CSCML 2022

2022 INTERNATIONAL SYMPOSIUM ON CYBER SECURITY CRYPTOGRAPHY AND MACHINE LEARNING

JUNE 30 - JULY 1, 2022
VIRTUAL CONFERENCE

PROFESSOR SHLOMI DOLEV
GENERAL CHAIR

CSCML2022

SPONSORS

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

BGU NHSA

IBM

Springer

Journal of Computer and System Sciences

Lecture Notes in Computer Science
LNCS LNAI LNBI

CBG
Cyber@Ben-Gurion University of the Negev

cryptography

Science & Technology
www.science.co.il

In Cooperation With:

International Association for Cryptologic Research

6th International Symposium on Cyber Security Cryptography and Machine Learning (CSCML 2022)

TECHNICAL REPORT

Editors

Shlomi Dolev and Oded Margalit and Yonah Alexandre Bronstein

Technical Report #22-01

June 30, 2022

The BGU-Negev Hi-Tech Faculty Startup

Accelerator, Department of Computer Science,

Ben-Gurion University, Be’er Sheva, Israel

Table of Contents

Ph.D./ Masters Student
Research Track
Chaired by: Oded Margalit

Ph.D./ Masters Student Research Track chaired by Prof. Oded Margalit

The Ph.D./ Masters Track session is a great opportunity to have a bird's-eye view of the research on CSCML topics. This year we had diverse topics: from new insight on classic problems like the shortest path problem, via interesting research on automata learning, to novel usage of quantum entanglement for distributed coordination.

I hope we'll hold next year's conference face-2-face, but in any case, I strongly recommend all researchers in Cyber-Security, Machine Learning and Cryptology to submit their work and participate in the Ph.D./Masters track next year. It is a great way to learn about the state of the art research and receive helpful feedback on your own work.

Looking forward to CSCML 2023.

Regards,

Prof. Oded Margalit,
CSCML 2022 Ph.D./Masters Track Chair
Computer Science department, BGU
and Cyber Security Innovation Center, Citi

Distributed Coordination Based on Quantum Entanglement

Yotam Ashkenazi[1] and Shlomi Dolev[1]

Department of Computer Science, Ben-Gurion University of the Negev, Israel

Abstract. This paper demonstrates and proves that the coordination of actions in a distributed swarm can be enhanced by using quantum entanglement. In particular, we focus on

- Global and local simultaneous random walks, using entangled qubits that collapse into the same (or opposite) direction, either random direction (when qubits are in superposition) or totally controlled simultaneous movements (when the entangled qubits are in pure states).
- Identifying eavesdropping of malicious eavesdroppers aimed to disturb the coordinated random walks using entangled qubits that were sent on random or predefined bases.
- Identifying Byzantine robots using entangled qubits.
- The use of Pseudo Telepathy to coordinate robots' actions.

Keywords: Mobile robots · Byzantine faults · Self-stabilization · Quantum entanglement.

1 Introduction

This paper presents methods to achieve distributed coordination in a swarm of robots using quantum entanglement. We demonstrate a new benefit of quantum mechanics (using the entanglement capabilities) in the scope of distributed secure computing. Many applications use quantum entanglement to enhance the classical algorithm capabilities. In order to achieve coordination between the robots, we use similar methods used in quantum key distribution and pseudo telepathy.

Quantum key distribution algorithms based on distributing entanglement photons were developed decades ago [1]. However, a practical experience of distributing the key to two far away parties was done a few years ago, when scientists were able to distribute entanglement photons over 1200 kilometers using satellites [2] to produce a symmetric key in two remote sites.

Quantum pseudo telepathy methods demonstrated in [3], achieved better results in several games compared to ways that do not have access to the entangled quantum system. In addition to the above quantum techniques, we also focus on randomization, as it is a significant source in computing, particularly in distributed computing. In this paper, we employ entangled qubits to gain random coordinated actions and/or to break the symmetry.

Randomized algorithms are used to compute a task that might have a better performance or efficiency than a deterministic non-randomized algorithm [4] and [5]. In the scope of distributed computing, randomized algorithms overcome impossible results, such as [6], coping with situations where symmetry can not be otherwise broken.

2 Simultaneous Random Walks

Definition 1. (Coordinated random walk) *A path of random steps defines a random walk. Each participant has a random path that is not affected by other participants.*

The problem. Develop an algorithm where the participants can walk in a random fashion, but coordinate with other participants, regardless of their location. The idea is to share a random sequence between the participants, and then the participants can move in a coordinated random fashion.

The classic solution. There are several ways to share a random sequence. One of them, and the obvious one, is based on physical meetings (just as done during key parties). The parties later use the physically exchanged (and agreed on common random sequence) when executing the algorithm.

This scenario has significant drawbacks. A robot needs to predict upfront the robots that it will need to communicate with and establish a shared key in a pre-processing stage (assuming that a public key system with a certificate authority is too expensive to implement). Another drawback is the possibility of using the knowledge on the sequence and the risk of its leakage prior to the actual use of the sequence.

Every time we would like to use the random algorithm, the parties would need a new random sequence as an input to the algorithm, which implies the need for another coordination rendezvous. Our solution would like to have a random sequence with an infinite size over time whenever there is a need.

