An Efficient Reversible Algorithm for Linear Regression

Erik D. Demaine
Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology,
Cambridge, MA, USA
edemaine@mit.edu

Jayson Lynch
Cheriton School of Computer Science, University of Waterloo,
Waterloo, ON, Canada
jayson.lynch@uwaterloo.ca

Jiaying Sun
Miss Porter's School
Farmington, Connecticut, USA

***Abstract*—This paper presents an efficient reversible algorithm for linear regression, both with and without ridge regression. Our reversible algorithm matches the asymptotic time and space complexity of standard irreversible algorithms for this problem. Needed for this result is the expansion of the analysis of efficient reversible matrix multiplication to rectangular matrices and matrix inversion.**

***Index Terms*—Reversible Computing, Algorithms, Machine Learning**

I. Introduction

Machine learning has become a major tool across many domains and part of the increasing success and capabilities can be attributed to more data and computing power. The energy demands of computing have been exponentially increasing despite exponential increases in energy efficiency [1], [2]. Reversible computing is a technology that has the potential to drastically reduce the energy consumption of computers and is a necessary technology for continued efficiency gains in the long term [3]. However many theoretical questions remain open about reversible computing and many engineering challenges stand in the way of taking advantage of these potential gains. By studying reversible algorithms for machine learning problems we gain a better understanding of theoretical aspects of logical reversibility which we hope will aid future work leading to greener computing.

In this paper we give asymptotically efficient algorithms for solving systems of linear equations and for computing linear regressions. Section II-A extends the analysis of reversible matrix multiplication to apply to rectangular matrix multiplication. Section II-B gives an efficient reversible algorithm for matrix inversion. Section III-A uses the prior results to give an efficient reversible algorithm for ordinary least squares regression and Section III-B does the same for ridge regression.

A. Related Works

Although there is a limited number of previous studies on reversible computing for machine learning, there is work on reversible algorithms. We now describe some of the fascinating work on reversible algorithms more broadly.

Supported by NSERC.

Reversible Neural Networks: Reversible or invertible machine learning algorithms have also been studied without the context of reversible computing. For example algorithms for invertible neural nets have been developed to try to save on the memory requirements for training various types of neural nets [4]–[8]. These have gone on to find other uses including the development of involutive neural nets and other architectures which enforce certain group symmetries [9]. Although the level of reversibility in these schemes does not suffice for the needs of reversible computing, they certainly make things much easier by ensuring large logical portions of the algorithm are bijective. Hopefully techniques in reversible algorithm design can inspire further techniques.

Universal Transforms: The first universal transforms for reversible computing were independently discovered by Lecerf and Bennett [10] [11] who make functions bijective by storing inputs at every computation step giving a time complexity $T'(n) = O(T(n))$ and a space complexity $S'(n) = O(S(n)T(n))$. In 1979, Bennett [11] developed a recursive recomputing, which involves recursively computing to the midpoint of the computation, storing it, and uncomputing to the last checkpoint. The time complexity for this would be $T'(n) = O(T(n)lg(T(n)))$ and space complexity as $S'(n) = O(S(n)lg(T(n)))$. Lange, Mckenzie, Tapp in 2000 [12] gave an algorithm which walked the entire computation tree and is reminiscence of Savitch's Algorithm giving a time complexity of $T'(n) = O(2^{T(n)})$ and a space complexity of $S'(n) = O(S(n))$. These techniques were combined by Williams in 2000 and Buhrman, Li, Tromp, and Vitanyi [13] in 2001 to give a trade-off relationship between time and space. By embedding configuration space enumeration at the bottom of a Bennett recursion they give a family of algorithms for any parameter k in which the time complexity would be $T'(n) = O(S(n)k(T(n)/2^k))$ and time complexity would be $S'(n) = O(kS(n))$. Xu in 2015 [14] investigated computing with 'dirty ancilla bits' and determined that space can be temporarily reused without knowing its prior state giving a time complexity of $T'(n) = O(2^{T(n)})$ and a space complexity of $S'(n) = S(n) + O(1)$. The study of efficient reversible algorithms for specific problems (as opposed to these universal transforms) has been a more recent area of study and some of this work is described in the following paragraphs.

