FROM F_2 TO $\mathbb{Z}$ SOLUTIONS OF BRENT EQUATIONS

A PREPRINT

Axel Kemper [*]

May 24, 2025

ABSTRACT

Solutions of Brent Equations represent matrix multiplication algorithms.
We describe a method to generalize F_2 solutions to coefficient ring $\mathbb{Z}$. This is relevant, because only solutions with coefficients in domain $\mathbb{Z}$ can be turned into practically applicable algorithms.
Recent results of Moosbauer and Poole make use of Hensel lifting. Their resulting schemes are using coefficients in $\{-2, -1, +1, +2\}$.
Our transformation from F_2 to $\mathbb{Z}$ is based on constraint programming. We make use of the Python interface *z3py* of Microsoft's *Z3* solver to find integer solutions for Brent Equations.
As a practical result, we present new algorithms with coefficients in $\{-1, +1\}$ for general matrix multiplication $\langle 4 \times 5 \times 6\_90 \rangle$ and $\langle 5 \times 5 \times 5\_93 \rangle$ and $\langle 6 \times 6 \times 6\_153 \rangle$.

Keywords matrix multiplication · Brent Equations

1 Introduction

As proposed by Brent [5], a bilinear non-commutative algorithm to multiply two matrices $[a_{ij}]$ and $[b_{mn}]$ with r elementary products can be written as:

$$[c_{pq}] = \sum_{k=1}^{r} \gamma_{pq}^{(k)} \sum_{ij} \alpha_{ij}^{(k)} a_{ij} \sum_{mn} \beta_{mn}^{(k)} b_{mn} \tag{1}$$

If an algorithm multiplies an $M \times N$ matrix with an $N \times P$ matrix with r elementary products, it is denoted in the following as $\langle M \times N \times P\_r \rangle$.

Brent discovered that the coefficients can be derived as solution to the following set of *Brent Equations*:

$$\forall i \forall j \forall m \forall n \forall p \forall q \quad \sum_{k=1}^{r} \alpha_{ij}^{(k)} \beta_{mn}^{(k)} \gamma_{pq}^{(k)} = \delta_{ip} \delta_{jm} \delta_{nq} \tag{2}$$

In equation (2), *Kronecker delta* δ_{xy} is 1 for $x = y$ and 0 for $x \neq y$.

Brent Equations are highly non-linear. For the time being, this allows solutions only for very small matrix dimensions.

As a workaround, [14] suggested to firstly solve the set of equations modulo 2. The second step is then expected to be easier, because the number of variables and their domain are reduced. This approach was also taken by [6], to find a solution for $\langle 3 \times 3 \times 3\_23 \rangle$. Such a solution was published by Laderman in [13] 35 years earlier without disclosing the solution method.

In recent years, many solutions for $\langle 3 \times 3 \times 3\_23 \rangle$ where found and published. [9] [10] found thousands of solutions and used Gröbner bases to lift the schemes from coefficient domain $\mathbb{Z}_2$ to $\mathbb{Z}$.

[*]Contact `axel.kemper@gmail.com`, orcid 0009-0004-7474-7171

With worldwide public attention [7] published a solution for $\langle 4 \times 4 \times 4\_47 \rangle$. It went largely unnoticed by the media that this was a $\mathbb{Z}_2$ solution. Despite many attempts, it turned out to be impossible to lift this to a general $\mathbb{Z}$ solution. That is unfortunate, because $\langle 4 \times 4 \times 4\_47 \rangle$ would be exponentially superior compared to Strassen's groundbreaking $\langle 2 \times 2 \times 2\_7 \rangle$ solution [17] from 1969.
Note: $7 = 2^{2.80735\ldots}, 47 = 4^{2.7773\ldots}$

In [16], new solutions for $\langle 5 \times 5 \times 5\_93 \rangle$ and $\langle 6 \times 6 \times 6\_153 \rangle$ were presented. This included lifted versions for coefficient domain $\mathbb{Z}$. These were found via Hensel lifting [20]. However, the lifted solutions make use of coefficients in $\{-2, -1, +1, +2\}$. Factors ± 2 make them less attractive for practical calculations.

Coincidently, [1] also found a solution for $\langle 5 \times 5 \times 5\_93 \rangle$. Their description leaves open, how the lifting was accomplished. The solution makes use of coefficients in $\{-1, +1\}$.

