} \right\rfloor$ auxiliary variables. Therefore, the number of auxiliary variables in the worst case is $N^{\text{all+}}(n,d) = \sum_{i=3}^{d} \binom{n}{i} \left\lfloor \frac{i-1}{2} \right\rfloor = O\left(\left\lfloor \frac{\hat{d}-1}{2} \right\rfloor \frac{n!}{\hat{d}!(n-\hat{d})!} \right) = O\left(\left\lfloor \frac{\hat{d}-1}{2} \right\rfloor \frac{n^{\hat{d}}}{\hat{d}!} \right)$, where $\hat{d} = \min\{\lceil n/2 \rceil, d\}$ and the expression is with respect to asymptotically large n.

For the CCG-based quadratization algorithm, the worst case also occurs when all monomials are positive. Each positive monomial of degree $i \geq 3$ generates 2 auxiliary variables (i.e., X_a and X_L in Eq. (22)) when it is reduced to the sum of a quadratic polynomial and a monomial of degree $i-1$, which can then be combined with existing monomials of degree $i-1$ if they are composed of the same variables. This combination of monomials can take place in each iteration, until the whole PBF becomes quadratic. In the worst case, only positive monomials remain after the combining step of each iteration, and therefore the number of auxiliary variables is $\sum_{i=3}^{d} 2\binom{n}{i} = O\left(\frac{n!}{\hat{d}!(n-\hat{d})!} \right) = O\left(\frac{n^{\hat{d}}}{\hat{d}!} \right)$.

In the best case, where all monomials are negative (assuming that all monomials up to degree d are present), both polynomial expansion and the CCG-based quadratization algorithm need just one auxiliary variable for each monomial, i.e., $N^{\text{all-}}(n,d) = \sum_{i=3}^{d} \binom{n}{i} = O\left(\frac{n!}{\hat{d}!(n-\hat{d})!} \right) = O\left(\frac{n^{\hat{d}}}{\hat{d}!} \right)$.

Table 1 summarizes our theoretical results. It shows that the CCG-based quadratization algorithm is advantageous over both polynomial expansion and γ flipping in terms of the required number of auxiliary variables. γ flipping has the same complexity regardless of the number of monomials in the input PBF, which is undesirable for PBFs that do not have an exponential number of monomials. In the best case, the numbers of auxiliary variables required by polynomial expansion and the CCG-based quadratization algorithm are only polynomial in n, while that of γ flipping is exponential in n.

4.2 Experimental Results

In this subsection, we focus on an experimental comparison of polynomial expansion and the CCG-based quadratization algorithm, since both of them have

[2] We follow the worst case definition in [21].

Table 1: Number of auxiliary variables for different quadratization algorithms as a function of the number of variables n and $\hat{d} = \min\{\lceil n/2 \rceil, d\}$, where d is the degree of the PBF.

	if $n \neq d$	if $n = d$
Polynomial expansion (worst case)	$O\left(\left\lfloor \frac{\hat{d}-1}{2} \right\rfloor \frac{n^{\hat{d}}}{\hat{d}!}\right)$	$2^{d-2}(d-3)+1$
CCG-based (worst case)	$O\left(\frac{n^{\hat{d}}}{\hat{d}!}\right)$	$2^{d+1}-2-2d-d(d-1)$
Polynomial expansion and CCG-based (best case)	$O\left(\frac{n^{\hat{d}}}{\hat{d}!}\right)$	$2^d - 1 - \frac{d(d+1)}{2}$
γ flipping	$2^n - 1$	$2^d - 1$

Table 2: Number of auxiliary variables, number of terms in the quadratization, runtime of the quadratization algorithm, and runtime of the QPBO solver for the minimization of the quadratization. All reported numbers are averaged over 10 instances with $n = d$ and $m = 2^d$ monomials. Numbers after $\pm$ are standard deviations. The monomial coefficients are integers chosen randomly from $[1, 300]$. The smaller numbers of auxiliary variables and terms of quadratizations of each column are highlighted.