Energy Complexity of Algorithms: Demaine et al. [15] studied energy variations of computations on circuit RAM, word RAM, and trans-dichotomous RAM. Their paper is the first that performs a systematic study of time/space/energy complexity of reversible computing. In contrast to most previous investigations, they allow algorithms to destroy bits. They developed a high-level pseudocode for fundamental models, such as control logic, memory allocation, garbage collection, and logging and unrolling, and demonstrated that these programming structures may be implemented with zero energy overhead in time and space. Li et al. [16] also investigated time, space, and erasure complexity of computation from an information theoretic viewpoint. They proved in their paper that Bennett's pebbling strategy uses the least space complexity for the greatest efficiency in the pebbling model. Comparing a restatement in terms of $E^t(\cdot)$ as

$$E^{t'}(x, \in) \leq C^t(x) + 2C^t(t|x) + 4 \log C^t(t|x)$$

with $t'(|x|) = O(2^{|x|}t(|x|)$ to the righthand inequality of equation $E^t(x, \in) \geq C^t(x) \geq \frac{1}{2}E^{t'}(x, \in)$, they have improved the upper bound on erasure cost. They also established a trade-off hierarchy of time versus irreversibility for exponential time computations:

For every large enough n there is a string x of length n and a sequence of $m = \frac{1}{2}\sqrt{n}$ (exponential) time functions $t_1(n) < t_2(n) < \cdots < t_m(n)$, such that

$$E^{t_1}(x, \in) > E^{t_2}(x, \in) > \cdots > E^{t_m}(x, \in).$$

Comparison Sorts: Axelsen et al. [17] proposed the following criteria for determining optimality of reversible simulation: A reversible simulation is faithful if it incurs no asymptotic time overheard and bounds the space overhead (the garbage) by some function $g(n)$, and they also showed how to use programming approaches to create reversible simulations of many wells that are both accurate and hygienic (minimizing garbage bits) of several comparison sorting algorithms, including insertion sort and quicksort. They also included prior works on reversible sorting: In $MOQA$ quicksort, there appears a rank (a number in the range $[0, n! - 1]$ indicating the permutation) for demonstrating injectivity. Yokoyama et al. used rankings to generate a reversible insertion and merge sorts [18]. Lutz et al. used a permutation approach to attempt for a reversible bubble sort [19]. Sorting was also considered by Frank [20] and Demaine et al. [15].

Shortest-Path Algorithms: Guo et al. [21] presented several reversible algorithms along with code for shortest paths problems. These include reversible Bellman-Ford algorithm, reversible Dijkstra's algorithm, and reversible Floyd-Marshall algorithm. Most of the reversible programs they have examined have asymptotically minimal additional space, and therefore the resulting time complexity is the same as their irreversible counterparts.

Frank's thesis [20] examined sorting algorithms, which he discovered that any standard sorting algorithms have $\Theta(n \log n)$-time comparison, and are easy to turn into good reversible algorithms by saving away bits giving the result of each comparison. Moreover, he also investigated the Floyd-Warshall algorithm which takes $\Theta(n^3)$ time, $\Theta(n^2)$ space. He also examines a reversible iterated matrix multiplication algorithm giving reversible all-pairs shortest path in $\Theta(n^3 \log n)$ time and $\Theta(n^2 \log n)$ space.

Demaine et al. [15] also examined Bellman-Ford and Floyd-Warshall algorithms which they found that Reversible Bellman-Ford algorithm runs in $\Theta(VE)$ time, $\Theta(VE)$ space, and 0 energy and the Reversible Floyd-Warshall algorithm runs in $\Theta(VE)$ time, $\Theta(VE)$ space, and 0 energy.

Implementations of many graph algorithms in Janus were provided in [22].

II. Reversible Matrix Algorithms

In this section we extend Frank's reversible matrix multiplication algorithm to rectangular matrices (Section II-A) and we give an efficient reversible algorithm for computing the inverse of a matrix and for matrix inversion (Section II-B). Matrix multiplication, transpose, and inverse will be the primary components needed to perform linear regression.

A. Rectangular Matrix Multiplication

Efficient reversible algorithms for square matrix multiplication are discussed in [20] and [15]. The same tricks suffice for making an efficient reversible algorithm for rectangular matrix multiplication.

