J Math Model Algor (2014) 13:143–167
DOI 10.1007/s10852-013-9225-9

Encoding Binary Arithmetic Operations in Integer Programming Formulations

Raul Conejeros · Vassilios S. Vassiliadis · Thomas A. Pogiatzis

Received: 4 August 2012 / Accepted: 1 April 2013 / Published online: 3 May 2013

Abstract This paper presents the encoding of binary arithmetic operations within Integer Programming (IP) formulations, specifically the encoding of binary multiplication and addition/subtraction. This allows the direct manipulation of integer quantities represented as binary strings of arbitrary size. Many articles published in the past within the Chemical Engineering community have used this representation of integer quantities within Mixed-Integer formulations for Process Optimization and Design. Applications such as these can benefit from the formulations derived in this work. As a demonstrative application we consider the simple number factorization problem, according to which given an odd number C factors A and B are to be found such that C equals their product. If any such factors are found the number is factorable, else it is proven to be prime. An IP formulation is derived involving upper and lower bounding logical constraints to encode for the value of the binary string digits. The formulation involves $\mathcal{O}(\log C)$ binary variables, $\mathcal{O}((\log C)^2)$ continuous variables, and $\mathcal{O}((\log C)^2)$ constraints to describe the problem. Computational results demonstrate the validity of this approach, highlighting also the fact that such formulations are not very tight thus resulting in large numbers of iterations of the Branch and Bound algorithm used. It is also observed that the formulations become

R. Conejeros
Escuela de Ingeniería Bioquímica, Pontificia Universidad Católica de Valparaíso,
Av. Brasil 2147, Valparaíso, Chile
e-mail: rconejer@ucv.cl

V. S. Vassiliadis (✉) · T. A. Pogiatzis
Department of Chemical Engineering and Biotechnology,
University of Cambridge, Pembroke Street, Cambridge CB2 3RA, UK
e-mail: vsv20@cam.ac.uk

T. A. Pogiatzis
e-mail: tp309@cam.ac.uk

significantly tighter if logical upper bounding constraints forcing continuous variables involved to be zero are included.

Keywords Binary strings · Binary arithmetic · Number factorization · Integer programming

1 Introduction

Integer Programming (IP) has many applications in various disciplines. In the particular case of Chemical Engineering, IP has enjoyed significant growth over the last three decades, with contributions both in terms of applications and formulations, as well as novel algorithms. A textbook covering the extent of applications and algorithms in Chemical Engineering is [8], and [9] gives a review of Mixed-Integer techniques. Applications of IP range from the scheduling of batch plants [11, 14], to supply chain design [18], and to the design of chemical processes [10].

Of interest to the theme of our work in this paper is IP in which the quantities involved are integer numbers and not exclusively binary ones (0–1 programming). A general classification of domains where such a requirement is found is

1. Production Planning [16], where resources may be constrained to be integer (time, labour force size, quantities of materials, capacities etc.).
2. Vehicle Routing Problem (VRP) and Capacitated VRP (CVRP), where the route selection is optimized along with the load of each vehicle [12, 17]. Routing is carried out via the use of binary variables, while the requirement for integrality of the quantities transported necessitates the use of integer variables
3. Network Design, optimizing both the topology of a network and the associated line capacities [3]. Binary variables capture the design of the topology, while integer variables are needed for integral capacities.

Logic constraints within IP are formulated frequently via the use of binary variables, but also it is possible to have conditions depending on the value of general integer variables. The representation of integer variables within IP formulations is straightforward, if desired, through their expansion as binary strings with the digits represented by binary variables [1, 15].

In this paper, we demonstrate how to encode direct arithmetic operations on integer variables via their binary expansion within the IP formulations involved. As an application we consider the problem where a number is to be proven factorable or prime. The aim is to demonstrate the capability of logic programming to extend to purely abstract mathematical problems. For the particular application chosen, extensive bibliographical searches revealed no relevant publications. This was the case for both binary arithmetic encoding and for the problem of number factorization via IP.

This paper is organized as follows: Section 2 develops the main theory for proving that an integer be factorable, focusing on the integer representation of the variables involved, Section 3 discusses in detail the development of the mathematical programming formulation for the problem; the optimization problem and computational results are presented in Section 4, and finally conclusions are presented in Section 5.

2 Number Factorability Testing Formulation

The factorization of numbers and proof of primality are extensively developed topics in Number Theory [7, 13] with a relatively recent result proving that prime number factoring is in $\mathcal{P}$ [2], where $\mathcal{P}$ is the set of problems for which an answer is computable in polynomial time. For the purpose of demonstrating the encoding of binary arithmetic in IP formulations, we consider the simplest method possible, according to which given an odd number C it may be tested for factorability by finding factors A and B such that:

$$A \cdot B = C \tag{1}$$

The first number, A, may be used sequentially to divide C and test whether the result is an integer number. If no integer A divides exactly C then the given number is prime. In searching for a value of A sequentially it is only necessary to reach the square root of C skipping over the even numbers. Thus for A we need to scan the following values:

$$3, 5, 7, \ldots, \left\lfloor \sqrt{C} \right\rfloor \tag{2}$$

With the above and counting the divisions as the operations of this algorithm, only half of the numbers up to the square root of the number tested need to be scanned at most, so we find that the algorithm's complexity is $\mathcal{O}(\frac{1}{2}\sqrt{C})$. We explore this approach through IP as it is used to demonstrate the direct encoding of binary arithmetic into IP formulations.

Numbers A, B and C are represented in binary basis by [1, 15]:

$$A = \sum_{i=0}^{NA} x_i \cdot 2^i \tag{3a}$$

$$B = \sum_{j=0}^{NB} y_j \cdot 2^j \tag{3b}$$

$$C = \sum_{k=0}^{NC} t_k \cdot 2^k \tag{3c}$$

Variable vectors $x \in \{0, 1\}^{(NA+1)}$ and $y \in \{0, 1\}^{(NB+1)}$ are to be determined such that the condition in Eq. 1 is satisfied. The vector $t \in \{0, 1\}^{(NC+1)}$ is a set of constants which represent number C in binary format.

The size of the binary strings for numbers A and C, given by parameters NDA and NDC, are such that:

$$NDA = NA + 1 = \left\lfloor \log_2 \left\lfloor \sqrt{C} \right\rfloor \right\rfloor + 1 \tag{4a}$$

$$NDC = NC + 1 = \lfloor \log_2 C \rfloor + 1 \tag{4b}$$

The size of number B may be considered to be the same as that of number C, with the saving that both A and B must be greater or equal to 3, if C is factorable. This

yields that $A \geq 3$, hence from Eq. 1 we get that $C/B \geq 3$ and $B \leq C/3$. This yields that the number of digits NDB for number B is given by:

$$\begin{aligned} NDB = NB + 1 &= \left\lfloor \log_2 \left(\frac{C}{3} \right) \right\rfloor + 1 \\ &= \lfloor \log_2 C \rfloor - \lfloor \log_2 3 \rfloor + 1 \\ &= \lfloor \log_2 C \rfloor \\ &= NDC - 1 \\ &= NC \end{aligned} \tag{4c}$$

As described in the Eqs. 4a–4c, NA, NB, and NC are one bit shorter than the minimum binary string length required to represent the associated numbers A, B, and C. This is so because the counting in Eqs. 3a–3c starts from index value 0 (the 0-th power of 2).