d		3	10	11	12	13	14	15
original number of terms		8	1024	2048	4096	8192	16384	32758
number of auxiliary variables	Poly	**1**	**1793**	4097	9217	20481	45057	>24 hrs
	CCG	2	1936	**3962**	**8034**	**16200**	**32556**	65294
number of terms	Poly	**11**	12089	29508	70736	167005	389227	>24 hrs
	CCG	14	**7988**	**17162**	**36572**	**77482**	**163444**	343610
quadratization time (s)[3]	Poly	0.0006	10.87 ±0.348	58.84 ±2.43	341.19 ±27.87	3435.50 ±39.34	17241.10 ±98.35	>24 hrs
	CCG	0.0006	1.828 ±0.007	5.133 ±0.006	23.16 ±0.10	104.58 ±1.94	584.33 ±21.90	3396.50 ±9.63
QPBO (s)[4]	Poly	0.0197 ±0.003	0.0835 ±0.0023	0.5086 ±0.0090	2.7316 ±0.081	14.5621 ±1.01	88.362 ±5.12	>24 hrs
	CCG	0.0080 ±0.00049	0.0043 ±0.00024	0.0132 ±0.00082	0.0616 ±0.0018	0.2968 ±0.0065	1.3018 ±0.0333	5.5763 ±0.5098

time complexities that are polynomial in d. We implement both algorithms in Python 2.7. Although *pseudo-Boolean* (PB) competitions have been regularly held [32], none of their instances have objective functions that are nonlinear PBFs. Therefore, we generate our own instances. We experiment with random instances and instances that model real-world facility location problems. These instances have a range of d wider than that of the problem of image denoising used in [21], which always has $d = 4$.

To generate random instances with a PBF on n variables, we generate each monomial with degree i randomly chosen from $\{0, \ldots, d\}$. Such a monomial has i unique variables randomly chosen from $\{1, \ldots, n\}$ along with a non-zero random integer coefficient. If a newly generated monomial is on the same variables as

[3]Intel Xeon 4-core 2.3 GHz/6-core 2.6 GHz

[4]Intel Core i7-4960HQ Processor 6M Cache 2.60 GHz 8 GB SDRAM

Table 3: Similar to Table 2, except that the monomial coefficients are non-zero integers chosen randomly from $[-300, 300]$.

d		3	10	11	12	13	14	15
original number of terms		8	1024	2048	4096	8192	16384	32758
number of auxiliary variables	Poly	**1**	**1371.7** ±13.50	**3022.1** ±28.75	6630.3 ±39.16	14317 ±43.35	30580 ±111	65623 ±193
	CCG	1.5 ±0.52	1545 ±14.5	3182 ±18.3	**6503** ±25.9	**13091** ±49.9	**26397** ±41.1	**52908** ±70
number of terms	Poly	**11**	9002.7 ±120.2	21205.8 ±227.1	49758.7 ±379.4	114482 ±493.3	259370 ±1055	588832 ±2111
	CCG	12.5 ±1.58	**7205.4** ±28.2	**15601** ±36.3	**33504** ±51.9	**71251** ±81.3	**151089** ±75.8	**318781** ±150
quadratization time (s)[3]	Poly	0.0006	4.01 ±0.18	23.22 ±0.91	133.5 ±0.71	1048.8 ±3.20	5724.5 ±747.7	84538 ±788.1
	CCG	0.0006	0.9516 ±0.011	4.21 ±0.022	19.14 ±0.057	133.9 ±0.58	594.8 ±73	3046 ±475
QPBO (s)[4]	Poly	0.0196 ±0.0012	0.0420 ±0.0094	0.27 ±0.0536	1.67 ±0.3289	8.43 ±0.0380	43.2 ±7.67	376.1 ±15.2
	CCG	0.0071 ±0.00021	0.0049 ±0.00052	0.0155 ±0.0025	0.058 ±0.011	0.37 ±0.010	1.10 ±0.26	4.38 ±0.90

those of already generated monomials, it is rejected and a new one is generated. We also check that at least one of the m terms generated in this way is of degree d. For polynomial expansion, we also add up all quadratic like terms in the resulting quadratization (which can be expensive since polynomial expansion can potentially generate a lot of like terms in Eq. (3)). We use the QPBO solver from [26] for the minimization of the quadratizations. We run it via the open-source MATLAB wrapper qpboMex [34] on MATLAB R2016a and measure the actual wall-clock time. For the CCG-based quadratization algorithm, all wall-clock times include the runtime of the CCG construction. For each monomial coefficient a, the CCG-based quadratization algorithm for all experiments uses $J = L + 1$ and $L = a + 1$. The exact values of these parameters do not matter insofar as the condition $J \geq L > a > 0$ holds.

