D:Wave
The Quantum Computing Company™

Measuring Performance of the Leap Constrained Quadratic Model Solver: August 2023 Update

TECHNICAL REPORT

2023-08-09

Overview

In this technical report we provide, an overview of the performance of the Constrained Quadratic Model solver available as part of D-Wave's™ hybrid solver service. We define a variety of problems based on widely used problem types and benchmark the most recent update of the Constrained Quadratic Model solver against previous versions.

CONTACT

Corporate Headquarters
3033 Beta Ave
Burnaby, BC V5G 4M9
Canada
Tel. 604-630-1428

US Office
2650 E Bayshore Rd
Palo Alto, CA 94303

Email: info@dwavesys.com

www.dwavesys.com

14-1065B-B
D-Wave Technical Report Series

Notice and Disclaimer

D-Wave Systems Inc. (“D-Wave”), its subsidiaries and affiliates, makes commercially reasonable efforts to ensure that the information in this document is accurate and up to date, but errors may occur. NONE OF D-Wave Systems Inc., its subsidiaries and affiliates, OR ANY OF ITS RESPECTIVE DIRECTORS, EMPLOYEES, AGENTS, OR OTHER REPRESENTATIVES WILL BE LIABLE FOR DAMAGES, CLAIMS, EXPENSES OR OTHER COSTS (INCLUDING WITHOUT LIMITATION LEGAL FEES) ARISING OUT OF OR IN CONNECTION WITH THE USE OF THIS DOCUMENT OR ANY INFORMATION CONTAINED OR REFERRED TO IN IT. THIS IS A COMPREHENSIVE LIMITATION OF LIABILITY THAT APPLIES TO ALL DAMAGES OF ANY KIND, INCLUDING (WITHOUT LIMITATION) COMPENSATORY, DIRECT, INDIRECT, EXEMPLARY, PUNITIVE AND CONSEQUENTIAL DAMAGES, LOSS OF PROGRAMS OR DATA, INCOME OR PROFIT, LOSS OR DAMAGE TO PROPERTY, AND CLAIMS OF THIRD PARTIES.

D Wave reserves the right to alter this document and other referenced documents without notice from time to time and at its sole discretion. D-Wave reserves its intellectual property rights in and to this document and its proprietary technology, including copyright, trademark rights, industrial design rights, and patent rights. D Wave trademarks used herein include D-Wave™, Leap™, Ocean™, Advantage™, Advantage2™, D-Wave 2000Q™, D-Wave 2X™, D-Wave Learn™, D-Wave Launch™, and the D-Wave logos (the D-Wave Marks). Other marks used in this document are the property of their respective owners. D Wave does not grant any license, assignment, or other grant of interest in or to the copyright of this document, the D-Wave Marks, any other marks used in this document, or any other intellectual property rights used or referred to herein, except as D-Wave may expressly provide in a written agreement. This document may refer to other documents, including documents subject to the rights of third parties. Nothing in this document constitutes a grant by D-Wave of any license, assignment, or any other interest in the copyright or other intellectual property rights of such other documents. Any use of such other documents is subject to the rights of D-Wave and/or any applicable third parties in those documents

1 Introduction

D-Wave's hybrid solver service (HSS) contains a portfolio of heuristic solvers that leverage both quantum and classical solution approaches to solve optimization problems much larger than can fit on Advantage™ quantum processors. The quantum processing unit (QPU) natively solves quadratic unconstrained binary optimization problems over the Pegasus™ graph topology [1], while the portfolio of HSS solvers provide interface support for applications well outside that native problem formulation. This interface reduces, and sometimes completely eliminates, the need for users to translate their application problems into a formulation that matches the quantum architecture.

Figure 1 illustrates the result of D-Wave's continuing efforts to expand the variety of problems that fall within scope of the HSS portfolio. The Binary Quadratic Model (BQM) and Discrete Quadratic Model (DQM) solvers read unconstrained quadratic problems defined on binary variables (that is, taking two values), and on discrete variables (that is, taking multiple values), respectively. The Constrained Quadratic Model (CQM) solver adds the capability of specifying linear and quadratic constraints for the quadratic model. Moreover, this solver accepts problems defined on binary, integer and, as of May 2022, real variables.[1] To our knowledge, this is the world's first and only hybrid solver capable of leveraging quantum computation to address both discrete and continuous problems.