A standard method to receive a random share sequence is to use random noise from the environment similar to "True random number generation based on environmental noise measurements for military applications" [7]. By using this method, an (almost) truly random sequence can be achieved from the environment. Several entities may receive and analyze a common random noise (e.g., from space). However, in this scenario, an eavesdropper/ Byzantine-robots/ attacker can discover/copy the procedure for harvesting the common noise and how the other robots will act. In our solution, we can, for instance, identify when a Byzantine or an attacker is eavesdropping and act accordingly.

The quantum solution. In the sequel, we propose and detail a new method to achieve distributed coordination between a swarm of robots. This can be based on one robot producing an entangled state and sending part of the state to another robot. Another option is based on a global entity (satellite, for example) sending entanglement photons to several robots.

Our solution suggests three ways of using quantum capabilities in the case of two robots to obtain a stream of an infinite number of random (qu)bits, while ensuring that no entity can clone or manipulate transmitted bits on their way.

- The first option is to use predetermined bases. Using this method, the robots (and the satellite, when used) decide on predetermined bases for each measurement and measure accordingly.
- The second method uses random bases, just as done in Quantum Key Distribution (QKD). Each robot chooses a random base for each measurement. The robots then send/ broadcast their information in randomly chosen directions over another secure channel, where attackers can listen to the communication, but can not modify it.
- The third method uses quantum telepathy, based on the Mermin–Peres magic square game [8]. The idea is to use the game results and to identify a wave interference.

When using the method of distributing entangled particles from a satellite, each robot receives a part of the entangled particle infinitely often.

We consider two cases of random walks. In the first one, we would like to achieve a **global coordinated random walk**, where the robots are located very far from each other. In this scenario, the robots may not be able to sense a common random noise from the environment and can not observe the movements of each other. Note that it is possible that the robots were close by each other in the past, but later moved apart.

In the second scenario, we would like to achieve a **local coordinated random walk** to prevent a collision of two robots executing a random walk. Consider the simple procedure in which a robot performs a simple random walk algorithm. The robot chooses its next move randomly (in our example, up, down, right or left) with the same probability. In one of the scenarios we consider that there are two robots, r_1, and r_2, which are located very close to each other. There is a chance that r_1 randomly chooses to move toward r_2 and, at the same time, r_2 moves toward r_1. In this scenario, they may crash into each other.

We can address both of the cases by the use of entangled qubits. The robots measure the entanglement state and act simultaneously (possibly) even if they are very far from each other. Albert Einstein called, the (possible) faster than the speed of light coordination, "Spooky action at a distance." The robots can move in four directions. Each robot needs two particles to decide on the next move, meaning two entangled states $|\Phi 1\rangle$ and $|\Phi 2\rangle$ with a total of four particles for each step.

The robots r_1 and r_2 measure the states, and each robot interpenetrates the measured values to a command to be executed, for example, $|00\rangle$ up, $|11\rangle$ down, $|01\rangle$ right, and $|10\rangle$ left, where the $|x, y\rangle$ represents the value measured from the first and second particle. r_1 receives the first particle from $|\Phi 1\rangle$ and the first particle from $|\Phi 2\rangle$, and r_2 receives the second particle from $|\Phi 1\rangle$ and the second particle from $|\Phi 2\rangle$.

The entangled particles are Einstein–Podolsky–Rosen (EPR) pairs [9], so without loss of generality, we can assume the state is $|\Phi+\rangle$ and that the robots

measure on a normal basis. The robots measure their particles and move accordingly to the change as before. Using this simple algorithm, assuming r_1 observes $|01\rangle$, r_2 observes with a high probability the same result and the robots move left, the robots can execute the algorithm above. Therefore, they continue to move together in a random fashion and do not collide, if the distance between them is below the threshold.

When using the algorithm above, the robots move together forever. In Section 3, we demonstrate how the centralized entity can control the robot's movements using quantum entanglement. Additionally, we consider the case where the robots can move together in a random fashion. However, a Byzantine robot or an attacker can eavesdrop on the states and predict the robots' movements. Section 5 presents a method for preventing the eavesdropping attack.

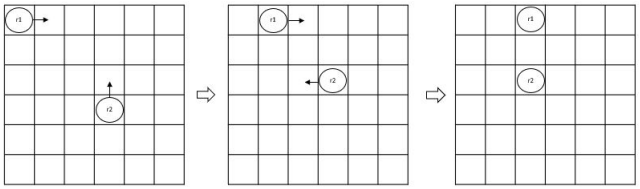


Fig. 1. Robots move randomly until the distance between them is 1

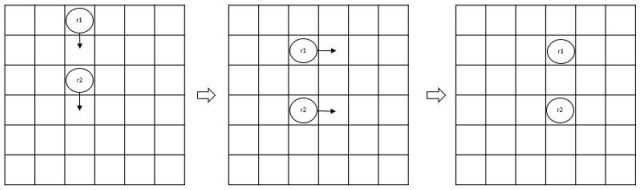


Fig. 2. When the distance between the robots is 1, the robots measure their particles from $|\Phi 1\rangle$ and $|\Phi 2\rangle$ and get $|11\rangle$, and they both move down. In the second step, robots measure $|01\rangle$, and they both move right

Previously, we used pairs of EPR entangled particles in the case of two robots. In the case of three or more robots, we can expand the $|\Phi\rangle$ state to consist of more than two qubits. For example, for three robots, r_1, r_2, and r_3, we can use the state $\frac{1}{\sqrt{2}}(\ |000\rangle + |111\rangle)$, and the robots need two states to create the mapping between the results and the directions. To clarify this point, let us assume that we have two states. The first state is $|xyz\rangle + |xyz\rangle$, and the second state is $|abc\rangle + |abc\rangle$. r_1 measures x and a, r_2 measures y and b, and r_3 measures

z and c. Using this method, the number of robots can be increased depending on the source of the entangled qubits, e.g., satellite, to expand the state.