For polynomial expansion and the CCG-based quadratization algorithm, we first compare the numbers of auxiliary variables, the numbers of terms in the quadratizations, the runtime of the quadratization algorithms, and the runtime of the QPBO solver for minimization of the quadratizations. The runtime of the quadratization algorithm, referred to as its *quadratization time*, also includes the time for combining like terms. While depending mostly on the number of auxiliary variables, the quadratization time also depends on the number of like terms combined in the quadratization algorithm and the number of terms in the quadratization. Table 2 reports these comparisons for PBFs with all positive monomials. In each instance, the number of monomials is maximized, with integer coefficients chosen randomly from the interval $[1, 300]$. Table 2 shows the average and standard deviation for the results of the worst-case scenario over 10 instances where $n = d$ and the number of monomials is $m = 2^d$. The CCG-based quadratization algorithm significantly outperforms polynomial expansion in all four metrics as d increases.

Table 4: Similar to Table 2, except that all reported numbers are averaged over 100 instances, $n = 15$, $m = 500$, and the monomial coefficients are non-zero integers chosen randomly from $[-300, 300]$.

d		3	4	5	6	7	8	9	10	11
original number of terms		500	500	500	500	500	500	500	500	500
number of auxiliary variables	Poly	**394.05** ±0.2179	**405.71** ±2.7	**491.56** ±9.23	**538.27** ±10.43	**614.47** ±17.73	**666.57** ±19.45	**733.76** ±22.15	**789.70** ±23.12	**859.77** ±26.37
	CCG	584.11 ±9.91	720.21 ±20.62	905.18 ±29.41	1088.23 ±43.52	1271.9 ±55.54	1438.61 ±61.74	1591.95 ±63.92	1742.11 ±70.80	1873.79 ±75.93
number of terms	Poly	**1637.18** ±0.887	**1896.20** ±15.02	**2596.18** ±61.81	**3121.74** ±75.32	**3954.80** ±154.1	**4662.54** ±181.1	**5607.75** ±214.2	**6498.62** ±249.0	**7630.98** ±314.8
	CCG	2029.31 ±19.92	2642.21 ±58.01	3473.31 ±98.33	4374.41 ±155.8	5346.31 ±220.1	6331.16 ±262.8	7343.51 ±275.9	8397.41 ±323.4	9430.44 ±374.6
quadratization time (s)[3]	Poly	0.1147 ±0.0096	0.1783 ±0.0122	0.3170 ±0.0261	0.5284 ±0.0624	0.8172 ±0.0928	1.1943 ±0.1459	1.6922 ±0.1574	2.3604 ±0.2247	3.2838 ±0.3174
	CCG	0.1126 ±0.0066	0.1730 ±0.0121	0.2754 ±0.0211	0.4289 ±0.0428	0.5937 ±0.0534	0.8082 ±0.0741	1.0397 ±0.0799	1.3350 ±0.1003	1.6727 ±0.1312
QPBO (s)[4]	Poly	0.0013 ±0.0003	0.0019 ±0.0002	0.0032 ±0.0004	0.0048 ±0.0006	0.0076 ±0.0010	0.0105 ±0.0018	0.0148 ±0.0032	0.0205 ±0.0045	0.0291 ±0.0063
	CCG	0.0011 ±0.0001	0.0015 ±0.0002	0.0020 ±0.0001	0.0027 ±0.0002	0.0035 ±0.0003	0.0043 ±0.0003	0.0052 ±0.0007	0.0062 ±0.0006	0.0071 ±0.0006

Table 3 reports a comparison similar to Table 2 for the "average" cases, i.e., in each instance, $n = d$ and the number of monomials is 2^d but the coefficient of each monomial is a non-zero integer chosen randomly from the interval $[-300, 300]$. Here, too, the CCG-based quadratization algorithm significantly outperforms polynomial expansion in all four metrics as d increases.