One might think that there are further savings to be made since if A has the range $[1, \lfloor \sqrt{C} \rfloor]$, thus B need only be searched for in the range $[\lceil \sqrt{C} \rceil, C]$. The range for B thus appears shorter and hence fewer digits may be required. Number B would have to be represented by:

$$B = \left\lceil \sqrt{C} \right\rceil + \sum_{j=0}^{NB} y_j \cdot 2^j \tag{5}$$

First, when the product with number A is formed this would complicate the resulting formulation and involve large integer numbers ($\lceil \sqrt{C} \rceil$ would appear in the constraints). Second, the gains are minimal as in this case we would need NB digits given by (dropping the rounding up/down operations):

$$\begin{aligned} NB &= \log_2 \left(C - \sqrt{C} \right) \\ &= \log_2 \left[\sqrt{C} \cdot \left(\sqrt{C} - 1 \right) \right] \\ &= \frac{1}{2} \log_2 C + \log \left(\sqrt{C} - 1 \right) \\ &\approx \log_2 C \\ &= NC \end{aligned} \tag{6}$$

It is noted that for big C there are thus no gains as even the restricted range of B would require the same number of digits to represent it.

The factorability test in Eq. 1 with the definitions in Eqs. 3a–3c becomes:

$$\sum_{i=0}^{NA} \sum_{j=0}^{NB} x_i \cdot y_j \cdot 2^{(i+j)} = \sum_{k=0}^{NC} t_k \cdot 2^k \tag{7}$$

It is seen that the LHS of the equation involves bilinear terms $x_i \cdot y_j$ which are replaced by the continuous variables z_{ij} and a set of constraints declared in Section 3.

The representation of a number C in binary format in terms of its bits t_k, $k = 0, 1, \ldots, NC$ is computed by Algorithm 1.

Algorithm 1 Number representation in binary format

$NDC \leftarrow \lfloor \log_2 C \rfloor + 1$
$temp \leftarrow C$
for $k = NDC \rightarrow 1$ **do**
  $t_{k-1} \leftarrow temp \mod 2$
  $temp \leftarrow \lfloor temp/2 \rfloor$
end for

```
      1111
       101 x
---------
      1111 (level 0 sum)
     0000  +
---------
     01111 (level 1 sum)
    1111  +
---------
   1001011 (level 2 sum)
```

Fig. 1 Example of binary multiplication

These terms are needed once we have defined how binary multiplication is to be encoded using logical constraints. The constraint in Eq. 7 is not used directly in the formulation that follows.

With regards to binary arithmetic [6] we focus first on representing binary multiplication as a stagewise addition of the multiplicand an equal number of times to the number of digits of the multiplier, storing the intermediate addition results explicitly. As an example, consider multiplication of $B = 1111$ (= binary 15) with $A = 101$ (= binary 5). The multiplication is shown in levels in the example in Fig. 1, equal to the number $NA = 3$ of the multiplier (using as multiplier always the smaller number), while the number of digits required being determined by number $NB = 4$.

It is seen that at each level of the multiplication process we shift by one position from the previous level sum. The resulting sum is at most equal to the number of bits of number B plus 1 digit.

3 Mathematical Programming Formulation: Variables and Constraints

In this section, the binary multiplication algorithm is encoded using binary variables and appropriate constraints.

3.1 Problem Variables

The bilinear products $z_{ij} = x_i \cdot y_j$ are required, and these are defined in Eqs. 8a–8d below, based on the fact that variables x and y are binary.

$$0 \le z_{ij} \le 1 \tag{8a}$$

$$z_{ij} \le x_i \tag{8b}$$

$$z_{ij} \le y_j \tag{8c}$$

$$x_i + y_j - 1 \le z_{ij} \tag{8d}$$
$$i = 0, 1, \ldots NA; \quad j = 0, 1, \ldots, NB$$

There are $(NA + 1) \times (NB + 1)$, or approximately $\mathcal{O}(\frac{1}{2}(\log_2 C)^2)$ continuous variables z_{ij}. The above defines $3 \times (NA + 1) \times (NB + 1)$ constraints, excluding the simple bounds on the z_{ij} variables in Eq. 8a.

Variables need to be defined for the summations' digits of each level $i = 0, 1, \ldots, NA$:

$$0 \le s_{ij} \le 1; \quad i = 0, 1, \ldots, NA; \quad j = 0, 1, \ldots, (NB + 1) \tag{9}$$

There are $(NA + 1) \times (NB + 2)$ variables of this type, or approximately $\mathcal{O}(\frac{1}{2}(\log_2 C)^2)$.

Variables need to be defined also for the carryover digit of the summations of each level $i = 0, 1, \ldots, NA$:

$$0 \le c_{ik} \le 1; \quad i = 1, 2, \ldots, NA; \quad k = 0, 1, \ldots, NB \tag{10}$$

There are $NA \times (NB + 1)$ variables of this type, or approximately $\mathcal{O}(\frac{1}{2}(\log_2 C)^2)$.

We need further variables to enforce the constraint that the factors are such that $A \le B$ directly from the binary string representation (if desired to be enforced, see later discussion). What is implemented is the two's complement method of subtracting factor A from factor B. First the binary subtraction summation digits' variables are defined:

$$0 \le ss_j \le 1; \quad j = 0, 1, \ldots, NB \tag{11}$$

There are $(NB + 1)$ variables of this type, or approximately $\mathcal{O}(\log_2 C)$.

The carryover digits of this summation are defined via the following variables:

$$0 \le cc_j \le 1; \quad j = 0, 1, \ldots, NB \tag{12}$$

There are $(NB + 1)$ variables of this type, or approximately $\mathcal{O}(\log_2 C)$.

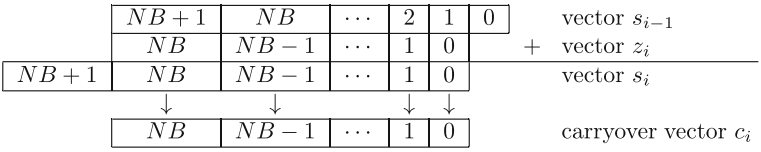

Fig. 2 Binary addition structure indices

3.2 Main Constraints' Definition

The construction of the constraints follows the addition structure within binary multiplication, and is shown in Fig. 2.

The constraints are as follows.

Level $i = 0$ summation digits fixing

$$s_{0,j} = z_{0,j}; \quad j = 0, 1, \ldots, NB \tag{13a}$$

$$s_{0,NB+1} = 0 \tag{13b}$$

The last constraint need not be imposed directly, instead the upper bound of the summation digit variable is set to 0.

Not counting the last constraint, there are $(NB + 1)$ constraints here.

Level $i = 1, 2, \ldots, NA$ position $k = 0$ of the added number

At this position there is no previous carryover as the current position is a shifted one. The digit $s_{i,0}$ is produced as a result of the summation of $s_{i-1,1} + z_{i,0}$ potentially producing a carryover digit $c_{i,0}$.

The condition relating the summation digit and the carryover digit at position $k = 0$ is as follows, noting that this is not strictly required in the formulation:

$$s_{i,0} + 2 \cdot c_{i,0} = s_{i-1,1} + z_{i,0}; \quad i = 1, 2, \ldots, NA \tag{14a}$$

If the variables involved in the above equation were all declared to be binary it would be sufficient on its own. For the general situation the following constraints need to be also added.