Given an $m \times n$ and a $p \times n$ matrix we wish to compute the resulting $p \times m$ matrix in time:

$$T(MM(n, m, p)) = O(nmp)$$

and space:

$$S(MM(nmp)) = O(mn + pn + pm)$$

The standard algorithm runs through all entries of the output matrix and calculates a dot product between the associated vectors. If we use either of the standard universal transforms for reversible algorithms we will either end up with $O(nmp)$ space or a logarithmic overhead in space and time. However, we can treat the innermost dotproduct as a reversible subroutine to save space.

```
RegularMatrixMultiplication():
    for $i = 1$ to $m$:
        for $j = 1$ to $p$:
            for $k = 1$ to $n$:
                $C_{i,j}$ += $A_{i,k} * B_{k,j}$
```

Given a reversible algorithm which calculates some function f, we can calculate the output of the function, copy it into another location in memory, then uncompute the function. To remove the output of the reversible subroutine we will need to recalculate the output. Thus at a cost of doubling the time needed we can reduce the storage needed for intermediary calculation to the size of their inputs and outputs. This technique has been observed in some fashion at least since [23] and has been used whether explicitly or implicitly in most efficient reversible algorithms.

We give pseudocode below which considers the innermost for-loop a reversible subroutine. The individual multiplications of the dot product, $A_{i,n} * B_{n,j}$, are stored into $temp\_mult$ and then added into $C_{i,j}$. After each multiplication we can recompute the multiplication freeing the temporary stored value. For the backwards computation, the dot product is recalculated and stored into $temp\_mult$, where $C_{i,j}$ is zeroed by subtracting off each element of the dot product. This whole process takes up less space as $O(pm)$ because now we don't have to waste extra space to store the intermediate values. However, although the multiplication is done four times we maintain the same asymptotic running time of $O(nmp)$.

```
REVERSIBLEMATRIXMULTIPLY():
    Forwards:
    for $i = 1$ to $m$:
        for $j = 1$ to $p$:
            for $k = 1$ to $n$:
                $(temp\_mult, A_{i,k}, B_{k,j}) =$
                $(A_{i,k} * B_{k,j}, A_{i,k}, B_{k,j})$ # forwards
                $C_{i,j} \mathrel{+}= temp\_mult$
                $temp\_mult \mathrel{-}= A_{i,k} * B_{k,j}$ # backwards

    Backwards:
    for $i = 1$ to $m$:
        for $j = 1$ to $p$:
            for $k = 1$ to $n$:
                $(temp\_mult, A_{i,k}, B_{k,j}) =$
                    $(A_{i,k} * B_{k,j}, A_{i,k}, B_{k,j})$
                    # recalculate $A_{i,k} * B_{k,j}$
                $C_{i,j} \mathrel{+}= temp\_mult$
                $temp\_mult \mathrel{-}= A_{i,k} * B_{k,j}$ # reverse
```

B. Gaussian Elimination and Matrix Inversion

In this section we give a reversible version of Gaussian Elimination for computing the inverse of a matrix. One might hope the same technique for matrix multiplication will simply work with a standard matrix inversion algorithm. However, that is not the case and we will briefly examine the issue with Gaussian Elimination before giving a modification to that algorithm which will admit an efficient reversible subroutine and adapt into an efficient reversible algorithm. Pseudocode for the modified Gaussian Elimination is given below.

Gaussian elimination uses row operations to transform a matrix into the Identity. If these same operations are applied to an identity matrix, a vector of unknowns, or other entities this method can be used to solve systems of linear equations or calculate matrix inverses or LU decompositions [24]. A basic version of the algorithm starts by looping over all rows in the matrix, normalizing the first non-zero entry, and then subtracting multiples of that row from all the other rows to zero out all but one entry in a column of the matrix. If we look at the innermost loop, subtracting a multiple of one row from another, we have a sub-routine which is called $O(n^2)$ times and writes over an entire row in the matrix. Since its output space is $O(n)$ we would still need to store a total of $O(n^3)$ space to reverse all of these operations. If we move outward and consider each row zeroing a column of the matrix, we have $O(n)$ calls to a subroutine that writes over the entire matrix. Thus the output space is $O(n^2)$ for each call of the subroutine and we again do not save any space. For matrix multiplication we were able to save because the output space of the dot product was a single number, much smaller than the total running time of the subroutine.

```
MODIFIED GAUSSIAN ELIMINATION($A$)
    # Identity matrix which will be modified into the inverse
    Inverse_$A$ = Identity(size($A$))
    # Loop over all rows
    for $i = 1$ to $n$:
        # Loop over all prior rows
        for $j = 1$ to $i$:
            # Calculate multiplier needed to zero out entry $j$
            row_multiplier $= A_{i,j}/A_{j,i}$
            for $k = 1$ to $n$:
                # Write over entries **in row $i$**
                $A_{i,k} \mathrel{-}= A_{j,k} *$ row_multiplier
                Inverse_$A_{i,k} \mathrel{-}= A_{j,k} *$ row_multiplier
        # normalize based on entry $i$
        reducing_multiplier $= A_{i,i}$
        for $k = i$ to $n$:
            $A_{i,k} = A_{i,k}$ / reducing_multiplier
            Inverse_$A_{i,k} = A_{i,k}$ / reducing_multiplier

        # Loop backwards over all rows
        for $i = n$ to $1$:
            # Loop over all prior rows
            for $j = i + 1$ to $n$:
                # Calculate multiplier needed to zero out entry $j$
                row_multiplier $= A_{i,j}$
                for $k = 1$ to $n$:
                    # Write over entries **in row $i$**
                    $A_{i,k} \mathrel{-}= A_{j,k} *$ row_multiplier
                    Inverse_$A_{i,k} \mathrel{-}= A_{j,k} *$ row_multiplier
        return Inverse_$A$
```