Table 4 reports a comparison similar to Table 2 for the case where $n = 15 > d$. Here, the degree i of each monomial is randomly chosen from $\{0, \ldots, d\}$. Then, i unique variables are chosen randomly from $\{1, \ldots, n\}$ to construct this monomial along with a non-zero integer coefficient for it chosen randomly from the interval $[-300, 300]$. $m = 500$ such monomials are generated and we report averages over 100 instances. Table 4 shows that the CCG-based quadratization algorithm continues to outperform polynomial expansion in the quadratization time and the runtime of the QPBO solver as d increases, although it uses more auxiliary variables and results in a quadratization with more terms. The reason is that the CCG-based quadratization algorithm derives its advantage from the recursive combinations of monomials, and the probability that these combinations take place decreases as the gap between $\sum_{i=0}^{d} \binom{n}{i}$, the maximum number of terms a degree-d PBF can have, and m, the actual number, increases. (As this gap increases, it is more difficult to encounter monomials that are of the same variables during each reduction process in the CCG-based quadratization algorithm). Nonetheless, the CCG-based quadratization algorithm is more efficient than polynomial expansion in its quadratization time since polynomial expansion not only generates more quadratic like terms for each monomial but also considers each monomial individually and altogether accumulates many quadratic like terms to be added up.

We finally investigate the role of m in comparison to the worst-case number of monomials. Table 5 reports a comparison similar to Table 2 for the case of vary-

Table 5: Similar to Table 2, except that $n = d = 12$, m varies in density $100m/2^d$, and the monomial coefficients are non-zero integers drawn randomly from $[-300, 300]$.

d		12	12	12	12	12
original number of terms (density)		10%	20%	80%	90%	100%
number of auxiliary variables	Poly	**671** ±20.1	**1295** ±23.8	**5287** ±35.6	6062 ±40.3	6641 ±43.3
	CCG	1219 ±28.3	2031 ±35.6	5612 ±49.8	**6005** ±40.3	**6510** ±46.2
number of terms	Poly	**5645** ±20.9	10769 ±30.0	39996 ±320.9	45019 ±380.1	49812 ±410.3
	CCG	6025 ±24.1	**10142** ±28.6	**28733** ±95.5	**31264** ±102.2	**33519** ±106.8
quadratization time (s)[3]	Poly	1.74 ±0.03	6.31 ±0.05	108.07 ±10.1	136.05 ±12.2	186.50 ±20.16
	CCG	0.68 ±0.06	1.92 ±0.05	16.80 ±2.80	20.51 ±3.23	24.24 ±5.75
QPBO (s)[4]	Poly	0.0157 ±0.017	0.0825 ±0.018	1.1389 ±0.210	1.50 ±0.232	1.66 ±0.243
	CCG	0.0029 ±0.0005	0.0068 ±0.0015	0.0397 ±0.0079	0.042 ±0.0079	0.054 ±0.0082

ing density, i.e., $100m/2^d$ in percentage. We set $n = d = 12$ and report averages over 10 instances. We observe that the advantages of the CCG-based quadratization algorithm become more pronounced as the density increases. While the CCG-based quadratization algorithm is advantageous in quadratization time and runtime of the QPBO solver for all densities, it becomes more useful in the number of auxiliary variables and the number of terms in the quadratizations as the density increases.

4.3 Case Study: The Uncapacitated Facility Location Problem

We consider a real-world problem called the *uncapacitated facility location problem* (UFLP), also known as the *simple plant location problem*. The UFLP can also be used to model other real-world problems such as vehicle dispatching. This problem is NP-hard and can be reformulated as a nonlinear PBO [2, 5, 19, 20].