The carryover digit $c_{i,0}$ is equal to 1 only if both variables $s_{i-1,1}$ and $z_{i,0}$ are equal to 1.

$$c_{i,0} \geq 1 + s_{i-1,1} + z_{i,0} - 2; \quad i = 1, 2, \ldots, NA \tag{14b}$$

The cases $c_{i,0}$ equals 0 are to be satisfied via the objective function. Additional constraints that would enforce this directly, although not necessary under the current objective function in Eq. 23, are as follows: variables $c_{i,0}$ are 0 if both variables $s_{i-1,1}$ and $z_{i,0}$ are equal to 0, or one of them is equal to 1 and the other equal to 0.

$$c_{i,0} \leq 2 - (1 - s_{i-1,1}) - (1 - z_{i,0}) \tag{14c}$$

$$c_{i,0} \leq 2 - s_{i-1,1} - (1 - z_{i,0}) \tag{14d}$$

$$c_{i,0} \leq 2 - (1 - s_{i-1,1}) - z_{i,0} \tag{14e}$$
$$i = 1, 2, \ldots, NA$$

The summation digit $s_{i,0}$ is equal to 1 if either one of the variables $s_{i-1,i}$ and $z_{i,0}$ is equal to 1 and the other equal to 0.

$$s_{i,0} \geq 1 + s_{i-1,1} + (1 - z_{i,0}) - 2 \tag{14f}$$

$$s_{i,0} \geq 1 + (1 - s_{i-1,1}) + z_{i,0} - 2 \tag{14g}$$

$$i = 1, 2, \ldots NA$$

The cases $s_{i,0}$ equals 0 are to be satisfied via the objective function. Additional constraints that would enforce this directly, although not necessary under the current objective function in Eq. 23, are as follows: variables $s_{i,0}$ are 0 if both variables $s_{i-1,1}$ and $z_{i,0}$ are both equal to 1, or both of them are equal to 0.

$$s_{i,0} \leq 2 - s_{i-1,1} - z_{i,0} \tag{14h}$$

$$s_{i,0} \leq 2 - (1 - s_{i-1,1}) - (1 - z_{i,0}) \tag{14i}$$

$$i = 1, 2, \ldots NA$$

There are $9 \times NA$ constraints defined above, or approximately $\mathcal{O}(\frac{9}{2} \log_2 C)$.

Level $i = 1, 2, \ldots, NA$ **positions** $k = 1, 2, \ldots, NB$ **of the added number**

At this position, the digit $s_{i,k}$ is produced as a result of the summation of $s_{i-1,k+1} + z_{i,k} + c_{i,k-1}$ potentially producing a carryover digit $c_{i,k}$.

The condition relating the summation digit and the carryover digit at position $k = 1, 2, \ldots, NB$ is as follows, noting that this is not strictly required in the formulation:

$$s_{ik} + 2 \cdot c_{ik} = s_{i-1,k+1} + z_{ik} + c_{i,k-1} \tag{15a}$$

$$i = 1, 2, \ldots, NA; \quad k = 1, 2, \ldots, NB$$

There are $NA \times (NB - 1)$ constraints of this type, or approximately $\mathcal{O}(\frac{1}{2}(\log_2 C)^2)$.

If the variables involved in the above equation were all declared to be binary it would be sufficient on its own. For the general situation the following constraints need to be also added.

The carryover digit c_{ik} is equal to 1 only if the three variables $s_{i-1,i+k}$, z_{ik}, and $c_{i,k-1}$ involved are all three equal to 1, or any two of the three equal to 1 and the other equal to 0.

$$c_{ik} \geq 1 + s_{i-1,k+1} + z_{ik} + c_{i,k-1} - 3 \tag{15b}$$

$$c_{ik} \geq 1 + s_{i-1,k+1} + z_{ik} + (1 - c_{i,k-1}) - 3 \tag{15c}$$

$$c_{ik} \geq 1 + s_{i-1,k+1} + (1 - z_{ik}) + c_{i,k-1} - 3 \tag{15d}$$

$$c_{ik} \geq 1 + (1 - s_{i-1,k+1}) + z_{ik} + c_{i,k-1} - 3 \tag{15e}$$

$$i = 1, 2, \ldots, NA; \quad k = 1, 2, \ldots, NB$$

There are $4 \times NA \times (NB - 1)$ constraints of this type, or approximately $\mathcal{O}(2(\log_2 C)^2)$.

The cases where c_{ik} equals 0 are to be satisfied via the objective function. Additional constraints that would enforce this directly, although not necessary under the current objective function in Eq. 23, are as follows: variables c_{ik} are 0 if either all variables $s_{i-1,k+1}$, z_{ik} and $c_{i,k-1}$ are equal to 0, or if any one of them is equal to 1 and the other two equal to 0.

$$c_{ik} \leq 3 - (1 - s_{i-1,k+1}) - (1 - z_{ik}) - (1 - c_{i,k-1}) \tag{15f}$$

$$c_{ik} \leq 3 - s_{i-1,k+1} - (1 - z_{ik}) - (1 - c_{i,k-1}) \tag{15g}$$

$$c_{ik} \leq 3 - (1 - s_{i-1,k+1}) - z_{ik} - (1 - c_{i,k-1}) \tag{15h}$$

$$c_{ik} \leq 3 - (1 - s_{i-1,k+1}) - (1 - z_{ik}) - c_{i,k-1} \tag{15i}$$

$$i = 1, 2, \ldots, NA; \quad k = 1, 2, \ldots, NB$$

There are $4 \times NA \times (NB - 1)$ constraints of this type, or approximately, $\mathcal{O}(2(\log_2 C)^2)$.

The summation digit s_{ik} is equal to 1 only if the three variables $s_{i-1,k+1}$, z_{ik}, and $c_{i,k-1}$ are all three are equal to 1, or they involve exactly one of them equal to 1 and the other two equal to 0.

$$s_{ik} \geq 1 + s_{i-1,k+1} + z_{ik} + c_{i,k-1} - 3 \tag{15j}$$

$$s_{ik} \geq 1 + s_{i-1,k+1} + (1 - z_{ik}) + (1 - c_{i,k-1}) - 3 \tag{15k}$$

$$s_{ik} \geq 1 + (1 - s_{i-1,k+1}) + z_{ik} + (1 - c_{i,k-1}) - 3 \tag{15l}$$

$$s_{ik} \geq 1 + (1 - s_{i-1,k+1}) + (1 - z_{ik}) + c_{i,k-1} - 3 \tag{15m}$$

$$i = 1, 2, \ldots, NA; \quad k = 1, 2, \ldots, NB$$

There are $4 \times NA \times (NB - 1)$ constraints of this type, or approximately, $\mathcal{O}(2(\log_2 C)^2)$.

The cases s_{ik} equals 0 are to be satisfied via the objective function. Additional constraints that would enforce this directly, although not necessary under the current objective function in Eq. 23, are as follows: variables s_{ik} are 0 if either all variables $s_{i-1,k+1}$, z_{ik} and $c_{i,k-1}$ are equal to 0, or if any two of them are equal to 1 and the other is equal to 0.

$$s_{ik} \leq 3 - (1 - s_{i-1,k+1}) - (1 - z_{ik}) - (1 - c_{i,k-1}) \tag{15n}$$

$$s_{ik} \leq 3 - s_{i-1,k+1} - z_{ik} - (1 - c_{i,k-1}) \tag{15o}$$

$$s_{ik} \leq 3 - s_{i-1,k+1} - (1 - z_{ik}) - c_{i,k-1} \tag{15p}$$

$$s_{ik} \leq 3 - (1 - s_{i-1,k+1}) - z_{ik} - c_{i,k-1} \tag{15q}$$

$$i = 1, 2, \ldots, NA; \quad k = 1, 2, \ldots, NB$$

There are $4 \times NA \times (NB - 1)$ constraints of this type, or approximately $\mathcal{O}(2(\log_2 C)^2)$.

Level $i = 1, 2, \ldots, NA$ position $k = NB + 1$ of the added number

At this position, the digit $s_{i,NB+1}$ is produced. This is simply the carryover from the previous position summation, at $k = NB$, as there are no digits above it.

$$s_{i,NB+1} = c_{i,NB} \tag{16}$$

$$i = 1, 2, \ldots, NA$$

There are NA constraints of this type, or approximately $\mathcal{O}(\frac{1}{2} \log_2 C)$.