Another solution of [1] is for $\langle 4 \times 5 \times 6\_90 \rangle$ with coefficients in $\{-2, -1, +1 + 2\}$. We converted this solution back to $\mathbb{Z}_2$ and lifted it to a solution with coefficients in $\{-1, +1\}$. It was not possible for us to do the same for the solutions for $\langle 3 \times 5 \times 6\_68 \rangle$ and $\langle 3 \times 5 \times 7\_80 \rangle$

Summing it up, the lifting process has not received much attention. It was hardly mentioned and described in detail. This note aims to fill this gap and provide a fully automated lifting method with moderate implementation complexity.

2 Hensel lifting

In [11], [15], [16] Hensel lifting is described as a method to generalize solutions of Brent Equations. The referenced textbook is [20].

Starting from equation (2), the following ansatz is made modulo 2^s:

$$\forall i \forall j \forall m \forall n \forall p \forall q \quad \sum_{k=1}^{r} (\alpha_{ij}^{(k)} + \hat{\alpha}_{ij}^{(k)} 2^s)(\beta_{mn}^{(k)} + \hat{\beta}_{mn}^{(k)} 2^s)(\gamma_{pq}^{(k)} + \hat{\gamma}_{pq}^{(k)} 2^s) \equiv \delta_{ip}\delta_{jm}\delta_{nq} \mod 2^{s+1} \tag{3}$$

This leads to:

$$\forall i \forall j \forall m \forall n \forall p \forall q \quad \sum_{k=1}^{r} \alpha_{ij}^{(k)} \beta_{mn}^{(k)} \hat{\gamma}_{pq}^{(k)} + \alpha_{ij}^{(k)} \hat{\beta}_{mn}^{(k)} \gamma_{pq}^{(k)} + \hat{\alpha}_{ij}^{(k)} \beta_{mn}^{(k)} \gamma_{pq}^{(k)} = 0 \tag{4}$$

The ansatz (3) turns a system of nonlinear equations into a system (4) of linear homogeneous equations over $\mathbb{Z}_2$.

The system is underdetermined. There are fever variables than equations. Therefore, one of the possible solutions has to be selected, used or refined by lifting it to higher modulo 2^s. [15] reports that *Mathematica* function `LinarSolve` was used with s up to $s = 100$.

3 Gröbner bases

Gröbner bases [20] are used for solving systems of polynomial equations.

[10] describe how they made use of Gröbner bases to generalize the coefficient ring of solutions to Brent Equations. They mention `Mathematica` and `Singular` as engines to perform these calculations. The reader is referred to the cited papers for details.

4 Lifting Brent Equations with constraint programming

The $\mathbb{Z}_2$ solution of the $\langle M \times N \times P\_r \rangle$ problem results in a set of coefficients in $\{0, +1\}$. Coefficients equal to 0 vanish from the solution. Their polarity does not need to be determined. Coefficients equal to 1 can either be $+1$ or -1 in the lifted solution.

There are $M^2 \times N^2 \times P^2$ equations. Every equation has r summands of the form $\alpha_{ijk}\beta_{mnk}\gamma_{pqk}$. Each of the summands has three coefficient factors. They are therefore called *triples* in the following.

The total number of coefficient variables is $r \times (M \times N + N \times P + M \times P)$. As most of the variables of the $\mathbb{Z}_2$ solution tend to be 0, the $\mathbb{Z}$ solution has far fewer variables.

4.1 Algorithm to define search constraints

```
""" Input:  coefficients of Z2 solution
    3D matrices: alpha, beta, gamma
    Output: solver assertions
    i = rows of [A], j = columns of [A]
    m = rows of [B], n = columns of [B]
    p = rows of [C], q = columns of [C] """
for all combinations of i, j, m, n, p, q:
        triples = {}    """ empty set of triples """
        for k in range(r):
                if alpha[i][j][k] and beta[m][n][k] and gamma[p][q][k]:
                        """ add new Xor() to set of triples
                            not shown: create variables on the fly
                                       re-use reoccurring expressions
                                       Xor(a[i][j][k], b[m][n][k]) """
                        t = Xor(a[i][j][k], b[m][n][k], c[p][q][k])
                        assert(x == t)
                        triples.Add(x)
        cardinality(triples)
```

Listing 1: Define search constraints

4.2 Algorithm to define cardinality constraints

```
""" Input:  collection of triples
    Output: solver assertions  """
n = len(triples)
if n == 0:
    """ nothing to do: no polarities to decide upon """
elif n == 1:
    """ single triple must be true """
    assert(triples[0] == true)
elif n == 2:
    """ two triples of opposite signs """
    assert(triples[0] != triples[1])
else:
    """ ceil() rounded-up integer division """
    no_of_trues = (n + 1) // 2
    assert(Sum(triples) == no_of_trues)
```

Listing 2: Define cardinality constraints

The algorithms 4.1 and 4.2 sketched above rely on a constraint solver which is capable to process *Xor* and *cardinality* constraints.