Formally, the UFLP is characterized by a set of locations $I = \{1, \ldots, M\}$ and a set of users $J = \{1, \ldots, N\}$. Let f_i be the fixed cost of opening and operating a facility at location $i \in I$. Each user $j \in J$ is required to be served by exactly one facility. An $M \times N$ matrix $C = [c_{ij}]$ specifies the transportation cost of delivering products from a facility at location i to user j. The goal is to open facilities at a subset $S \subseteq I$ of locations that minimizes the sum of fixed costs and transportation costs, i.e.,

$$\sum_{i \in S} f_i + \sum_{j \in J} \min_{i \in S} c_{ij} \,. \tag{23}$$

In [5, 19], the following method is used for reformulating the UFLP as a nonlinear PBO. For each column j in C, we assume a non-decreasing ordering of its elements as

$$c_{i_1^j} \leq c_{i_2^j} \leq \cdots \leq c_{i_M^j} \,. \tag{24}$$

We denote the difference between consecutive elements as

$$\Delta c_{0j} = c_{i_1^j}\,, \tag{25}$$

$$\Delta c_{lj} = c_{i_{l+1}^j} - c_{i_l^j}\,, \quad 1 \le l < M\,. \tag{26}$$

Let $z_i = \begin{cases} 0 & \text{if } i \in S \\ 1 & \text{otherwise} \end{cases}$ for each $i \in \{1, \ldots, M\}$ and $\mathbf{z} = (z_1, \ldots, z_M)$. For any valid solution S, we have $\mathbf{z} \neq (1, \ldots, 1)$, and therefore $\forall j \in J : \min_{i|z_i=0} c_{ij} = \Delta c_{0j} + \sum_{l=1}^{M-1} \Delta c_{lj} z_{i_1^j} \ldots z_{i_l^j}$. Therefore, according to Eq. (23), the transportation cost is $\sum_{j\in J} \min_{i\in S} c_{ij} = \sum_{j=1}^{N} \left\{ \Delta c_{0j} + \sum_{l=1}^{M-1} \Delta c_{lj} z_{i_1^j} \ldots z_{i_l^j} \right\}$ and the fixed cost is $\sum_{i\in S} f_i = \sum_{i=1}^{M} f_i(1 - z_i)$. Hence, the total cost is

$$\sum_{i=1}^{M} f_i(1 - z_i) + \sum_{j=1}^{N} \left\{ \Delta c_{0j} + \sum_{l=1}^{M-1} \Delta c_{lj} z_{i_1^j} \ldots z_{i_l^j} \right\}. \tag{27}$$

The UFLP is equivalent to computing

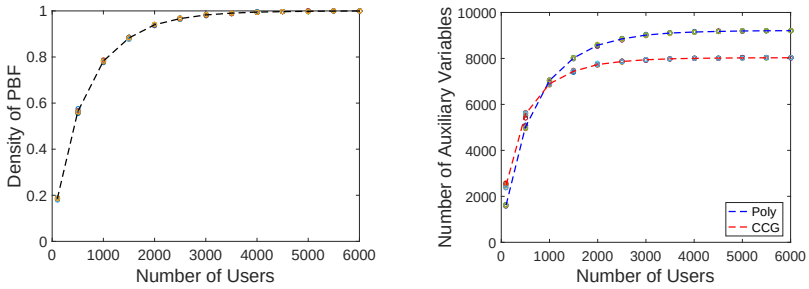
$$\begin{aligned} &\arg\min_{\mathbf{z}} \left\{ \sum_{i=1}^{M} f_i(1 - z_i) + \sum_{j=1}^{N} \left\{ \Delta c_{0j} + \sum_{l=1}^{M-1} \Delta c_{lj} z_{i_1^j} \ldots z_{i_l^j} \right\} \right\} \\ &\text{subject to: } \mathbf{z} \neq (1, \ldots, 1), \end{aligned}$$

which, in turn, is equivalent to computing

$$\arg\min_{\mathbf{z}} \left\{ \sum_{i=1}^{M} f_i(1 - z_i) + \sum_{j=1}^{N} \left\{ \Delta c_{0j} + \sum_{l=1}^{M-1} \Delta c_{lj} z_{i_1^j} \ldots z_{i_l^j} \right\} + \lambda \prod_{i=1}^{M} z_i \right\},$$

where $\lambda > \max_{i\in I} f_i$. All nonlinear terms have positive coefficients because of Eq. (24). The degree of the resulting PBF is determined by the number of facilities. While different columns of C potentially use different orderings, the same ordering can be applicable to different columns. The number of unlike terms in the PBF is determined by the number of different orderings, which is generally affected by the number of users.