3 Controlling the Robot's Movements

The previous sections considered the case in which robots move together and execute simultaneous random walks. This section presents how a centralized entity can control the robots' movements.

The problem. In some cases, we would like a centralized entity (a satellite, for example) to control the robots' movements in a deterministic fashion, say one wants to direct a swarm of drones in a specific direction.

The classic solution. Controlling the robots' movements can be achieved using the same classical algorithm as in the previous section. Instead of sending a random sequence, the satellite can send specific bits which map the exact path of the robots.

The quantum solution. Controlling the robots' movements can be achieved using the same simultaneous random walks quantum algorithm. However, instead of sending a random EPR state, the satellite can send an entangled state with the form of $|00\rangle$ or $|11\rangle$. In this case, using the same conditions as above, the centralized entity can decide on the complete path of the robots. The robots continue to execute the algorithm and can not identify their movements as being predefined by the centralized entity. In addition, the centralized entity can control the robot's movement by using a different state, so each robot moves in a different direction. The centralized entity can prevent the situation in which robots stay close forever, while the robots do not move in a random fashion.

4 Avoid Robots Colliding in a Random Fashion

The problem. The previous section considered the case to avoid collision in a deterministic way. In this section, the robots avoid colliding and still move in a random fashion.

The classic solution. It is not trivial to solve the problem using a classical algorithm. The centralized entity can use one of the methods to share random sequence as demonstrated in Section 2. However, if the centralized entity sends the same sequence to the robots, the robots keep moving together forever. The centralized entity can send a different sequence to each of the robots. The centralized entity gets the two random bits from the sequence to r_1, calculates all other options for two bits to r_2 and chooses one option randomly. r_1 and r_2 receive the bits and act accordingly.

The quantum solution. The centralized entity creates a random state with fewer options, so the robots continue to move in a random fashion, without the probability of colliding. This can be done by sending two different EPR states where the robots move in a random direction, but not in the same direction. For example, if two robots are located at a distance one from each other, then the

centralized entity can send the state $\frac{1}{\sqrt{2}}(\ |00\rangle + |11\rangle)$ for the first qubit and $\frac{1}{\sqrt{2}}(\ |01\rangle + |10\rangle)$ for the second qubit. In this case, the options for robot r_1 and robot r_2 are:

- r_1 and r_2 measure the first qubit from the first state $|\Phi 1\rangle$ and the first qubit from the second state $|\Phi 2\rangle$, such that r_1 observes 00 and r_2 observes 01. r_1 moves up, and r_2 moves right.
- r_1 and r_2 measure the first qubit from the first state $|\Phi 1\rangle$ and the second qubit from the second state $|\Phi 2\rangle$, such that r_1 observes 01 and r_2 observes 00. r_1 moves right, and r_2 moves up.
- r_1 and r_2 measure the second qubit from the first state $|\Phi 1\rangle$ and the first qubit from the second state $|\Phi 2\rangle$, such that r_1 observes 10 and r_2 observes 11. r_1 moves left, and r_2 moves down.
- r_1 and r_2 measure the second qubit from the first state $|\Phi 1\rangle$ and the second qubit from the second state $|\Phi 2\rangle$, such that r_1 observes 11 and r_2 observes 10. r_1 moves down, and r_2 moves left.

Another option is that the centralized entity can send the state $\frac{1}{\sqrt{2}}(\ |01\rangle + |10\rangle)$ for the first qubit and $\frac{1}{\sqrt{2}}(\ |00\rangle + |11\rangle)$ for the second qubit. In this case, the options for robot r_1 and robot r_2 are:

- r_1 and r_2 measure the first qubit from the first state $|\Phi 1\rangle$ and the first qubit from the second state $|\Phi 2\rangle$, such that r_1 observes 00 and r_2 observes 10. r_1 moves up, and r_2 moves left.
- r_1 and r_2 measure the first qubit from the first state $|\Phi 1\rangle$ and the second qubit from the second state $|\Phi 2\rangle$, such that r_1 observes 01 and r_2 observes 11. r_1 moves right, and r_2 moves down.
- r_1 and r_2 measure the second qubit from the first state $|\Phi 1\rangle$ and the first qubit from the second state $|\Phi 2\rangle$, such that r_1 observes 10 and r_2 observes 00. r_1 moves left, and r_2 moves up.
- r_1 and r_2 measure the second qubit from the first state $|\Phi 1\rangle$ and the second qubit from the second state $|\Phi 2\rangle$, such that r_1 observes 11 and r_2 observes 01. r_1 moves down, and r_2 moves right.