Xor constraints assert that a Boolean variable is equal to the exclusive *Or* of two other variables. The variable is *true*, if the input variables have opposite values.

Cardinality constraints used for our transformation assert that the number of *true* Boolean variables in a given collection is equal to a specified value.

It is rather tedious but also possible to encode these constraints in *Conjunctive Normal Form* suitable as input for a satisfiability solver. Examples for such solvers include *Clasp* [8] and *CaDiCal* [2].

5 Implementation

The constraint programming approach was implemented in Python 3.13. Package *z3py* [4] served as constraint solver. It is based on Microsoft's Z3 SMT Solver (version 4.14, 64bit) [3]. A Windows 11 PC with Intel® Core(TM) i7-14700 CPU @ 2.10 GHz was used in single-threaded mode.

Boolean variables are created on the fly, when needed. A variable is only useful if its $\mathbb{Z}_2$ counterpart is part of a non-false triple conjunction of three variables.

Rather than defining all *Xor* constraints with three variables, common subexpressions are detected and re-used. Many solvers do detect such re-occurrences and eliminate them automatically.

To get the variable values of the $\mathbb{Z}_2$ solution, a routine was developed to transform the tensor format of [16] into 3D arrays. The three dimensions are matrix row, matrix column and product number.

The solutions of [12] were translated from $\mathbb{Z}$ to $\mathbb{Z}_2$ before they were lifted to $\mathbb{Z}$ again with a reduced set of coefficients.

5.1 Implementation with MiniZinc

A second implementation makes use of the *MiniZinc* framework [18] for constraint programming. We translated the system of Brent Equations into a *MiniZinc* model. This model was then solved for some of the examples. For others, the solver returned an `UNSATISFIABLE` status. *Google OR Tools* [19] served as *MiniZinc* back-end solver. The run-times were comparable to the *Z3* run-times.

Using *MiniZinc* reduced the effort to experiment with different model encodings:

Encoding	Description
Variables in $\{-1, +1\}$	triple product is simple $a_{ijk} * b_{mnk} * c_{pqk}$
Variables in $\{0, +1\}$	product is $(1 - 2 * a_{ijk}) * (1 - 2 * b_{mnk}) * (1 - 2 * c_{pqk})$
Boolean variables	triple product is modelled as *XOR3*

The sobering result is, that all these different encodings do not make a big difference. They have not yielded significantly better results or shorter run-times.

6 Results

The new method was applied to lift the $\mathbb{Z}_2$ solutions of [16] for $\langle 5 \times 5 \times 5\_93 \rangle$ and $\langle 6 \times 6 \times 6\_153 \rangle$. This was successful in both cases. The original $\mathbb{Z}_2$ and $\mathbb{Z}$ solutions of [16] are available for download in GitHub.

As answer to [16], new schemes for various rectangular matrix formats were presented in [12]. Six of these could be lifted successfully with our new method. They are also listed in the table below.

Example: The solution of [12] for $\langle 5 \times 6 \times 6\_130 \rangle$ has coefficients in $\{-4, -2, -1, +1, +2, +3, +4, +5, +8, +9\}$. The new solution has coefficients in $\{-1, +1\}$.

Note for the quadratic schemes: $93 = 5^{2.81626...}$, $153 = 6^{2.80754...}$
The new solutions are exponentially inferior to Strassen's $\langle 2 \times 2 \times 2\_7 \rangle$. To be superior, it would need a $\langle 5 \times 5 \times 5\_91 \rangle$ solution or a $\langle 6 \times 6 \times 6\_152 \rangle$ solution.