We perform experiments to compare the performance (in terms of the number of auxiliary variables) of polynomial expansion and the CCG-based quadratization algorithm on PBFs resulting from UFLP instances. As reported in [21] and Tables 2 to 5, the number of auxiliary variables is the most important comparison parameter and is also indicative of the quadratization time and the runtime of the QPBO solver. We set the number of locations $M = 12$, which results in PBFs of degree 12. We vary the number of users N, which results in PBFs of varying densities. Except that the linear term coefficients are negative, this resembles the case of $d = 12$ in Table 2. For each instance, we randomly select the fixed costs of the facilities and the transportation costs from location i to user j. As the number of users increases, we plot the number of terms in the PBFs in Fig. 2a and the number of auxiliary variables for each quadratization algorithm in Fig. 2b.



(a) (b)

Fig. 2: UFLP experimental results. (a) shows $M = 12$ facility locations and N users. 10 instances are generated for each N. For each instance, the fixed cost for location i is an integer chosen randomly from $[1, 10000]$. The transportation cost from location i to user j is an integer chosen randomly from the interval $[1, 100]$. The density of the PBF is defined as the number of terms in the PBF divided by 2^M. (b) compares the number of auxiliary variables in the quadratizations of polynomial expansion and the CCG-based quadratization algorithm on all PBFs in (a). The y-axis indicates the number of auxiliary variables for each quadratization algorithm.

Figure 2a shows that, as more users are added, the density increases rapidly at first and quickly approaches 1. Figure 2b shows that polynomial expansion is preferable when the number of users is small but is quickly outperformed by the CCG-based quadratization algorithm as the number of users increases. This observation is consistent with the results in Table 5, which show that the CCG-based quadratization algorithm is more beneficial for higher densities. For the UFLP, the superior performance of the CCG-based quadratization algorithm with respect to the number of auxiliary variables is due to three possible reasons: (i) The degrees of the resulting PBFs are usually high, (ii) the coefficients of the nonlinear terms in these PBFs are all positive, and (iii) the PBFs become denser with a higher number of users.

5 Conclusion

We developed the CCG-based quadratization algorithm for the nonlinear PBO on general PBFs and compared it to state-of-the-art algorithms. We first proved the theoretical advantages of the CCG-based quadratization algorithm over other algorithms. We then experimentally verified these advantages. We observed that our CCG-based quadratization algorithm not only significantly outperforms other algorithms on medium-sized and large PBFs but is also preferable for smaller PBFs, to which asymptotic theoretical results are not directly applicable. We also showed that the CCG-based quadratization algorithm is applicable to real-world problems such as the UFLP, especially when the number of users to deliver products to is large.