Now we describe the key idea in the modified version of the algorithm. Again just considering the first part of the algorithm, converting the matrix to row echelon form. To solve this problem we will reorder the algorithm so that in each loop we have a single row calculate all the updates it needs, thus making changes to only one row on each iteration. In this case we have a subroutine that is still called $O(n)$ times and performs $O(n^2)$ operations, but has an output of size $O(n)$ allowing us to save garbage bits by recalculating. One can think of this as each row pulling information from $O(n)$ other rows to calculate the effect from every other row of the matrix on itself, as opposed to the standard algorithm where each row pushes the information of its impact to every other row.

The second half of the algorithm again goes over all the rows using the previous reduced rows to cancel out remaining entries in the row. Again we make all updates to the same row in the same loop so we have a reversible subroutine with output size $O(n)$ and time $O(n^2)$ which is called $O(n)$ times. After this transformation has finished the program returns the new inverse matrix.

We have not addressed pivoting or other techniques to improve numerical stability or efficiency. Common techniques such as swapping rows based on zero valued entries or the size of the leading entries now cannot occur because partially reduced rows cannot be compared since each row goes form being unchanged to fully reduced before any other rows are altered. Addressing these techniques as well as others used to make matrix inversion more efficient and robust are important challenges moving forward.

III. Linear Regression

Linear regression is a predictive model with a linear hypothesis class.[1]

$$h(x; \theta, \theta_0) = \theta^T x + \theta_0,$$

Typically one minimizes the squared error, so least squares linear regression wants to minimize the following loss function with respect to θ.

$$J(\theta, \theta_0) = \frac{1}{n} \sum_{i=1}^{n} \left(\theta^T x^{(i)} + \theta_0 - y^{(i)} \right)^2$$

where $y(i)$ is the actual value and $\theta^T x^{(i)} + \theta_0$ is the predicted value. This can be solved analytically, and we give an efficient reversible algorithm in Section III-A. In Section III-B we give an efficient reversible algorithm for ridge regression, a common form of regularized linear regression.

A. Ordinary Least Squares

Ordinary least squares, or linear least squares, estimates the parameters in a regression model by minimizing the sum of the squared losses. We can achieve this by finding a closed-form formula and we can first set the derivative of J to 0, and solve for θ. When the derivative of J equals to 0, J can be either the minimum or maximum, so we have to make sure the point we are calculating is not the maximum or an inflection point.

Training data can be expressed in the form of matrices. Each column of X is a measure and each y is a predicted data, where d is the dimension of the matrices and n is the number of x columns.

$$X = \begin{bmatrix} x_1^{(1)} & \dots & x_1^{(n)} \\ \vdots & \ddots & \vdots \\ x_d^{(1)} & \dots & x_d^{(n)} \end{bmatrix}; \; Y = \begin{bmatrix} y^{(1)} & \dots & y^{(n)} \end{bmatrix}.$$

[1]We follow the notation used in [25].

We can also transpose the matrices where columns become rows, rows become columns, and the original y horizontal vector now becomes a vertical one.

$$W = X^T = \begin{bmatrix} x_1^{(1)} & \dots & x_d^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(n)} & \dots & x_d^{(n)} \end{bmatrix}; \; T = Y^T = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix}.$$

Now, we can get

$$\begin{aligned}(\theta) &= \frac{1}{n} \underbrace{(W\theta - T)^T}_{1\times n} \underbrace{(W\theta - T)}_{n\times 1} \\ &= \frac{1}{n} \sum_{i=1}^{n} \left(\left(\sum_{j=1}^{d} W_{ij}\theta_j \right) - T_i \right)^2 .\end{aligned}$$

After setting to 0 and solving, we get

$$\theta = \underbrace{(W^T W)^{-1}}_{d\times d} \underbrace{W^T}_{d\times n} \underbrace{T}_{n\times 1}$$

where θ is the model parameter. At the end, we have θ equal to a specific number because the dimensions can multiply out since the resulting dimension of $(d \times d)(d \times n)(n \times 1)$ is 1. The process should approximately take $O(dn + n^2)$ as time complexity. Note, it is possible for $W^T W$ to be singular and this should be addressed if it is relevant to the application area.