The centralized entity can send the states $\frac{1}{\sqrt{2}}(\ |00\rangle + |11\rangle)$ for the first qubit and $\frac{1}{\sqrt{2}}(\ |01\rangle + |10\rangle)$ for the second qubit to the robots. In this case, the distance between the robots can increase or remain identical with a positive probability, see Figure 3

5 Eavesdropping Prevention

The problem. In the previous sections, we presented a method of distributed coordination. An eavesdropper can easily attack this method by measuring the random sequence before/together with the robots. In our case, we have four identities; the centralized entity c sending the random sequence, r_1 and r_2

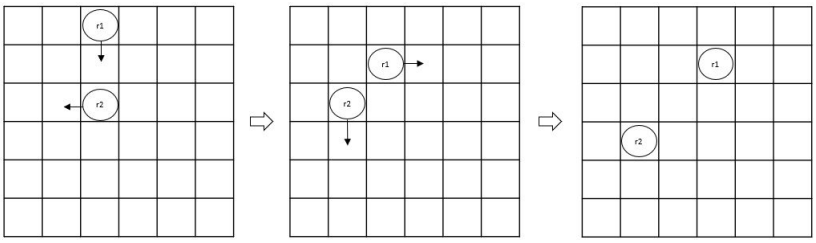


Fig. 3. In the first step, robots measure the fourth option from case 1 (r_1 observes 11 and moves down. r_2 observes 10 and moves left). In the second step, the robot measures the second option from the second case (r_1 observes 01 and moves right, and r_2 observes 11 and moves down)

receiving the sequence, and an attacker *eve* trying to gain information on the random sequence or the next robot's movements.

The classic solution. In case the entities share a secret or have a Public Key Infrastructure (PKI), the obvious and most straightforward method to avoid eavesdropping is to use encryption. Consider the case where all the information is encrypted and *eve* does not have the secret, no eavesdropping can be done.

The quantum solution. We extend the quantum algorithm to be resilient to eavesdropping attacks by sending the photons in one of several states. We obtain very high security using our solutions, the same as the secure method for quantum key distribution, e.g., [10].

The parties can use predefined bases, so each of the c, r_1, and r_2 have the same sequence of bases and each state can be measured on the z basis or x basis.

c creates the state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ for the x basis and $\frac{1}{\sqrt{2}}(|{+}{+}\rangle + |{-}{-}\rangle)$ for the z basis, and sends the entangled state to r_1 and r_2. r_1 and r_2 measure (separately) their photon on a predefined basis. In this case, c, r_1, and r_2 measure the state on the same basis, and this is a valid measurement.

Another case is to use randomized bases. c chooses a random base z basis or x basis and creates the state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ or $\frac{1}{\sqrt{2}}(|{+}{+}\rangle + |{-}{-}\rangle)$ respectively. For each received photon, each robot r_1 and r_2 choose a random basis z basis or x basis and measure the photon state using the selected basis. After several measurements, c, r_1, and r_2 publish their selected bases in an authenticated secure channel. A valid measurement is when c, r_1 and r_2 choose the same basis for each measurement. If the basis is selected in a random fashion, the probability of the same basis is $\frac{1}{4}$. When *eve* is not active, c, r_1 and r_2 keep only the valid measurements, and c, r_1 and r_2 have the same values. At this point, the parties c, r_1, and r_2 use an authenticated secure channel where *eve* has access to the data in the channel, but can not change it. This algorithm still required a shared secret key or PKI same as the classical algorithm. However, it does not require encryption to create the authenticated secure channel.

In the case of *eve* being active, we would like to prevent *eve* from eavesdropping on the states. Without loss of generality, for all the valid measurements, consider the case where *eve* measures the state before r_1 (or r_2) and returns the state after the measurement to r_1. If *eve* measures the state with the same basis as r_1, *eve* and r_1 (and c, and r_2) measure an identical value. If *eve* measures the state with another basis and then sends the state to r_1, r_1 might measure a different result from r_2. In order to identify *eve*, r_1 and r_2 can publish several valid measurement results. If the measurement results are not identical (more than an error rate), c, r_1 and r_2 can assume *eve* eavesdropped on several states, and the measurements are invalid. Using this method, the honest participants can identify if an eavesdropping attack was executed with a high probability.

6 Identify Byzantine Robots Using Entangled Qubits

The problem. In this section, we present a method of identifying Byzantine robots based on entangled qubits, where the non-Byzantine robots agreed on predefined bases.

The quantum solution. Consider the case of simultaneous random walks, where the centralized entity and all the r_i non-Byzantine robots agreed on predefined bases. This scenario can be done using a physical meeting of all the parties. However, it can be done by physical meetings of two identities at a time, meaning the centralized entity can meet r_1 and agree on the bases. Then, r_1 can meet r_2 and transfer the information about the bases until all r_i have the same bases. In this scenario, the centralized entity does not know which of the robots have the predefined bases and which of the robots do not have it. The robots that do not have the bases consider Byzantine robots b_j and have no knowledge of the bases.

For each step, the centralized entity creates two entangle states with $i + j$ qubits each, so each robot (Byzantine and non-Byzantine) receives two qubits (in order to move). During every step, all the non-Byzantine robots measure their qubits and act accordingly. The Non-Byzantine robots move in the same direction, as they all measure on the same base and receive the same results.