In this report, we focus on understanding the performance of the CQM solver on a wide variety of constrained quadratic problems. As D-Wave continues to update the CQM solver, including algorithmic improvements and increasing support for more problem types, the benchmarking framework used in this report can be used to quantify the impact of these changes.

This report presents an overview of performance of the Constrained Quadratic Model solver in the following sections:

- Section 2 surveys the varieties of problem types that serve as industry-standard benchmarks.

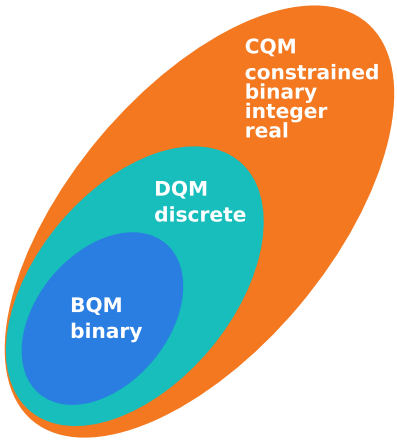

Figure 1: The hierarchy of models available in Ocean. The Binary Quadratic Model and Discrete Quadratic Model are subsets of the more general Constrained Quadratic Model. This figure originally appeared in [2].

- Section 3 presents details about the releases of the CQM solver that are compared in this benchmark.
- Section 4 presents the results of the benchmark.
- Section 5 highlights one problem class that benefited from algorithmic improvements to the CQM solver.

The hybrid solver service is cloud-based and offered by subscription via the Leap™ web portal; see [3, 4] to learn more about Leap and the hybrid solver service.

2 Problem Classes for Understanding Performance

Constrained quadratic models are a large class of models which can contain binary, discrete, integer, and real variables. The most general version of the CQM formulation can include interactions between two variables of any kind. In addition to specifying an objective function to be optimized, the model can include several kinds of constraints which must be satisfied. These constraints themselves take the form of quadratic models. The CQM model supports formulation of equal-

[1] Some notational conflict is unavoidable in standard usage: `binary`, `discrete`, and `integer` variables in computer science are examples of discrete number domains in mathematics, and `real` variables belongs to the continuous number domain.

ity and inequality constraints, which may be linear or quadratic as well as hard (weighted) or soft (weighted). Constraints can be defined on binary, integer, or continuous variables.[2] Samples returned by the CQM solver are called *feasible* if they satisfy the constraints provided; otherwise, they are called *infeasible*.[3]

Understanding performance on all CQMs can be difficult because this problem class is very diverse. Instead, we can break it up into a menagerie of smaller interrelated problem classes that resemble specific applications types. We can use a rough hierarchy of generality to categorize these kinds of problems, as illustrated in Figure 2.

Binary Problems Binary quadratic problems are often used in feature selection, satisfying Boolean expressions, and quantum simulation. Binary quadratic problems are closest to the native model supported by the QPU. In addition, a CQM with binary variables can be used to represent a model with discrete variables. Given a discrete variable x, which takes values in $\{D_1, \ldots, D_n\}$, we can encode the state of x as a set of n Booleans d_i which are equal to 1 if $x = D_i$. This is not sufficient, as x can only take one state at a time, and so we have to add the constraint that $\sum D_i = 1$. Therefore, we also use problems with discrete variables, such as graph coloring, and give them binary-only formulations.

Integer Problems Integer problems, for which variables can be assigned multiple ordered values like $0, 1, 2, 3, \ldots$, are widely used in discrete optimization. Here we use "integer problem" to include problems defined on integer and/or binary variables, but not on real (continuous) variables. Many problems are natively integer, for example, financial problems that involve purchasing whole units. Integer problems are often also useful for problems with discrete units of time or space, such as in scheduling problems where events are scheduled in 15-minute intervals. These problems are naturally integer and not discrete because time and space are ordered, whereas discrete variables encoded as binary are unordered. Often, formulations for problems, like bin packing, can be done using either integer or real variables; however, the choice of which formulation is more appropriate is determined by the specifics of a given application. The integer versions of these problems are created by discretizing the continuous variables, like time or space.

Mixed Integer Problems Mixed integer problems, which can contain a combination of binary, integer, and real variables, are the most general types used in our tests. Models containing real variables are typically found when the values to be assigned to nodes represent variables that are naturally continuous, like locations in space, time, and money (when allowing for subdividing dollars).