References

1. Abío, I., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E., Mayer-Eichberger, V.: A new look at BDDs for pseudo-Boolean constraints. Journal of Artificial Intelligence Research **45**(1), 443–480 (2012)
2. AlBdaiwi, B.F., Goldengorin, B., Sierksma, G.: Equivalent instances of the simple plant location problem. Computers & Mathematics with Applications **57**(5), 812–820 (2009)
3. Anthony, M., Boros, E., Crama, Y., Gruber, A.: Quadratization of symmetric pseudo-Boolean functions. Discrete Applied Mathematics **203**, 1–12 (2016). https://doi.org/10.1016/j.dam.2016.01.001
4. Anthony, M., Boros, E., Crama, Y., Gruber, A.: Quadratic reformulations of nonlinear binary optimization problems. Mathematical Programming **162**(1-2), 115–144 (2017). https://doi.org/10.1007/s10107-016-1032-4
5. Beresnev, V.: On a problem of mathematical standardization theory. Upravliajemyje Sistemy **11**, 43–54 (in Russian) (1973)
6. Berthold, T., Heinz, S., Pfetsch, M.E.: Nonlinear pseudo-Boolean optimization: Relaxation or propagation? In: the International Conference on Theory and Applications of Satisfiability Testing. pp. 441–446 (2009). https://doi.org/10.1007/978-3-642-02777-2_40
7. Bockmayr, A.: Logic programming with pseudo-Boolean constraints. In: Constraint Logic Programming. pp. 327–350 (1993)
8. Bofill, M., Coll, J., Suy, J., Villaret, M.: Compact MDDs for pseudo-Boolean constraints with at-most-one relations in resource-constrained scheduling problems. In: the International Joint Conference on Artificial Intelligence. pp. 555–562 (2017). https://doi.org/10.24963/ijcai.2017/78
9. Boros, E., Crama, Y., Rodríguez-Heck, E.: Quadratizations of symmetric pseudo-Boolean functions: Sub-linear bounds on the number of auxiliary variables. In: the International Symposium on Artificial Intelligence and Mathematics (2018), http://isaim2018.cs.virginia.edu/papers/ISAIM2018_Boolean_Boros_etal.pdf
10. Boros, E., Gruber, A.: On quadratization of pseudo-Boolean functions. arXiv preprint arXiv:1404.6538 (2014)
11. Boros, E., Hammer, P.L., Minoux, M., Rader Jr, D.J.: Optimal cell flipping to minimize channel density in VLSI design and pseudo-Boolean optimization. Discrete Applied Mathematics **90**(1–3), 69–88 (1999)
12. Choi, V.: Minor-embedding in adiabatic quantum computation: I. the parameter setting problem. Quantum Information Processing **7**(5), 193–209 (2008). https://doi.org/10.1007/s11128-008-0082-9
13. Eén, N., Sörensson, N.: Translating pseudo-Boolean constraints into SAT. Journal on Satisfiability, Boolean Modeling and Computation **2**, 1–26 (2006)
14. Fioretto, F., Xu, H., Koenig, S., Kumar, T.K.S.: Solving multiagent constraint optimization problems on the constraint composite graph. In: the International Conference on Principles and Practice of Multi-Agent Systems. pp. 106–122 (2018). https://doi.org/10.1007/978-3-030-03098-8_7
15. Freeman, R.J., Gogerty, D.C., Graves, G.W., Brooks, R.B.: A mathematical model of supply support for space operations. Operations Research **14**(1), 1–15 (1966)
16. Glover, F., Woolsey, E.: Converting the 0-1 polynomial programming problem to a 0-1 linear program. Operations Research **22**(1), 180–182 (1974)
17. Gruber, A.G.: Algorithmic and complexity results for Boolean and pseudo-Boolean functions. Ph.D. thesis, Rutgers University-Graduate School-New Brunswick (2015)