Matching of the input number C binary string

The input string is matched by enforcing it at the final summation level, as described by the following constraints.

$$s_{i,0} = t_i \tag{17a}$$

$$i = 0, 1, 2, \ldots, NA$$

$$s_{NA,k} = t_{NA+k} \tag{17b}$$

$$k = 1, 2, \ldots, (NC - NA)$$

$$s_{NA,k} = 0 \tag{17c}$$

$$k = (NC - NA + 1), (NC - NA + 2), \ldots, (NB + 1)$$

Further logical constraints

Additional constraints demanding that factors A and B are both greater than 3 (they are both odd by C being odd) are the following:

$$\sum_{i=0}^{NA} x_i \geq 2 \tag{18a}$$

$$\sum_{j=0}^{NB} y_j \geq 2 \tag{18b}$$

3.3 Tightening Constraints Ensuring $A \leq B$

To ensure that the factors A and B encoded in the binary vectors x and y, respectively, are such that $A \leq B$, we implement the binary number subtraction of $B - A$ using the two's complement approach [6]. According to this, number A, which has fewer digits than B, is padded with zeros for higher power digits, and then its digits are complemented (flipped from 1 to 0 and vice versa, using subtraction from 1: $x'_i = 1 - x_i$) and the result is added to the binary representation of B, along with the addition of 1 to the lowest digit location.

Position $k = 0$

At this position there is no previous carryover. The digit ss_0 is produced as a result of the summation of $y_0 + (1 - x_0) + 1$, potentially producing a carryover digit cc_0.

The condition relating the summation digit and the carryover digit at position $k = 0$ is as follows, noting that this is not strictly required in the formulation:

$$ss_0 + 2 \cdot cc_0 = y_0 + (1 - x_0) + 1 \tag{19a}$$

If the variables involved in the above equation were all declared to be binary it would be sufficient on its own. For the general situation the following constraints need to be also added.

The carryover digit cc_0 is equal to 1 only if both y_0 and $(1 - x_0)$ are equal to 1, or when only one of these terms is 1 and the other is 0.

$$cc_0 \geq 1 + y_0 + (1 - x_0) - 2 \tag{19b}$$

$$cc_0 \geq 1 + y_0 + x_0 - 2 \tag{19c}$$

$$cc_0 \geq 1 + (1 - y_0) + (1 - x_0) - 2 \tag{19d}$$

The cases cc_0 equals 0 are to be satisfied via the objective function. Additional constraints that would enforce this directly, although not necessary under the current objective function in Eq. 23, are as follows: variable cc_0 is 0 if both y_0 and $(1 - x_0)$ are equal to 0.

$$cc_0 \leq 2 - (1 - y_0) - x_0 \tag{19e}$$

The summation digit ss_0 is equal to 1 if either y_0 and $(1 - x_0)$ are both equal to 1, or both terms are equal to 0.

$$ss_0 \geq 1 + y_0 + (1 - x_0) - 2 \tag{19f}$$

$$ss_0 \geq 1 + (1 - y_0) + x_0 - 2 \tag{19g}$$

The cases ss_0 equals 0 are to be satisfied via the objective function. Additional constraints that would enforce this directly, although not necessary under the current objective function in Eq. 23, are as follows: variable ss_0 is 0 if either one of y_0 and $(1 - x_0)$ is equal to 1 and the other is equal to 0.

$$ss_0 \leq 2 - y_0 - x_0 \tag{19h}$$

$$ss_0 \leq 2 - (1 - y_0) - (1 - x_0) \tag{19i}$$

There are 9 constraints defined above.

Position $k = 1, 2, \ldots, NA$

The digit ss_k is produced as a result of the summation of $y_k + (1 - x_0) + cc_{k-1}$, potentially producing a carryover digit cc_k.

The condition relating the summation digit and the carryover digit at position $k = 1, 2, \ldots, NA$ is as follows, noting that this is not strictly required in the formulation:

$$ss_k + 2 \cdot cc_k = y_k + (1 - x_k) + cc_{k-1} \tag{20a}$$

$$k = 1, 2, \ldots, NA$$

If the variables involved in the above equation were all declared to be binary it would be sufficient on its own. For the general situation the following constraints need to be also added.

The carryover digit cc_k is equal to 1 only if all of y_k, $(1 - x_k)$ and cc_{k-1} are equal to 1, or when only two of these terms are 1 and the other is 0.

$$cc_k \geq 1 + y_k + (1 - x_k) + cc_{k-1} - 3 \tag{20b}$$

$$cc_k \geq 1 + y_k + (1 - x_k) + (1 - cc_{k-1}) - 3 \tag{20c}$$

$$cc_k \geq 1 + y_k + x_k + cc_{k-1} - 3 \tag{20d}$$

$$cc_k \geq 1 + (1 - y_k) + (1 - x_k) + cc_{k-1} - 3 \tag{20e}$$

$$k = 1, 2, \ldots, NA$$

The cases cc_k equals 0 are to be satisfied via the objective function. Additional constraints that would enforce this directly, although not necessary under the current

objective function in Eq. 23, are as follows: variable cc_k is 0 if all of y_k, $(1 - x_k)$ and cc_{k-1} are equal to 0, or any two are equal to 0 and the other one is equal to 1.

$$cc_k \leq 3 - (1 - y_k) - x_k - (1 - cc_{k-1}) \tag{20f}$$

$$cc_k \leq 3 - (1 - y_k) - x_k - cc_{k-1} \tag{20g}$$

$$cc_k \leq 3 - (1 - y_k) - (1 - x_k) - (1 - cc_{k-1}) \tag{20h}$$

$$cc_k \leq 3 - y_k - x_k - (1 - cc_{k-1}) \tag{20i}$$

$$k = 1, 2, \ldots, NA$$

The summation digit ss_k is equal to 1 if all of y_k, $(1 - x_k)$ and cc_{k-1} are equal to 1, or if only one of them is equal to 1 and the other two terms are equal to 0.

$$ss_k \geq 1 + y_k + (1 - x_k) + cc_{k-1} - 3 \tag{20j}$$

$$ss_k \geq 1 + y_k + x_k + (1 - cc_{k-1}) - 3 \tag{20k}$$

$$ss_k \geq 1 + (1 - y_k) + (1 - x_k) + (1 - cc_{k-1}) - 3 \tag{20l}$$

$$ss_k \geq 1 + (1 - y_k) + x_k + cc_{k-1} - 3 \tag{20m}$$

$$k = 1, 2, \ldots, NA$$

The cases ss_k equals 0 are to be satisfied via the objective function. Additional constraints that would enforce this directly, although not necessary under the current objective function in Eq. 23, are as follows: variable ss_k is 0 if all of y_k, $(1 - x_k)$ and cc_{k-1} are equal to 0, or any two are equal to 1 and the other one is equal to 0.

$$ss_k \leq 3 - (1 - y_k) - x_k - (1 - cc_{k-1}) \tag{20n}$$

$$ss_k \leq 3 - y_k - (1 - x_k) - (1 - cc_{k-1}) \tag{20o}$$

$$ss_k \leq 3 - y_k - x_k - cc_{k-1} \tag{20p}$$

$$ss_k \leq 3 - (1 - y_k) - (1 - x_k) - cc_{k-1} \tag{20q}$$

There are $17 \times NA$ constraints defined above, or approximately $\mathcal{O}(\frac{17}{2} \log_2 C)$.

Position $k = (NA + 1), (NA + 2), \ldots, NB$

The digit ss_k is produced as a result of the summation of $y_k + cc_{k-1} + 1$ potentially producing a carryover digit cc_k.