Problem	Run-time	Assertions	Variables	Xor
$\langle 4 \times 5 \times 6\_90 \rangle$	1.1	1,375	975	5,266
$\langle 5 \times 5 \times 5\_93 \rangle$	3.5s	2,372	1,224	9,483
$\langle 4 \times 6 \times 6\_106 \rangle$	3.0s	3,735	1,528	15,130
$\langle 5 \times 5 \times 6\_110 \rangle$	3.6s	3,728	1,575	14,929
$\langle 5 \times 5 \times 7\_127 \rangle$	6.9s	4,045	1,807	16,304
$\langle 5 \times 6 \times 6\_130 \rangle$	8.8s	4,629	1,956	19,648
$\langle 5 \times 6 \times 7\_150 \rangle$	14.9s	6,883	2,467	29,824
$\langle 6 \times 6 \times 6\_153 \rangle$	50.2s	7,662	2,574	30,536

In the table above, *Variables* are not including internal switching variables. Only the variables of the $\mathbb{Z}$ solution are counted.

Run-times with *Yices 2* as SMT solver were in the same region of the run-times reported above. The run-times listed are the times used by the *Z3* solver [3].

It was not possible for us to lift the $\langle 4 \times 4 \times 4\_47 \rangle$ $\mathbb{Z}_2$ solutions of [7]. The solver returned *unsatisfiable* right away.

The collected solutions are available for download from *GitHub*:

`https://github.com/a1880/matrix-multiplication`

7 Conclusion

There is still no established method to solve Brent Equations for $\mathbb{Z}_2$ for matrix dimensions beyond 3×3. The lifting of an existing solution from $\mathbb{Z}_2$ to $\mathbb{Z}$ does take considerably less time and effort. It was shown in this paper, that solutions for $\langle 5 \times 5 \times 5\_93 \rangle$ and $\langle 6 \times 6 \times 6\_153 \rangle$ can be lifted in less than a minute on modest hardware.

However, not every solution can be lifted [7]. There is no guarantee, that a $\mathbb{Z}_2$ solution actually has a generalization for $\mathbb{Z}$.

It remains to be shown, which lifting technique is suitable for which problem. Candidates mentioned in the literature are Hensel lifting [16] and Gröbner bases [9]. The constraint programming approach described in this paper has the advantage, that coefficients out of the set $\{-1, +1\}$ can be found, provided such a solution exists.

References

[1] M. Balog, A. Novikov, P. Kohli, and colleagues, *Alphaevolve: A coding agent for scientific and algorithmic discovery*, (2025).

[2] A. Biere, *Cadical at the sat race 2019*, (2019).

[3] N. Bjørner and colleagues, *The z3 theorem prover*, 2025.

[4] N. Bjørner and A. Dutcher, *z3-solver 4.14.1.0*, 2025.

[5] R. Brent, *Algorithms for matrix multiplication*, (1970).

[6] N. Courtois, G. Bard, and D. Hulme, *A new general-purpose method to multiply 3x3 matrices using only 23 multiplications*, (2011).

[7] A. Fawzi and colleagues, *Discovering faster matrix multiplication algorithms with reinforcement learning*, nature 610, (2022).

[8] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, *Answer Set Solving in Practice*, Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool, 2012.

[9] M. Heule, M. Kauers, and M. Seidl, *Local search for fast matrix multiplication*, Conference Theory and Applications of Satisfiability Testing - SAT 2019, (2019).

[10] M. J. Heule, M. Kauers, and M. Seidl, *New ways to multiply 3× 3-matrices*, Journal of Symbolic Computation, 104 (2021), pp. 899–916.

[11] M. Kauers and J. Moosbauer, *Flip graphs for matrix multiplication*, (2023).

[12] M. Kauers and I. Wood, *Consequences of the moosbauer-poole algorithms*, (2025).

[13] J. Laderman, *A non-commutative algorithm for multiplying 3x3 matrices*, Bull. Amer. Math. Soc. Volume 82, Number 1, (1976).

[14] C. M. Li, B. Jurkowiak, and P. W. Purdom, *Integrating symmetry breaking into a dll procedure*, International conference on theory and applications of satisfiability testing, (2002), pp. 149–155.

[15] J. Moosbauer, *Search Techniques for Matrix Algorithms*, Johannes Keppler University Linz, 2023.

[16] J. Moosbauer and M. Poole, *Flip graphs with symmetry and new matrix multiplication schemes*, (2025).

[17] V. Strassen, *Gaussian elimination is not optimal*, Numer. Math. 13, (1969), pp. 354–356.

[18] P. Stuckey, K. Marriott, and G. Tack, *Minizinc handbook*, 2025.

[19] G. Turovsky, *Solving optimization problems with minizinc and google or tools*, 2024.

[20] J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*, Cambridge University Press, 2013.

AXEL KEMPER, D-31832 SPRINGE, GERMANY
Email address: axel.kemper@gmail.com