The Byzantine robots have several methods to decide on their next move. The first method is to guess the base and measure the qubits. The chance to move to the correct location using the first method is 50%, as the predefined bases were chosen in a random fashion from two options. Another method is to decide on a random direction and move accordingly. When using the second method, each Byzantine robot has a 25% chance to move with the honest robots. Using the first two methods, the probability that a Byzantine robot guesses all the correct movements for a long time is negligible. The third method is to wait until the non-Byzantine robots start to move and follow them. In the third method, it is easy to identify the Byzantine. The non-Byzantine robots can synchronize the time of their movements and, by that, can determine which of the robots delay and recognize them as Byzantine.

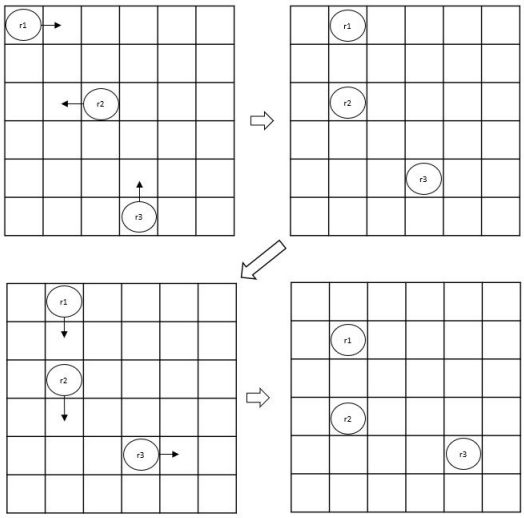


Fig. 4. r_1, r_2 and r_3 move randomly. When the distance between robots r_1 and r_2 is 1, they use the shared valid measurement and move down. r_3 continues to move randomly.

7 Coordinated Random Walk with More Than Two Robots

The problem. In Section 2, we presented an algorithm to achieve coordination between the robots. This section presents a method to achieve coordination in a multi (more than two) robots swarm.

The classic solution. In case all the robots receive the same random sequence, the same algorithm presented in Section 2 can be executed here.

The quantum solution. If we use random bases with multi (more than two) robots, the solution is more sophisticated. The obvious and trivial methods can work, but if the number of robots increases, the probability that all the robots measure the same state decrease dramatically. For example, the probability that all c, r_1, $\ldots r_n$ choose the same basis from the two options, z basis or x basis, is $(\frac{1}{2})^n$. This method is inefficient and can cause many "invalid" measurements.

In order to improve the method above, each robot can execute the same algorithm as in Section 5. However, instead of ignoring all the measurements where the basis is not the same for all the robots, each robot stores the results where the measurement is equal between a subset of the robots and c. For example, if we have three robots r_1, r_2 and r_3. Assume c, r_1 and r_2 measure on the same basis, while r_3 measures on a different basis. r_1 stores the result of this measurement for only an equal result with r_2 (and r_2 stores the result of this measurement for only r_1). In case some operations involve r_1 and r_2 only, they can still use the measurement results.

8 Quantum Pseudo Telepathy Among Swarming Robots

The problem. In this section, we present a method to use quantum telepathy to achieve coordination between the robots using the same idea of Mermin–Peres magic square game described in [8] and [11]. The game includes a 3×3 board and each tile consisting of 1 or -1. The first player returns the line values where the multiple of each tile in the row is 1. The second player returns the values of a column where the multiple of the tiles in the column is -1. The players can share information before the game begins but can not share any information later.

The centralized entity sends a line number to the first player and a column number to the second player. The players win if the number in the tile shared by their row and column is the same.

The quantum solution. It is easy to prove that if the players do not know the line and column numbers in advance, the best win probability without using a quantum entanglement is 8/9. However, if the two players share two quantum entanglement states, they can win with a probability of 1.

The robots can decide on predefined bases for each of the tiles on the board and measure the states using the relevant bases. In our scenario, the two robots can achieve coordination in case the centralized entity publishes the information about which row and column numbers were selected. The two robots have the same result in the shared tile.

Another option to consider is if the robots send their results (one robot sends the row result and the second sends the column result) to a board with 9 sensors arranged in a 3×3 structure. The sensors receive the results, and if a wave interference occurs, the sensor executes an action. As the results in the shared tile are equal, only one relevant sensor (the sensor in the chosen line and row) identifies the wave interference.

Although this algorithm needs two quantum entanglement states, which is less efficient than the previous method, the players have additional information they can use later in this method. In addition, the first robot knows that the multiple of each tile in the chosen row is 1, and the second robot knows that the multiple of each tile in the chosen row is -1.

References

1. title = Quantum cryptography: Public key distribution and coin tossing, author = Bennett, C. H. and Brassard, G., year = 1984, biburl = https://www.bibsonomy.org/bibtex/2ca89602a28a4416dfc6a74ffae7e3292/bronckobuster, booktitle = Proceedings of IEEE International Conference on Computers, Systems, and Signal Processing, interhash = 2ec1f042f41d6db18ff61cfba05e970d, intrahash = ca89602a28a4416dfc6a74ffae7e3292, keywords = imported, location = Bangalore, pages = 175, timestamp = 2009-03-03T17:20:15.000+0100,
2. Author = Yin, J. et al., title = Satellite-based entanglement distribution over 1200 kilometers, journal = Science, volume = 356, number = 11, pages = 1140–1144, year = 2017,