The condition relating the summation digit and the carryover digit at position $k = (NA+1), (NA+2), \ldots, NB$ is as follows, noting that this is not strictly required in the formulation:

$$ss_k + 2 \cdot cc_k = y_k + cc_{k-1} + 1 \tag{21a}$$

$$k = (NA+1), (NA+2), \ldots, NB$$

If the variables involved in the above equation were all declared to be binary it would be sufficient on its own. For the general situation the following constraints need to be also added.

The carryover digit cc_k is equal to 1 only if both of y_k and cc_{k-1} are equal to 1, or when either one of them is equal to 1 and the other equal to 0.

$$cc_k \geq 1 + y_k + cc_{k-1} - 2 \tag{21b}$$

$$cc_k \geq 1 + y_k + (1 - cc_{k-1}) - 2 \tag{21c}$$

$$cc_k \geq 1 + (1 - y_k) + cc_{k-1} - 2 \tag{21d}$$

$$k = (NA+1), (NA+2), \ldots, NB$$

The cases cc_k equals 0 are to be satisfied via the objective function. Additional constraints that would enforce this directly, although not necessary under the current objective function in Eq. 23, are as follows: variable cc_k is 0 if both of y_k and cc_{k-1} are equal to 0.

$$cc_k \leq 2 - (1 - y_k) - (1 - cc_{k-1}) \tag{21e}$$

The summation digit ss_k is equal to 1 if both of y_k and cc_{k-1} are equal to 1, or if both are equal to 0.

$$ss_k \geq 1 + y_k + cc_{k-1} - 2 \tag{21f}$$

$$ss_k \geq 1 + (1 - y_k) + (1 - cc_{k-1}) - 2 \tag{21g}$$

$$k = (NA+1), (NA+2), \ldots, NB$$

The cases ss_k equals 0 are to be satisfied via the objective function. Additional constraints that would enforce this directly, although not necessary under the current objective function in Eq. 23, are as follows: variable ss_k is 0 if either one of y_k and cc_{k-1} is equal to 1, and the other one is equal to 0.

$$ss_k \leq 2 - y_k - (1 - cc_{k-1}) \tag{21h}$$

$$ss_k \leq 2 - (1 - y_k) - cc_{k-1} \tag{21i}$$

There are $9 \times (NB - NA - 1)$ constraints defined above, or approximately $\mathcal{O}(\frac{9}{2} \log_2 C)$.

Position "$k = NB + 1$"

The digit cc_{NB} would be the digit assigned to the $(NB + 1)^{th}$ final digit of the summation. Based on subtraction of binary numbers using the two's complement method, if this digit is equal to 1 we discard it indicating that the previous digits of the summation carry the difference of $B - A$ and that $B > A$. If this digit is 0, this indicates that $B < A$. To exclude such solutions from the optimization problem, we simply impose the following single constraint:

$$cc_{NB} = 1 \tag{22}$$

3.4 Objective Function

The objective function does not reflect any quantity derived from the underlying factorization problem: it is just a means to set the continuous variables s, c, ss and cc to zero, if they are left loose by the lower bounding constraints defined throughout the formulation.

$$\min_{x,y,s,c,ss,cc} \sum_{i=0}^{NA} \sum_{j=0}^{NB+1} s_{ij} + \sum_{i=1}^{NA} \sum_{k=0}^{NB} c_{ik} + \sum_{j=0}^{NB} (ss_j + cc_j) \tag{23}$$

If the variables involved are defined as binary also, then the objective serves no purpose other than complete the feasibility problem as an optimization problem.

4 Optimization Problem and Computational Results

The optimization problem involves the objective function in Eq. 23, subject to the constraints defined by equations in Sections 3.2 and 3.3.

Of these we may wish to consider including the upper bounding constraints for the continuous variables, as outlined in the various parts of the formulation, and to include also the constraints that ensure $A \leq B$ if so desired.

Running this problem with a factorable number C produces feasible solutions with variables s, c, ss and cc attaining integer values (either 0 or 1 only). The run must be terminated at the first feasible all-integer solution, which proves the number to be factorable. Running with a prime number C should produce an infeasible solution at the point of exhaustion of the Branch and Bound (BB) tree. This would prove the number to be prime.

Here we present results for four types of formulation of the factorization problem. The formulations follow.

4.1 Model I

This model is comprised of the objective function in Eq. 23 and the constraints given by Eqs. 8a–8d, 9, 10, 13a, 13b, 14a, 14b, 14f, 14g, 15a–15e 15j–15m, 16, 17a–17c, 18a, 18b. Upper bounding constraints for the continuous variables and constraints

enforcing $A < B$ are not included. Thus, variables *ss* and *cc* do not occur in this model.

4.2 Model II

This model is comprised of all the constraints and objective of Model I, with additional upper bounding constraints for the continuous variables *s* and *c*. The additional constraints are given by Eqs. 14c–14e, 14h, 14i, 15f–15i, 15n–15q. Constraints enforcing $A < B$ are not included, thus variables *ss* and *cc* do not occur in this model.

4.3 Model III

This model is comprised of all the constraints and objective of Model I, with additional constraints enforcing $A < B$. The additional constraints are given by Eqs. 11, 12, 19a–19d, 19f, 19g, 20a–20e, 20j–20m, 21a–21d, 21f, 21g, 22. Variables *ss* and *cc* are required in this model.

4.4 Model IV

This model is comprised of all the constraints and objective of Model III, with the additional upper bounding constraints on the continuous variables *ss* and *cc*, enforcing $A < B$. The additional constraints are given by Eqs. 19e, 19h, 19i, 20f–20i, 20n–20q, 21e, 21h, 21i. Variables *ss* and *cc* are required in this model.

4.5 Computational Results and Discussion

Table 1 presents the various case studies considered along with a description of the size of the resulting optimization problems. Table 2 displays the resource usage for the solver CPLEX version 10 [5]. Finally, Table 3 reports the factors A and B for each number C considered in the case studies.

All case studies were run on an INTEL Core i5 2.4 GHz computer, using the Mixed-Integer Linear Programming solver CPLEX, interfaced with the GAMS modeling language [4]. All options were left at the default values, except biasing the search for the generation of integer solutions, and terminating the run after the first integer solution is discovered.

In general, provision of upper bounding logical constraints on the continuous variables is beneficial in terms of computational performance. Thus usually Model II performs better than Model I, and Model IV better than Model III.

In terms of enforcing the constraint that factors A and B are such that $A < B$, it only makes a difference in the results of Case 3. In fact for larger numbers this condition is satisfied by all formulations computationally, and moreover it does not contribute to make the search faster but has the opposite in making convergence much slower. Thus Model III is slower than Model I, and Model IV is slower than Model II.