18. Hammer, P.L., Rudeanu, S.: Boolean methods in operations research and related areas, vol. 7. Springer Science & Business Media (2012)
19. Hammer, P.: Plant location — a pseudo-Boolean approach. Israel Journal of Technology **6**, 330–332 (1968)
20. Hansen, P., Kochetov, Y., Mladenovi, N.: Lower bounds for the uncapacitated facility location problem with user preferences. Groupe d'études et de recherche en analyse des décisions, HEC Montréal (2004)
21. Ishikawa, H.: Transformation of general binary MRF minimization to the first-order case. IEEE Transactions on Pattern Analysis and Machine Intelligence **33**(6), 1234–1249 (2011). https://doi.org/10.1109/TPAMI.2010.91
22. Johnson, M.W., Amin, M.H.S., Gildert, S., Lanting, T., Hamze, F., Dickson, N., Harris, R., Berkley, A.J., Johansson, J., Bunyk, P., Chapple, E.M., Enderud, C., Hilton, J.P., Karimi, K., Ladizinsky, E., Ladizinsky, N., Oh, T., Perminov, I., Rich, C., Thom, M.C., Tolkacheva, E., Truncik, C.J.S., Uchaikin, S., Wang, J., Wilson, B., Rose, G.: Quantum annealing with manufactured spins. Nature **473**, 194–198 (2011). https://doi.org/10.1038/nature10012
23. Joshi, S., Martins, R., Manquinho, V.: Generalized totalizer encoding for pseudo-Boolean constraints. In: the International Conference on Principles and Practice of Constraint Programming. pp. 200–209 (2015). https://doi.org/10.1007/978-3-319-23219-5_15
24. Kahl, F., Strandmark, P.: Generalized roof duality for pseudo-Boolean optimization. In: the International Conference on Computer Vision. pp. 255–262 (2011). https://doi.org/10.1109/ICCV.2011.6126250
25. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
26. Kolmogorov, V., Rother, C.: Minimizing nonsubmodular functions with graph cuts – a review. IEEE Transactions on Pattern Analysis and Machine Intelligence **29**(7), 1274–1279 (2007). https://doi.org/10.1109/TPAMI.2007.1031
27. Kumar, T.K.S.: Incremental computation of resource-envelopes in producer-consumer models. In: the International Conference on Principles and Practice of Constraint Programming. pp. 664–678 (2003). https://doi.org/10.1007/978-3-540-45193-8_45
28. Kumar, T.K.S.: A framework for hybrid tractability results in Boolean weighted constraint satisfaction problems. In: the International Conference on Principles and Practice of Constraint Programming. pp. 282–297 (2008). https://doi.org/10.1007/978-3-540-85958-1_19
29. Kumar, T.K.S.: Lifting techniques for weighted constraint satisfaction problems. In: International Symposium on Artificial Intelligence and Mathematics (2008), http://isaim2008.unl.edu/PAPERS/TechnicalProgram/ISAIM2008_0004_d1de5114b3cb94de7e670ab2905c3b3d.pdf
30. Kumar, T.K.S.: Kernelization, generation of bounds, and the scope of incremental computation for weighted constraint satisfaction problems. In: International Symposium on Artificial Intelligence and Mathematics (2016)
31. Manquinho, V., Marques-Silva, J., Planes, J.: Algorithms for weighted Boolean optimization. In: the International Conference on Theory and Applications of Satisfiability Testing. pp. 495–508 (2009). https://doi.org/10.1007/978-3-642-02777-2_45
32. Manquinho, V., Roussel, O.: Pseudo-Boolean competition. For 2016, see http://www.cril.univ-artois.fr/PB16 (2016)
33. Manquinho, V.M., Marques-Silva, J.: On using cutting planes in pseudo-Boolean optimization. Journal on Satisfiability, Boolean Modeling and Computation **2**, 209–219 (2006)

34. Osokin, A.: Matlab wrapper to the QPBO algorithm by V. Kolmogorov. https://github.com/aosokin/qpboMex (2014)
35. Philipp, T., Steinke, P.: PBLib – a library for encoding pseudo-Boolean constraints into CNF. In: the International Conference on Theory and Applications of Satisfiability Testing. pp. 9–16 (2015). https://doi.org/10.1007/978-3-319-24318-4_2
36. Rendl, F., Rinaldi, G., Wiegele, A.: Solving max-cut to optimality by intersecting semidefinite and polyhedral relaxations. Mathematical Programming **121**(2), 307 (2010)
37. Rhys, J.: A selection problem of shared fixed costs and network flows. Management Science **17**(3), 200–207 (1970)
38. Wegener, I., Witt, C.: On the analysis of a simple evolutionary algorithm on quadratic pseudo-Boolean functions. Journal of Discrete Algorithms **3**(1), 61–78 (2005)
39. Xu, H., Koenig, S., Kumar, T.K.S.: A constraint composite graph-based ILP encoding of the Boolean weighted CSP. In: the International Conference on Principles and Practice of Constraint Programming. pp. 630–638 (2017). https://doi.org/10.1007/978-3-319-66158-2_40
40. Xu, H., Kumar, T.K.S., Koenig, S.: The Nemhauser-Trotter reduction and lifted message passing for the weighted CSP. In: the International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming. pp. 387–402 (2017). https://doi.org/10.1007/978-3-319-59776-8_31