3. author = "Brassard, Gilles and Cleve, Richard and Tapp, Alain", title = "The Cost of exactly simulating quantum entanglement with classical communication", eprint = "quant-ph/9901035", archivePrefix = "arXiv", doi = "10.1103/PhysRevLett.83.1874", journal = "Phys. Rev. Lett.", volume = "83", pages = "1874–1877", year = "1999"
4. author = Sankalp Arora and Sebastian A. Scherer, title = Randomized algorithm for informative path planning with budget constraints, booktitle = 2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017, pages = 4997–5004, publisher = IEEE, year = 2017, url = https://doi.org/10.1109/ICRA.2017.7989582, doi = 10.1109/ICRA.2017.7989582, timestamp = Fri, 22 Nov 2019 16:53:25 +0100, biburl = https://dblp.org/rec/conf/icra/AroraS17.bib, bibsource = dblp computer science bibliography, https://dblp.org
5. title=Some complexity questions related to distributive computing(Preliminary Report), author=Andrew Chi-Chih Yao, journal=Proceedings of the eleventh annual ACM symposium on Theory of computing, year=1979
6. author=Turek, J. and Shasha, D., journal=Computer, title=The many faces of consensus in distributed systems, year=1992, volume=25, number=6, pages=8-17, doi=10.1109/2.153253
7. author = BARDIS, N. G.—MARKOVSKYI, A. P.—DOUKAS, N.—KARADIMAS, N. V., title = True Random Number Generation based on Environmental Noise Measurements for Military Applications, journal = Proc. of the 8th WSEAS International Conference on Signal Processing, Robotics and Automation (ISPRA 09), pages = 68–73, year = 2009,
8. title = Simple unified form for the major no-hidden-variables theorems, author = Mermin, N. David, journal = Phys. Rev. Lett., volume = 65, issue = 27, pages = 3373–3376, numpages = 0, year = 1990, month = Dec, publisher = American Physical Society, doi = 10.1103/PhysRevLett.65.3373, url = https://link.aps.org/doi/10.1103/PhysRevLett.65.3373
9. , title = On the Einstein Podolsky Rosen paradox, author = Bell, J. S., journal = Physics Physique Fizika, volume = 1, issue = 3, pages = 195–200, numpages = 6, year = 1964, month = Nov, publisher = American Physical Society, doi = 10.1103/PhysicsPhysiqueFizika.1.195, url = https://link.aps.org/doi/10.1103/PhysicsPhysiqueFizika.1.195
10. author = Yin, J. et al., title = Entanglement-based secure quantum cryptography over 1,120 kilometres, journal = Nature, volume = 582, number = 7813, pages = 501-505, year = 2020,
11. title = Incompatible results of quantum measurements, journal = Physics Letters A, volume = 151, number = 3, pages = 107-108, year = 1990, issn = 0375-9601, doi = https://doi.org/10.1016/0375-9601(90)90172-K, url = https://www.sciencedirect.com/science/article/pii/037596019090172K, author = Asher Peres, abstract = Quantum theory is incompatible with the following propositions. (1) The result of the measurement of an operator A depends solely on A and on the system being measured. (2) If operators A and B commute, the result of a measurement of their product AB is the product of the results of separate measurements of A and of B.
12. author = Peter W. Shor, editor = Leonard M. Adleman and Ming-Deh A. Huang, title = Polynominal time algorithms for discrete logarithms and factoring on a quantum computer, booktitle = Algorithmic Number Theory, First International Symposium, ANTS-I, Ithaca, NY, USA, May 6-9, 1994, Proceedings, series =

Lecture Notes in Computer Science, volume = 877, pages = 289, publisher = Springer, year = 1994, url = https://doi.org/10.1007/3-540-58691-1_68, doi = 10.1007/3-540-58691-1_68, timestamp = Fri, 17 Jul 2020 16:12:48 +0200, biburl = https://dblp.org/rec/conf/ants/Shor94.bib, bibsource = dblp computer science bibliography, https://dblp.org

13. author = Lov K. Grover, editor = Gary L. Miller, title = A Fast Quantum Mechanical Algorithm for Database Search, booktitle = Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996, pages = 212–219, publisher = ACM, year = 1996, url = https://doi.org/10.1145/237814.237866, doi = 10.1145/237814.237866, timestamp = Mon, 26 Nov 2018 15:05:57 +0100, biburl = https://dblp.org/rec/conf/stoc/Grover96.bib, bibsource = dblp computer science bibliography, https://dblp.org
14. title=Telegraphic Code to Insure Privacy and Secrecy in the Transmission of Telegrams, author=Miller, F., lccn=ca07004322, url=https://books.google.co.il/books?id=tT9WAAAAYAAJ, year=1882, publisher=C.M. Cornwell
15. author=Shannon, C. E., journal=The Bell System Technical Journal, title=Communication theory of secrecy systems, year=1949, volume=28, number=4, pages=656-715, doi=10.1002/j.1538-7305.1949.tb00928.x

Partially Disjoint k Shortest Paths
(CSCML Ph.D. Research Track)

Yefim Dinitz[1], Shlomi Dolev[1], Manish Kumar[1], and Baruch Schieber[2]

[1] Ben-Gurion University of the Negev
[2] New Jersey Institute of Technology
{dinitz,dolev@cs,manishk@post}.bgu.ac.il,
baruch.m.schieber@njit.edu

Abstract. A solution of the k shortest paths problem may output paths that are identical up to a single edge. On the other hand, a solution of the k independent shortest paths problem consists of paths that share neither an edge nor an intermediate node. We investigate the case in which the number of edges that are not shared among any two paths in the output k-set is a parameter.
We study two main directions: exploring *near-shortest* paths and exploring *exactly shortest paths*. We assume that the weighted graph $G = (V, E, w)$ has no parallel edges and that the edge lengths (weights) are positive. Our results are also generalized to the cases of k shortest paths where there are several weights per edge, and the results should take into account the multi-criteria prioritized weight.