The use of computational resources during the solution of the models with the BB algorithm of CPLEX are summarised in various figures, each analysing the performance of the 4 different models proposed in this work for each of the 12 integer numbers C tested for primality. Specifically, Fig. 3 shows the total number of nodes

Table 1 Case study descriptions

Case #	C	Prime	NA	NB	NC	Model											
						I			II			III			IV		
						B.V.	C.V.	Constr.	B.V.	C.V.	Constr.	B.V.	C.V.	Constr.	B.V.	C.V.	Constr.
1	35	N	2	4	5	8	57	149	8	57	223	8	67	186	8	67	285
2	37	Y	2	4	5	8	57	149	8	57	223	8	67	186	8	67	285
3	323	N	4	7	8	13	139	425	13	139	669	13	155	486	13	155	774
4	331	Y	4	7	8	13	139	425	13	139	669	13	155	486	13	155	774
5	5,825	N	6	11	12	19	279	923	19	279	1,481	19	303	1,014	19	303	1,638
6	5,821	Y	6	11	12	19	279	923	19	279	1,481	19	303	1,014	19	303	1,638
7	72,739	N	7	14	15	23	392	1,334	23	392	2,153	23	422	1,446	23	422	2,345
8	63,809	Y	7	14	15	23	392	1,334	23	392	2,153	23	422	1,446	23	422	2,345
9	837,537	N	9	18	19	29	610	2,144	29	610	3,485	29	648	2,286	29	648	3,729
10	835,847	Y	9	18	19	29	610	2,144	29	610	3,485	29	648	2,286	29	648	3,729
11	6,611,553	N	11	21	22	34	839	3,008	34	839	4,911	34	883	3,147	34	883	5,198
12	6,711,713	Y	11	21	22	34	839	3,008	34	839	4,911	34	883	3,174	34	883	5,198

Abbreviations are: "B.V." is binary variables, "C.V." is continuous variables, "Constr." is constraints

Table 2 Case study computational effort

Case #	Model											
	I			II			III			IV		
	CPU (s)	Nodes	Iter.	CPU (s)	Nodes	Iter.	CPU (s)	Nodes	Iter.	CPU (s)	Nodes	Iter.
1	<1	0	24	<1	0	15	<1	0	31	<1	0	18
2	<1	0	14	<1	0	10	<1	0	27	<1	0	14
3	<1	0	44	<1	0	48	<1	0	143	<1	0	132
4	<1	0	104	<1	0	86	<1	0	127	<1	0	158
5	<1	281	14,714	<1	16	919	<1	309	14,609	<1	22	1,422
6	1	400	17,589	<1	100	5,520	<1	200	10,062	<1	100	5,948
7	1	172	12,670	2	197	18,041	8	1,391	106,093	1	106	11,118
8	15	3,300	206,183	6	400	50,697	23	4,200	295,419	15	1,100	126,718
9	207	13,810	1,755,548	18	632	111,328	360	20,094	2,944,111	93	3,313	572,648
10	190	15,900	1,791,171	78	2,700	489,051	288	19,200	2,626,213	64	1,900	408,402
11	2,497	59,194	15,762,436	28	376	128,540	2,358	53,178	14,502,978	208	2,319	923,904
12	3,905	117,600	25,311,171	938	11,600	4,262,024	12,557	357,700	74,069,430	1,023	13,200	4,421,523

"Iter." stands for CPLEX iterations, x"Nodes" are the BB tree nodes examined

Table 3 Case study factors obtained

Case #	Model							
	I		II		III		IV	
	A	*B*	*A*	*B*	*A*	*B*	*A*	*B*
1	5	7	5	7	5	7	5	7
2	–	–	–	–	–	–	–	–
3	19	17	19	17	17	19	17	19
4	–	–	–	–	–	–	–	–
5	25	233	25	233	25	233	5	1,165
6	–	–	–	–	–	–	–	–
7	9	6,977	9	6,977	9	6,977	9	6,977
8	–	–	–	–	–	–	–	–
9	3	279,179	3	279,179	3	279,179	3	279,179
10	–	–	–	–	–	–	–	–
11	117	56,509	13	508,581	3	2,203,851	3	2,203,851
12	–	–	–	–	–	–	–	–

Dashes indicate the factors do not exist as *C* is prime

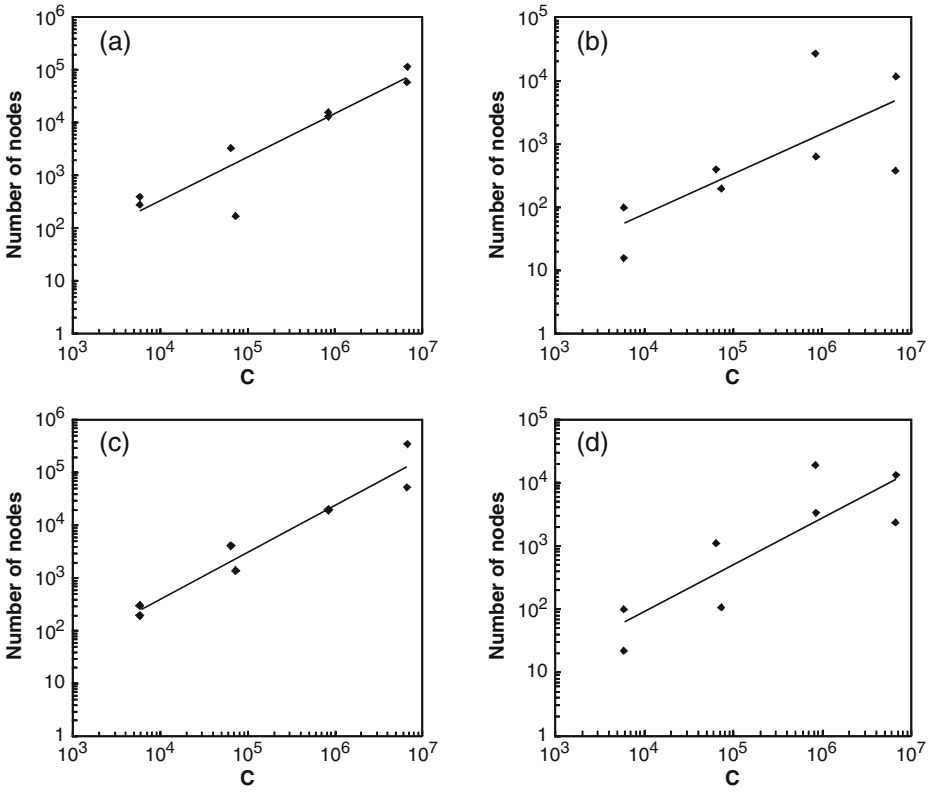

Fig. 3 Number of nodes in BB tree versus number *C* tested. **a** Model I, **b** Model II, **c** Model III and **d** Model IV

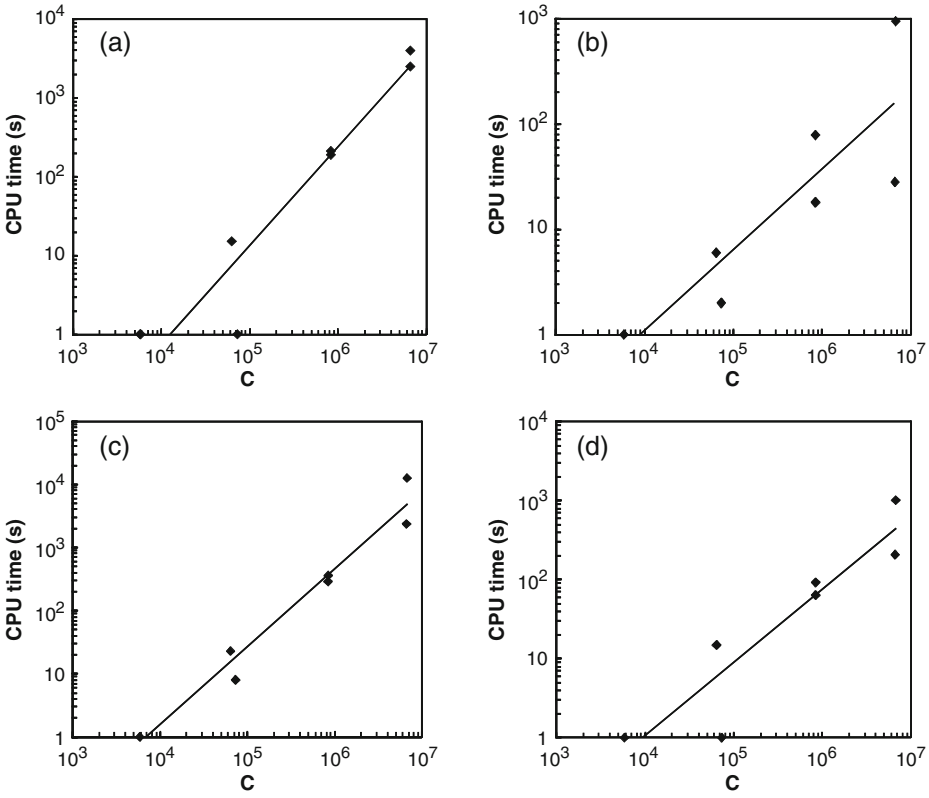

Fig. 4 Total CPU time (seconds) versus number C tested. **a** Model I, **b** Model II, **c** Model III and **d** Model IV

used in the BB trees, Fig. 4 shows the total CPU time usage, and Fig. 5 shows the average CPU time cost of each of the nodes in the BB trees.