Sourcing We source many of these problems from several discrete programming libraries [6–10], which often feature a variety of variable types, degree, and sizes of problems. Some libraries, like MINLPLib [6, 7] have a variety of application-specific problems which have been collected into benchmarks. Other libraries are generated based on a single application type, such as graph coloring [10]. All instances were sourced as `lp` or `mps` files and then converted into a solver-compatible format using the Ocean™ SDK [4].

We have also developed input generators for satisfiability and circuit satisfiability (used for factoring) inputs, which are mainly formulated as binary quadratic inputs. One benefit of writing problem generators is the ability to test performance over a variety of input parameters and formulation strategies, including performance on equivalent formulations. For example we can construct a Boolean satisfiability problem $(x_1 \wedge x_2)$ as an unconstrained CQM $Obj = -x_1 \cdot x_2$ or as a constrained CQM with $Obj = 0$ and the constraint $x_1 \cdot x_2 = 1$. We can do the same procedure for factoring problems expressed as multiplication circuits, since these are a kind of satisfiability problem. We also introduce offer-allocation problems to the benchmark, which are binary quadratic constrained problems discussed further in Section 5.

[2] Not every possible formulation has historically been supported by the solver, with real variables being introduced in May of 2022. Moreover, the solver does not support CQMs where there is a real variable in a quadratic interaction. Similarly, problems involving soft constraints can only be solved in versions of the solver introduced as of November of 2022. Problems with quadratic interactions of real variables became solvable in January of 2023.

[3] Because the CQM solver is a heuristic solver, it may return a mixture of feasible and infeasible samples, none of which are guaranteed to be optimal.

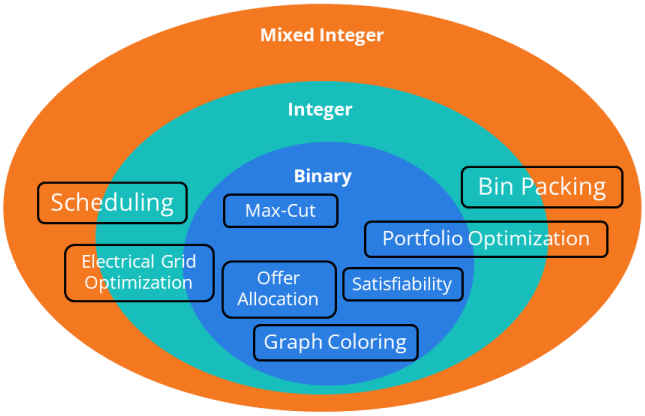

Figure 2: For the various problem classes that were tested, a non-exhaustive set of example application areas as they would be formulated in the CQM solver. Often, problems can be formulated several ways and the details of the formulation are application and performance specific. To see some of these examples in practice visit [5].

To achieve a similar diversity of equivalent formulations for problems drawn from benchmarking libraries, we create equivalent formulations by randomly applying a "flip" to a constraint $x \geq y \Rightarrow -x \leq -y$, or scaling constraints by random positive numbers $x \geq y \Rightarrow a \cdot x \geq a \cdot y, a > 0$. These augmentations to existing libraries help provide an even more diverse set of problems with which to characterize the performance of the solver. The specific mix of problems has changed between this publication and the previous one [11] because of this random element; however the sourcing has remained constant, unless otherwise noted.

3 Updates to the CQM Solver

To characterize the impact of updates to the CQM solver available in Leap, we select key releases since the last publication of this report. [4] Whereas the CQM solver releases in the previous report [11] focused on supporting broader classes of problems, the CQM solver releases in the current report focus on algorithmic improvements to existing classes of problems. We test 6 CQM solver releases: November 2022, December 2022, March 2023, May 2023, June 2023, and July 2023. Because the composition and timing of the benchmark has changed since the previous report, the earliest version of the CQM solver tested in this report is the latest version tested in the previous report [11]. Although this does not provide a fully comparable set of benchmarks across publication dates, the contiguity should provide a rough comparison across publication dates.

4 Performance of the CQM Solver

Every release of the CQM solver is based on the same hybrid quantum-classical workflow, as shown in Figure 3. The solver has a classical front end that reads an input Q and (optionally) a time limit T.[5] The classical front end then invokes one or more hybrid heuristic solvers (computation threads) to search for good-quality solutions to Q.

Each solver contains a classical *heuristic module* that explores the solution space, and a *quantum module (QM)*, which formulates *quantum queries* that are sent to a backend Advantage QPU. Responses from the QPU

[4] To better understand where these updates to the solver exist in the product timeline, see [**ReleaseNotesDWave**].