1 Introduction

Optimizing the cost of paths is fundamental task in Computer Science and Operations Research. In many scenarios, there is a need to compute the second best or in general, the k best alternatives to the optimal solution . The variety in the obtained k-set of best solutions can facilitate a choice of solutions due to other considerations (e.g., preferences of geographic locations or communication channels) among the close to optimal (or allowed budget) solutions. In some cases, the usage of all (or several) solutions from the set is preferable in order to allow the diversity of routing patterns while still being close to the optimal solution.

We believe that the investigation of the set of k near-optimal solutions can be useful in scopes of genome exploration evaluation (e.g., for viruses variants). The distance relation, in this case, is defined by the probability of possible changes in the genome. The k near-optimal solutions enable the tracing of the most probable paths from one variant to another or from one variant to many others.

One variant of this problem is finding k independent shortest paths [9]. This is a challenging computational task, where polynomial algorithms exist only for very limited cases. Thus, it leads to interesting theoretical questions that consider the relaxation of the independence constraint. Note that k 1-edge independent shortest paths (i.e., paths that differ by at least one edge) can be computed in polynomial time. The increase from 1-edge independence to the entire path

independence changes the complexity of the problem dramatically. We found polynomial algorithms for k shortest paths that are ℓ-edge independent that can serve as an intermediate solution instead of totally independent paths.

In a different scenario, where multiple objects or robots want to move (almost) collision-free [14, 10], our algorithms on partially independent paths can be useful. Other applications are possible for multi-agent systems such as object transportation [17, 21], search and rescue [15], robot path reconfiguration[18, 11], tasks spanning assemble [13], evacuation [20], formation control [23].

1.1 Related work

One of the related directions in the literature is studying optimal sets of paths, where a bounded number of shared vertices or edges is allowed. There, a set of paths is called optimal (shortest) if the *sum of the path lengths* is minimum possible, where the length of a path is defined as the sum of the weights of all its contained edges. Observe that this objective does not imply any guarantee on the quality of each path. Guo et al. [12] studied the problem of finding the shortest set of k paths that are edge-disjoint and partially vertex disjoint. They considered the *δ-vertex k edge-disjoint shortest path* ($\delta V - kEDSP$) problem, where at most δ vertices (besides s and t) are shared by at least any two paths. As mentioned there, the $0V-kEDSP$ problem can be simply solved via min-cost max-flow. For $k = 2$ and any positive δ, they solve the $\delta V - kEDSP$ problem in time $O(\delta m + n \log n)$, for a graph $G(V, E, w)$ with $n = |V|$ and $m = |E|$. For general k, the problem is still open. Similarly, the k partially edge-disjoint path problem ($kPESP$) computes the shortest set of k paths connecting s and t such that at most δ edges are shared by at least two paths. For $k = 2$, Yunyun et al. [6] introduced an exact algorithm with a runtime $O(mn \log_{(1+m/n)} n + \delta n^2)$. In the above works, $\delta > 0$ was referred to as the *disjointness* factor.

Chondrogiannis et al. [4] introduced the k Shortest Paths with Limited Overlap *(kSPwLO)* problem seeking to find k alternative paths which are (a) as short as possible and (b) sufficiently dissimilar based on a user-controlled similarity threshold. Given a set of simple paths P from a source s to destination t in an edge weighted graph $G(V, E, w)$, Chondrogiannis et al. called a path $p(s \to t)$ an *alternative path* to P if p is sufficiently dissimilar to every path $p' \in P$. The similarity of two simple paths p and p' is determined by their overlap ratio:

$$Sim(p, p') = \frac{\sum_{(x,y)\in p\cap p'} w_{xy}}{w(p')},$$

where $w(p)$ is the length of a path p, and $p_i \cap p'$ denotes the set of edges shared by p and p. The range of overlap ratio is $0 \leq Sim(p, p') \leq 1$, where $Sim(p, p') = 0$ holds if p shares no edge with p' and $Sim(p, p') = 1$ if $p = p'$. Chondrogiannis et al. introduced two algorithms. The first is a baseline algorithm based on Yen's algorithm [24]. The second algorithm, *OnePass algorithm*, considers the overlap constraint in each expansion step while traversing the network.

In another work by Chondrogiannis et al. [5] they considered the same similarity constraint and introduced the *MultiPass (exact) algorithm* that traverses

the network $k-1$ times and employs pruning criteria to reduce the number of potential alternative paths. The pruning criteria are the same as in the *OnePass algorithm*.

Let P be the given set of paths from source s to destination t, and p_i, p_j be two paths from source s to some intermediate node t'. If $w(p_i) < w(p_j)$ and $\forall p \in PSim(p_i, p) \leq Sim(p_j, p)$ holds, then the path p_j cannot be the prefix of any of the shortest alternative paths to P. It takes $O(m + K \cdot n \cdot \log n)$ time, where K ($K >> k$) is the number of shortest paths that have to be computed in order to cover the k results of the kSPwLO query. In comparison to the OnePass algorithm, the MultiPass algorithm may have to construct all paths from s to t, which results in higher time complexity, but experimental evaluation showed that MultiPass is much faster than OnePass.