Given ordinary least squares regression with n data points of dimension d, $OLS(n, d)$, we will analyse its time and space complexity. In the algorithm we perform one matrix inversion of a $(d \times d)$ matrix and three matrices multiplies of sizes: $(d \times n)$ times $(n \times d)$, $(d \times d)$ times $(d \times n)$, $(d \times d)$ times $(n \times 1)$. This gives a total running time of

$$T(OLS(n, d)) = O(dn^2 + d^3 + d^2 n).$$

If we take the common assumption that our number of data points n is larger than the dimension of the data d then this simplifies to

$$T(OLS(n, d)) = O(dn^2).$$

Similarly we can compute the space complexity given multiple calls to rectangular matrix multiply resulting in

$$S(OLS(n, d)) = O(d^2 + dn) = O(dn).$$

Reversible Analysis: As we just saw, linear least squares can be expressed as three matrix multiplications and a matrix inverse. Given our efficient reversible rectangular matrix multiplication and matrix inversion from the previous section, we can use that as a reversible subroutine to obtain a reversible ordinary least squares algorithm with the same asymptotic running time and space as the irreversible version.

If we had used one of the naive transforms, such as Bennett or Lecerf, then the reversible algorithm would have required either $\Theta(d^2 n)$ space or suffered a logarithmic overhead in time. Thus, we can compute the linear regression that minimizes the losses.

B. Regularizing linear regression

Regularization is a technique used to reduce errors by fitting the function appropriately and prevent overfitting. It is also possible for (W^TW) to be not be invertible. Both of these issues can be addressed with ridge regression which adds an L_2-norm regularizer, $\|\theta\|^2$, to the ordinary least squares with a tuning parameter λ.This gives the following objective function:

$$J_{ridge} = (\theta, \theta_0) = \frac{1}{n}\sum_{i=1}^{n}(\theta^T x^{(i)} = \theta_0 - y^{(i)})^2 + \lambda\|\theta\|^2.$$

This minimization also has an analytic solution given by

$$\theta_{ridge} = (W^TW + n\lambda I)^{-1}W^TT,$$

This differs from the prior example by scalar multiplications and a matrix addition which do not change the time or space complexity of the reversible or irreversible algorithms.

IV. Conclusion and Open Problems

There are significant engineering challenges that must be overcome to unlock the potential of general purpose reversible computing. These exist at every layer, from physics and gate design through theoretical algorithms. This paper gives asymptotically efficient reversible algorithms for several fundamental linear algebra and machine learning problems taking away another barrier to our goal. In addition, the study of efficient algorithms may help guide the development of special purpose hardware and heterogeneous architectures which use principals from reversible computing to save energy.

Despite significant progress in the past decade, the field of efficient reversible algorithms is still new and many fundamental algorithms have not been studied in this context. This includes essentially the entire field of machine learning. Clustering algorithms, nearest neighbor algorithms, logistic regression, and deep neural nets all seem like compelling cases. Going further one can make more demanding requests of the efficiency of reversible algorithms. For example, our algorithms are not hygienic as they produce more garbage bits than strictly necessary [17].

Even within the context of linear regression and linear algebra theoretical questions still abound. In this paper we chose simple, textbook algorithms to start building up more techniques for designing efficient reversible algorithms. One could consider fast matrix multiplication algorithms, numerically stable matrix inversion, cache-oblivious algorithms or the many other techniques used to improve performance both in theory and in practice. Cache-oblivious matrix multiplication [26] and Strassen's algorithm [27] do not have a linear size inner loop to use as a reversible subroutine and thus the technique here does not immediately adapt.

Our work here is also purely theoretical. It would be good to see more implementations of reversible algorithms in reversible programming languages like Janus. Further work simulating the resource use of these algorithms or estimating their efficiency on plausible reversible hardware would be useful for understanding the trade-offs of potential reversible computers.