[5] If no time limit is provided by the user, a default time that depends on input size is used.

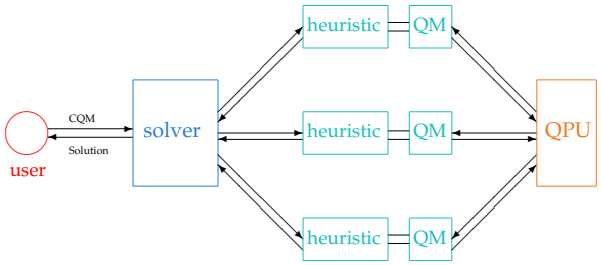

Figure 3: Structure of the CQM solver in HSS. The front end (blue) reads an input Q, and, optionally, a time limit T. The CQM solver invokes some number of heuristic solvers (threads) that run on classical CPUs and GPUs (teal) and search for good-quality solutions to Q. Each heuristic solver contains a quantum module (QM) that formulates and sends quantum queries to a D-Wave QPU (orange); QPU responses to these queries may be used to guide the heuristic search or to improve the quality of a current pool of solutions. This figure originally appeared in [2].

are used to guide the heuristic module toward more promising areas of the search space, or to find improvements to existing solutions. Each heuristic sends its best solutions to the front end before the time limit is reached, and the front end forwards best results to the user.

In a production environment, heuristic solvers run in parallel on state-of-the-art CPU and/or GPU platforms. These tests were carried out using a "laboratory" version of the CQM solver for each update tested. These versions run on less performant classical hardware, but allow us to maintain older updates which are no longer available in production, and to carefully control the hardware for comparisons. In contrast, the HSS production solvers available to the public are deployed for scalable use in the cloud. Since the hybrid framework shown in Figure 3 is heavily dependent on the performance and scale of hardware, the results of this section may differ somewhat from those observed in deployed systems; however, we generally expect the latter to be more efficient.

4.1 Methodology

Each solver was containerized and run with the contemporaneous Ocean software releases where possible, with small fixes to allow older updates of the solver to run. The solvers were all given 150 seconds (2.5 minutes) to solve each problem using the same hardware resources and access to the QPU. Each solver returned an algorithmically determined number of samples, and the objective value and feasibility of each sample was calculated.

In this section, we compare the different solver releases using a win-loss criterion. For each problem, the solver that has the lowest feasible objective value on any sample is given a "win", with ties being counted as a win for both solvers. If none of the solvers have a feasible objective value, meaning all of the samples for that problem across all solvers were infeasible, then the solver with the lowest overall objective value is counted as a win. This approach aligns with "solution-to-time framework" adopted by several publicly available repositories of benchmarks for quadratic solvers [e.g. 12].

4.2 Results

Figure 4 summarizes the results of running this experiment on the different releases of the CQM solver under the parameters outlined above.

Of the 2045 binary quadratic problems, the July 2023 release won 80.0% of problems. The second and third best solver on binary quadratic problems were June 2023 (71.1%) and March 2023 (62.6%) respectively. All of the other 3 solvers had at least one win. Of the 80 integer quadratic problems, the March 2023, May 2023, and July 2023 releases all won 81.2% of problems. All of the other 3 solvers had at least one win. Of the 1110 mixed integer quadratic problems, the July 2023 release won 40.2% of problems. The second and third best solver on mixed integer quadratic problems were June 2023 (30.5%) and May 2023 (27.5%) respectively. All of the other 3 solvers had at least one win.

5 Offer Allocation Problems

The algorithmic improvements presented in Section 4 are most concentrated on constrained binary quadratic models, which are a broad and expressive class of problems with many applications, as shown in Figure 2. The applicability of constrained binary quadratic models to financial services has been a long-standing line of research, most commonly finding use as a technique