The data plotted in all of these three figures show clearly an exponential trend which reveals the observed (experimental) computational complexity of the models, which is a function of both the models used as well as the specific internal implementation of the BB algorithm used in CPLEX. The data presented in these figures have been correlated with exponential functions and the results are summarised in Table 4.

4.6 Experimentally Determined Complexity of Number of Nodes in BB Tree

The experimentally determined average complexity of the number of nodes in the BB tree for the four proposed models is approximately $\mathcal{O}(C^{0.77})$. The best complexity is achieved for Model II which is approximately $\mathcal{O}(C^{0.63})$. The MIP formulations presented involve approximately $\mathcal{O}(\frac{3}{2}\log_2 C)$ binary variables. If the full combinatorial complexity of the problem were to be realised during solution, then the number of nodes in the BB tree would be approximately $\mathcal{O}(2^{\frac{3}{2}\log_2 C}) = \mathcal{O}(C^{1.5})$. It is evident that the number of nodes in increasing size problems grows exponentially with power 0.63

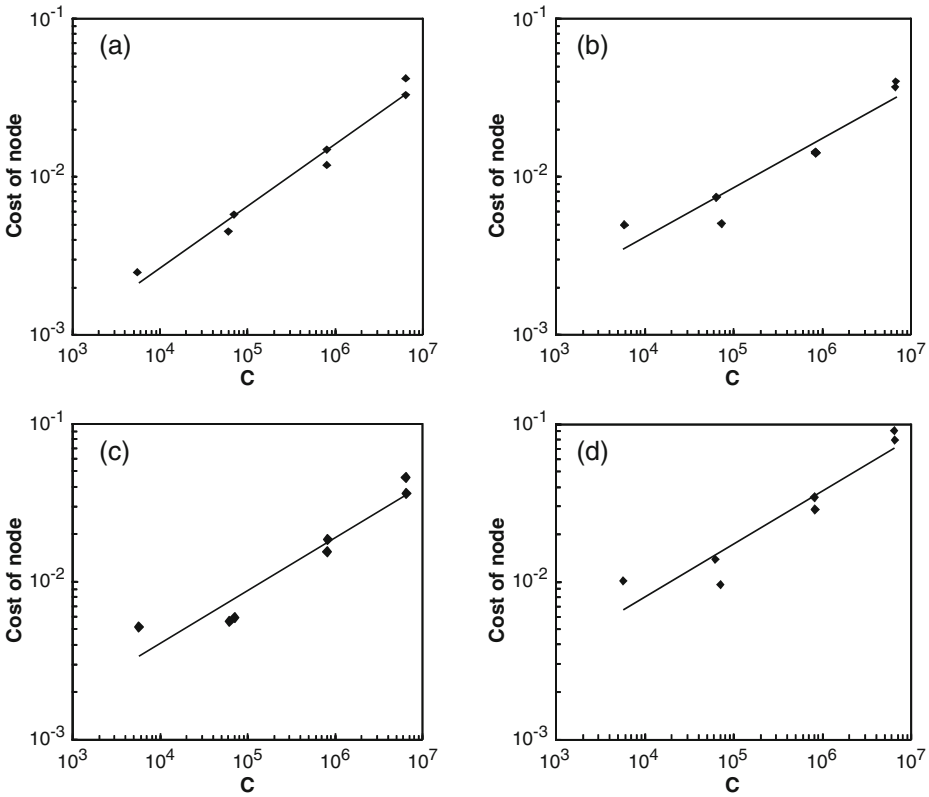

Fig. 5 Average CPU time (seconds) per node of BB tree versus number C tested. **a** Model I, **b** Model II, **c** Model III and **d** Model IV

(for Model II) which is smaller than the worst complexity expected for the models; in fact it is less than the square root, approximately, of the full combinatorial complexity of the MIP models.

The actual complexity of the original factorization algorithm considered is $\mathcal{O}(C^{0.5})$ which shows that computationally the present implementation we tested (and BB solver) results in slightly increased complexity. Thus computationally the models tested do not seem to offer an advantage with the present state of the formulations proposed.

The node complexity is what may be considered as the absolute complexity of the BB method. However as we shall see below the actual complexity as measured by the CPU time is of even higher value than this.

4.7 Experimentally Determined CPU Time Complexity of BB Runs

The experimentally determined CPU time average complexity of the CPU time in the BB trees for the four proposed models is approximately $\mathcal{O}(C^{1.04})$. The best complexity is achieved for Model II which is approximately $\mathcal{O}(C^{0.77})$. This is the actual cost of importance for practical application. Based on this, it is also clear that

Table 4 Fitted exponential models for resource usage indicators

Model	Fitted function	Correlation coefficient
Total nodes in BB tree		
I	$0.1759 \times C^{0.82}$	0.8496
II	$0.2523 \times C^{0.6268}$	0.5338
III	$0.1145 \times C^{0.889}$	0.9434
IV	$0.0949 \times C^{0.7467}$	0.7374
Total CPU time (seconds)		
I	$8.0 \times 10^{-6} \times C^{1.2413}$	0.9083
II	$0.0009 \times C^{0.7662}$	0.7444
III	$2.0 \times 10^{-5} \times C^{1.2403}$	0.9644
IV	$0.0002 \times C^{0.9319}$	0.8548
CPU time (seconds) per node of BB tree		
I	$7.0 \times 10^{-5} \times C^{0.3934}$	0.9754
II	$0.0005 \times C^{0.3156}$	0.9052
III	$0.0002 \times C^{0.3318}$	0.914
IV	$0.0004 \times C^{0.3355}$	0.8954

computational times for our case studies could become prohibitive if we decided to increase the number tested for factorability: based on the average complexity of all models an increase by 1 digit would result in a 10-fold increase of CPU times, while increasing by 3 digits would result in a 1000-fold increase in CPU time.

This complexity is higher than the one for the total number of nodes in the BB trees. This leads to the interesting question as to how what is the average complexity per node of the trees that the present implementation seems to involve. This is calculated next.

4.8 Experimentally Determined Average CPU Time Complexity Per Node of the BB Trees

By dividing the total CPU time by the total number of nodes in each run we calculate the average CPU time per node. This indicator is experimentally determined on average for the four proposed models to be approximately $\mathcal{O}(C^{0.34})$. The best complexity is achieved for Model II which is $\mathcal{O}(C^{0.32})$.

This shows that the average cost per node grows roughly with the cubic root of the size of the problem (magnitude of the integer number tested). Contrary to this we would have expected the Simplex method we used in our solver options to involve heuristically a cost that is proportional to the size of the problem tested, or at most a low power of it. Our models have a size complexity of $\mathcal{O}((\log C)^2)$, both for the size of total variables and constraints.