References

[1] IEA, "Data centres and data transmission networks, international energy agency report," 2020.

[2] J. Koomey, S. Berard, M. Sanchez, and H. Wong, "Implications of historical trends in the electrical efficiency of computing," *IEEE Annals of the History of Computing*, vol. 33, no. 3, pp. 46–54, 2010.

[3] M. P. Frank, "Introduction to reversible computing: motivation, progress, and challenges," in *Proceedings of the 2nd Conference on Computing Frontiers*, 2005, pp. 385–390.

[4] A. N. Al-Rabadi, "Reversible logic neural networks," in *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No. 04CH37541)*, vol. 4. IEEE, 2004, pp. 2677–2682.

[5] A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse, "The reversible residual network: Backpropagation without storing activations," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 2211–2221.

[6] D. Maclaurin, D. Duvenaud, and R. Adams, "Gradient-based hyperparameter optimization through reversible learning," in *International conference on machine learning*. PMLR, 2015, pp. 2113–2122.

[7] B. Chang, L. Meng, E. Haber, L. Ruthotto, D. Begert, and E. Holtham, "Reversible architectures for arbitrarily deep residual neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[8] M. MacKay, P. Vicol, J. Ba, and R. Grosse, "Reversible recurrent neural networks," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018, pp. 9043–9054.

[9] S. Spanbauer and L. Sciarappa, "Neural group actions," *arXiv preprint arXiv:2010.03733*, 2020.

[10] Y. Lecerf, "Logique mathematique. machines de Turing réversibles. récursive insolubilité en $n \in N$ de l'équation $u = \theta^n u$, où θ est un "isomorphisme de codes". note," *Comptes rendus hebdomadaires des séances de l'Academie des sciences*, vol. 257, no. 18, pp. 2597–2600, October 1963.

[11] C. H. Bennett, "Logical reversibility of computation," *IBM Journal of Research and Development*, vol. 17, no. 6, pp. 525–532, 1973.

[12] K.-J. Lange, P. McKenzie, and A. Tapp, "Reversible space equals deterministic space," *J. Comput. Syst. Sci.*, vol. 60, pp. 354–367, 2000.

[13] H. Buhrman, M. Li, J. Tromp, and P. Vitanyi, "Kolmogorov random graphs and the incompressibility method," 1997.

[14] S. Xu, "Reversible logic synthesis with minimal usage of ancilla bits," *CoRR*, vol. abs/1506.03777, 2015. [Online]. Available: http://arxiv.org/abs/1506.03777

[15] E. D. Demaine, J. Lynch, G. J. Mirano, and N. Tyagi, "Energy-efficient algorithms," in *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, 2016, pp. 321–332.

[16] M. Li and P. Vitányi, "Reversibility and adiabatic computation: trading time and space for energy," *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 452, pp. 769 – 789, 1996.

[17] H. B. Axelsen and T. Yokoyama, "Programming techniques for reversible comparison sorts," in *Asian Symposium on Programming Languages and Systems*. Springer, 2015, pp. 407–426.

[18] T. Yokoyama, "Reversible computation and reversible programming languages," *Electronic Notes in Theoretical Computer Science*, vol. 253, pp. 71–81, 03 2010.

[19] C. Lutz, "Janus: a time-reversible language," 1986, *Letter to R. Landauer*.

[20] M. P. Frank, "Reversibility for efficient computing," Ph.D. dissertation, Massachusetts Institute of Technology, 1999.

[21] L. Guo, C. Peng, and C. He, "New reversible computing algorithms for shortest paths problem," in *Proceedings of the 6th International Conference on Information Technology: IoT and Smart City*, ser. ICIT 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 82–87. [Online]. Available: https://doi.org/10.1145/3301551.3301605

[22] V. Sarah, "Reversible graph algorithms," Ph.D. dissertation, Master Thesis, University of Copenhagen, 2015.

[23] C. H. Bennett, "Time/space trade-offs for reversible computation," *SIAM Journal on Computing*, vol. 18, no. 4, pp. 766–776, Aug. 1989.

[24] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. The MIT Press, 2001.

[25] L. Kaelbling, "6.036 introduction to machine learning course notes, chapter 7: Regression," MIT Open Learning Library, 2020.

[26] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran, “Cache-oblivious algorithms,” in *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*. IEEE, 1999, pp. 285–297.

[27] V. Strassen, “Gaussian elimination is not optimal,” *Numerische mathematik*, vol. 13, no. 4, pp. 354–356, 1969.