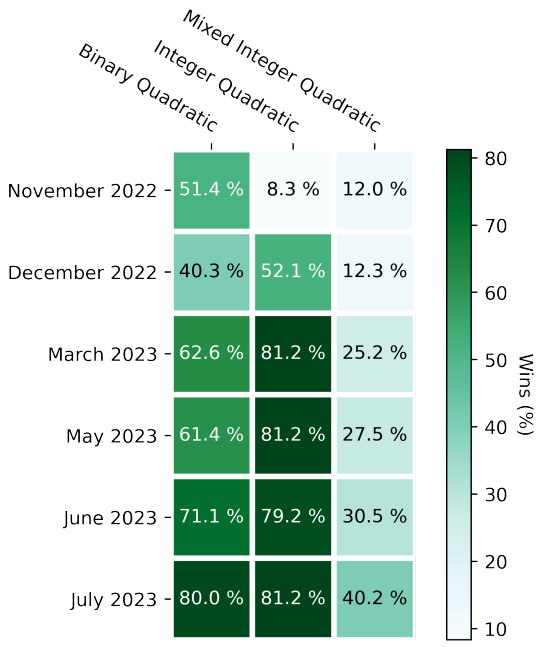

Figure 4: The percentage of "wins" for each update of the solver, allowing for ties to be counted for both solvers. A win is characterized by a lower feasible objective value on the same problem instance given the same hardware and time limit. If no feasible objective value is found, then the win is given to the solver with the lowest overall objective value.

for selecting optimal subsets of securities, predictors, or products when individual and pairwise information is known [e.g. 13, 14]. Optimal-selection problems are ever more ubiquitous, following the continuing rise of machine-learning based tools in finance, which solve individual and pairwise information prediction problems [15, 16].

One powerful example of these trends is the problem of individualized pricing for consumers, so-called price discrimination [17]. Price discrimination is often achieved by having one stated public price but offering individual consumers discounts or coupons which lower the individual consumer's price. Finding the optimal mix of discounts or coupons in order to maximize profitability is an optimal-selection problem, naturally formulated as a constrained quadratic binary model.

Specifically, offer allocation is the problem of assigning m offers (e.g. discounts, new products, rewards programs) to n consumers. Each of the m offers can only be given to a small number of consumers c_j, and each consumer can only receive a handful of offers c_i. To formulate this problem as a CQM, we assign a binary variable $a_{i,j}$ to each offer-consumer pair. We can then formulate the constraints on the number of consumers given an offer as

$$\sum_{i}^{n} a_{i,j} \leq c_j$$

and similarly, the constraints on the number of offers given to a consumer

$$\sum_{j}^{m} a_{i,j} \leq c_i \, .$$

We can see how to generate these sets of constraints in the function defined in Example 1 given an array of the total number of offers per consumer c_i (argument `c_consumers`) and total offers c_j (`c_offers`). One of the algorithmic improvements to the CQM solver since the previous report [11] is the handling of these constraint types more efficiently, which can explain part of the improvement in performance on these problems.

So far we have generated the constraints needed to create an offer-allocation problem, but we have not considered optimizing the offers. In order to do that we must first know the average value of a consumer when offered a given offer; however, more interestingly we can optimize based on the average value of a consumer given any pair of offerings. This is especially useful if

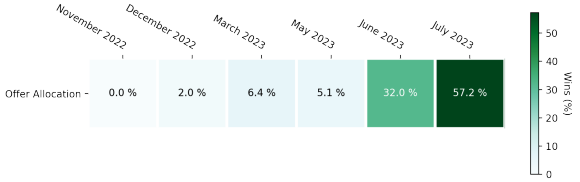

Figure 5: The percentage of "wins" for each update of the solver on offer-allocation problems tested, allowing for ties to be counted for both solvers. The 297 problem instances presented here are a subset of those in 4, specifically every offer-allocation problem is a binary constrained problem.

we think that offers influence each other. For a consumer i, the average value $v_{i,j,k}$ of a deal mix including both offer j and k can be used to create an objective which optimizes the offer allocation and is given by

$$\sum_{i}^{n}\sum_{j}^{m}\sum_{k>j}^{m} -v_{i,j,k} * a_{i,j} * a_{i,k} \, .$$

In Example 1, we pass in a dictionary with the values of $v_{i,j,k}$ (argument `values`). These values would usually be generated using statistics of previous offer mixes; for example, using machine learning models of customer values.