4.9 Overall Discussion of Experimental Results

From the above data it is clear that the best observed computational complexity is achieved for Model II. This contains the basic binary arithmetic constraints, with the addition of upper bounding constraints for the values of the variables of the formulation. Model III and Model IV which contain the constraints encoding that the factors are related by $A < B$ are more computationally intensive.

A particular interesting observation is also made for Case 9 and 10, ran with Model III and Model IV. Although the numbers are of the same number of digits and Case 9 is a factorable number while Case 10 is a prime number, the effort to solve the former (iterations, nodes and CPU time) is greater than for the latter. One possible explanation for this is that the number in Case 9 is actually larger than that of Case 10. Apart from this, the BB method is essentially a heuristic and the path followed during the implicit enumeration of the search tree can vary in unpredictable ways for cases where the formulation is effectively of the same size (the two numbers are very close to each other).

5 Conclusions

The main idea presented in the paper is the direct handling of arithmetic operations on integer numbers of arbitrary size represented encoded as binary strings, using binary variables to represent their digits within IP formulations. Often general integers are represented as summations of powers of two, multiplied by binary variables, and for this case we demonstrate how multiplication, addition and comparisons of magnitude of two numbers are encoded by appropriate integer constraints and cuts.

As an example, the factorization of an integer number C is considered and formulated as an optimization problem. The algorithm encoded is based on successive divisions of the number, and is known to have the worst complexity of all factorization methods. We encoded this algorithm as a set of integer constraints expecting the IP formulation to be somewhat better in terms of gross operations required (nodes in the BB enumeration tree). The case is that even as an IP formulation the algorithm is not competitive with other existing methods. It is notable that nowhere the formulation utilizes explicitly either the number C to be factorized or its square root, so ordinary arithmetic (as opposed to extended arithmetic for large numbers) mathematical programming software may be used even for numbers with a very large number of digits.

The formulation may require many BB iterations (nodes) to prove factorability as numerical experiments have shown, as the lower bounds obtained for the nodes may be too loose to fathom quickly parts of the BB tree. The numerical experiments conducted and reported in this work demonstrate clearly that this is the case for large numbers.

The IP formulation involves $(NA + NB + 2)$ binary variables. That approximately translates to $\mathcal{O}(\frac{3}{2}\log_2 C)$ binary variables, which yields a maximum complexity of all the combinations of the binary variables to be $\mathcal{O}(C^{3/2})$. This is higher than the brute-force division algorithm which is $\mathcal{O}(\frac{1}{2}\sqrt{C})$, but a lot of the combinations of the binary variables are infeasible due to the logical constraints imposed. This is verified by the computational results but unfortunately the resulting experimentally determined complexity is higher than that of the original factorization algorithm encoded in the MIP formulations.

The computational results obtained demonstrate the correctness of our formulation and its viability, however for the specific application of number factorization the computational performance was poor especially for large integers. This is primarily due to the fact that the objective function used does not convey any meaning related

to the underlying problem, and thus the lower bounding procedure is not tight enough to fathom BB tree nodes quickly.

In terms of the experimentally determined complexities of the overall implementation, the complexity in terms of nodes in the BB tree grows much slower than the full combinatorial complexity of the MIP models. However, when compounded with the also exponentially growing complexity that the experimental results indicate to be required at each node, the total CPU complexity is bigger. Overall, the node complexity alone is slightly bigger than that of the original factorization method considered and transcribed into the MIP models, which makes the present implementation non-competitive. The computationally determined complexities also make it clear that much larger integer numbers could not be tried with the present implementation as this would require excessive computational times.

The significance of this work is in the demonstration of the encoding of binary arithmetic directly into IP formulations and the derivation of suitable cuts to reduce the search space. In particular if large integers are represented as binary strings through the use of binary variables, it is demonstrated how basic operations like multiplication, addition and subtraction are implemented directly. It is further demonstrated how logical comparisons on the size of integer values are also achieved using this encoding, as for example with the comparison of the magnitude of factors A and B, through the two's complement subtraction method. The results highlight the necessity to include upper bounding logical constraints on continuous variables representing binary digits resulting from arithmetic operations, with significant impact on the reduction of the computational effort.

Further work following from what is presented in this paper needs to consider alternative tightening constraint schemes to improve the lower bounds generated by each LP solution at the nodes of the BB tree. This would increase the fathoming of nodes and accelerate convergence. Also, some consideration will be necessary in order to explain and improve the experimentally observed complexity of the average cost of each node.

Acknowledgements Funding from Onassis Foundation for Thomas A. Pogiatzis is gratefully acknowledged.

References

1. Adams, W.P., Henry, S.M.: Base-2 expansions for linearizing products of functions of discrete variables. Oper. Res. **60**(6), 1477–1490 (2012)
2. Agrawal, M., Kayal, N., Saxena, N.: Primes is in p. Ann. Math. **2**, 781–793 (2002)
3. Bienstock, D., Chopra, S., Günlük, O., Tsai, C.Y.: Minimum cost capacity installation for multicommodity network flows. Math. Program. **81**, 177–199 (1998)
4. Brooke, A., Kendrick, D., Meeraus, A.: GAMS: A User's Guide. Scientific Press, Palo Alto, CA (1988)
5. CPLEX: Using the CPLEX Callable Library. ILOG Inc., CPLEX Division, 930 Tahoe Blvd. #802-279, Incline Village, NV 89451-9436, USA (1997)
6. Dandamudi, S.P.: Fundamentals of Computer Organization and Design. Springer (2003)
7. Dietzfelbinger, M.: Primality Testing in Polynomial Time. Springer (2004)
8. Floudas, C.: Nonlinear and Mixed-Integer Optimization. Oxford University Press, 198, Madison Avenue, New York, New York 10016, USA (1995)
9. Grossmann, I.E.: Review of nonlinear mixed-integer and disjunctive programming techniques. Optim. Eng. **3**, 227–252 (2002)

10. Grossmann, I.E., Caballero, J.A., Yeomans, H.: Mathematical programming approaches to the synthesis of chemical process systems. Korean J. Chem. Eng. **16**, 407–426 (1999)
11. Harjunkoski, I., Grossmann, I.E.: Decomposition techniques for multistage scheduling problems using mixed-integer and constraint programming methods. Comput. Chem. Eng. **26**(11), 1533–1552 (2002)
12. Laporte, G., Nobert, Y.: A branch and bound algorithm for the capacitated vehicle routing problem. Oper. Res. Spectrum. **5**, 77–85 (1983)
13. Manin, Y.I., Panchishkin, A.A.: Introduction to Modern Number Theory. Springer (2007)
14. Méndez, C.A., Cerdá, J., Grossmann, I.E., Harjunkoski, I., Fahl, M.: State-of-the-art review of optimization methods for short-term scheduling of batch processes. Comput. Chem. Eng. **30**(6–7), 913–946 (2006)
15. Muldoon, F.M., Adams, W.P., Sherali, H.D.: Ideal representations of lexicographic orderings and base-2 expansions of integer variables. Oper. Res. Lett. **41**(1), 32–39 (2013)
16. Pochet, Y., Wolsey, L.: Production Planning by Mixed Integer Programming (Springer Series in Operations Research and Financial Engineering). Springer-Verlag New York, Inc., Secaucus, NJ, USA (2006)
17. Ralphs, T.K., Kopman, L., Pulleyblank, W.R., Trotter, L.E.: On the capacitated vehicle routing problem. Math. Program. **94**, 343–359 (2003)
18. You, F., Grossmann, I.E.: Mixed-integer nonlinear programming models and algorithms for large-scale supply chain design with stochastic inventory management. Ind. Eng. Chem. Res. **47**(20), 7802–7817 (2008)