We generate 297 problems instances of this kind with between 1,000 and 10,000 consumers, and include the problems in the results discussed in Section 4, specifically in the binary quadratic category. Figure 5 shows the results on this class of problems across different solver releases. There is a marked uptick in performance over the past two releases, from June 2023 and July 2023, with the July 2023 solver winning 57.2% of problem instances.

```
import dimod


def offer_allocation(
 m: int,
 n: int,
 c_consumers: list,
 c_offers: list,
 values: dict):
 cqm = dimod.ConstrainedQuadraticModel()

 assignments = {}

 for i in range(n):
  for j in range(m):
   # each offer-consumer pair is a binary
    variable
   pair = dimod.Binary(
       f"consumer_{i}_offer_{j}"
     )

   assignments[i, j] = pair

 # each consumer has limited number of offers
 for i in range(n):
  cqm.add_constraint_from_comparison(
   dimod.quicksum(
    assignments[i, j] for j in range(m)
   ) <= c_consumers[i]
  )

 # each offer can only be given a limited
    number of times
 for j in range(m):
  cqm.add_constraint_from_comparison(
   dimod.quicksum(
    assignments[i, j] for i in range(n)
   ) <= c_offers[j]
  )

 # maximize the total value of deal mix per
    customer offered deals
 cqm.set_objective(
  dimod.quicksum(
   -val*assignments[i, j]*assignments[i, k]
    for (i, j, k), val in values.items()
    if j > k
  ),
 )

 return cqm
```

Example 1: Creating consumer and offer variables and constraints using Ocean.

6 Conclusion

To understand the performance of algorithms at the rapidly advancing frontier of hybrid optimization, it is necessary to benchmark on a diverse set of problems. This report puts forward a taxonomy of problems which represent various real-world and theoretical benchmarks for D-Wave's CQM solver. Using this framework, we characterize the overall performance of the CQM solver's most recent release versus that of several previous ones. By comparing the performance of several recent releases of the CQM solver against each other, we are able to see the rapid improvement in performance of each new release compared to the earlier ones.

References

[1] C. McGeoch and P. Farré, "Advantage Processor Overview," D-Wave Technical Report Series (2022).

[2] "Hybrid Solvers for Quadratic Optimization," (2022).

[3] *D-Wave Leap*, https://cloud.dwavesys.com/leap.

[4] *D-Wave Ocean Software Documentation*, https://docs.ocean.dwavesys.com/.

[5] *D-Wave Systems Examples*, GitHub, https://github.com/dwave-examples.

[6] M. R. Bussieck, A. S. Drud, and A. Meeraus, "MINLPLib—A Collection of Test Models for Mixed-Integer Nonlinear Programming," INFORMS Journal on Computing **15**, 114–119 (2003).

[7] S. Vigerske, *MINLPLib: A Library of Mixed-Integer and Continuous Nonlinear Programming Instances*, (Oct. 14, 2022) https://www.minlplib.org/ (visited on 10/22/2022).

[8] A. Gleixner, G. Hendel, G. Gamrath, T. Achterberg, M. Bastubbe, et al., "MIPLIB 2017: Data-driven compilation of the 6th mixed-integer programming library," Mathematical Programming Computation, 10.1007/s12532-020-00194-3 (2021).

[9] F. Furini, E. Traversi, P. Belotti, A. Frangioni, A. Gleixner, et al., "QPLIB: a library of quadratic programming instances," Math. Prog. Comp. **11**, 237–265 (2019).

[10] S. Gualandi and M. Chiarandini, *Vertex Coloring - Graph Coloring Benchmarks*, https://sites.google.com/site/graphcoloring/vertex-coloring (visited on 10/22/2022).

[11] J. S. Lozano, P. Farré, and C. McGeoch, "Measuring Performance of the Leap Constrained Quadratic Model Solver," (2022).

[12] H. D. Mittelmann, *Decison Tree for Optimization Software*, http://plato.asu.edu/guide.html (visited on 10/22/2022).

[13] D. J. Laughhunn, "Quadratic Binary Programming with Application to Capital-Budgeting Problems," Operations Research **18**, 454–461 (1970).

[14] A. Milne, M. Rounds, and P. Goddard, "Optimal Feature Selection in Credit Scoring and Classification Using a Quantum Annealer," 1QBit whitepaper.

[15] J. W. Goodell, S. Kumar, W. M. Lim, and D. Pattnaik, "Artificial intelligence and machine learning in finance: Identifying foundations, themes, and research clusters from bibliometric analysis," Journal of Behavioral and Experimental Finance **32**, 100577 (2021).

[16] M. Obthong, N. Tantisantiwong, W. Jeamwatthanachai, and G. Wills, "A survey on machine learning for stock price prediction: algorithms and techniques," in 2nd International Conference on Finance, Economics, Management and IT Business (May 6, 2020), pp. 63–71.

[17] B. Shiller, *First Degree Price Discrimination Using Big Data*, Working Paper 58 (Brandeis University, Department of Economics and International Businesss School, Jan. 2014).