Below, we list a sample of previous results on the disjoint paths problem. For a set of k pairs of terminals in graph, the existence of k vertex-disjoint paths connecting each pair of terminals, Robertson and Seymour [19] developed a $O(n^3)$ time algorithm for any fixed k in their graph minor project. The problem of two disjoint shortest paths was first considered by Eilam-Tzoreff [9]. Eilam-Tzoreff provided a polynomial-time algorithm for k= 2, based on a dynamic programming approach for the weighted undirected vertex-disjoint case. This algorithm has a running time of $O(|V|^8)$. Later, Akhmedov [1] improved the algorithm of Eilam-Tzoreff, whose running time is $O(|V|^6)$ for the unit-length case of the 2-Disjoint Shortest Path and $O(|V|^7)$ for the weighted case of the 2-disjoint shortest path. In both cases, Akhmedov [1] considered the undirected vertex disjoint shortest path. In recent past, Bentert et al. [2] improved the result of Akhmedov [1]. In other work of Bérczi et al.[3] they showed that the undirected k-DSPP (disjoint shortest paths problem) and the vertex-disjoint version of the directed k-DSPP can be solved in polynomial time if the input graph is planar and k is a fixed constant. Lochet[16] shows that for any fixed k, the disjoint shortest paths problem admits a slicewise polynomial time algorithm.

1.2 Proposed Approach

We propose to study variation of the k *partially independent shortest paths*. The input to our problem consists of either directed or undirected weighted graph $G(V, E, w)$ with a length $w(e)$ associated with each edge $e \in E$. We are given set of terminals $((s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k))$, where s_i is a source node and t_i is a destination node. A (simple) path p connecting two vertices $s, t \in V$ is a sequence of vertices: $s = v_0, v_1, \ldots, v_r = t$ such that all v_i's are distinct and $(v_i, v_{i+1}) \in E$, for $i \in [0, 1, \ldots, r-1]$. Let $E(p)$ denote the set of edges in the path p, and $w(p)$ denote the length of path p, that is, $w(p) = \sum_{e \in E(p)} w(e)$. The goal is to find partially independent shortest paths connecting s_i to t_i, for $i \in [1, 2, \ldots, k]$.

Independence criteria in shortest path can be defined in several ways, and we list here some of the options we propose to investigate. In the first option we view a path as a set of edges (or nodes) and consider the size of either the set difference or the symmetric difference of two paths as their independence

measure. Another option is based on *pairwise* (edge) independence. For two paths p_1 and p_2, let $\mathsf{IND}(p_1, p_2)$ be their *independence measure*. This measure is defined as a number in $[0, 1]$, with value 0 if either $E(p_1) \subseteq E(p_2)$ or $E(p_2) \subseteq E(p_1)$, and value 1 if p_1 and p_2 are edge disjoint. The goal is to find a collection of k paths $p_1, \ldots, p_k$, where path p_i connects s_i to t_i, for $i \in [1, 2, \ldots, k]$, and the paths are pairwise independent according to the given independence measure. One possible independence measure that we are going to consider is

$$\mathsf{IND}(p_1, p_2) = \min\left\{\frac{|E(p_1)\backslash E(p_2)|}{|E(p_1)|}, \frac{|E(p_2)\backslash E(p_1)|}{|E(p_2)|}\right\}$$

As a warm-up, consider the 2 partially disjoint shortest paths problem and the first option for the independence measure of two paths, namely, the size of the set difference between the edge sets of the two paths. By the pigeonhole principle, any collection of $m + 1$ paths must include a pair of paths whose set difference is at least 2. Such $m + 1$ near-shortest paths can be found using Yen's algorithm [24]. This solves the 2 partially disjoint shortest path problems in this case for independence measure 2.

The paper is structured as follows. In Section 2, we consider the case of partially independent paths, both for the case in which arbitrary k shortest paths are considered (Section 2.1) and the case in which only strictly shortest paths qualify (Section 2.2).

Note that all our results can be generalized to the multi-criteria prioritized weights case. That is, to the case where there are several weights per edge and any arbitrarily small amount of weight i is more important than an arbitrarily big amount of weight j, for any $i < j$. This is possible by the reduction of this case to the case of a single weight provided in [8].

2 Partially Independent Paths

We study two main directions: exploring *near-shortest* paths in Section 2.1 and exploring *exactly shortest paths* in Section 2.2. We assume that the weighted graph $G = (V, E, w)$ has no parallel edges and that the edge lengths (weights) are positive.

2.1 Partially Independent Among k Shortest Paths

We consider the following approach: generate the near-shortest paths in order from best to worst, and find a pair of paths with the highest independence measure among them. The independence measure we consider is the size of the symmetric difference between either the edges or the nodes of the two paths. Our positive results are as follows.

- The first and the second near-shortest paths have at least three different edges and at least one different node.
- Among the first three near-shortest paths, there are two paths with at least four different edges and at least two different nodes.

– (A partly investigated conjecture) Among the first $O(n)$ near-shortest paths, there are two paths with at least six different edges and at least four different nodes.

For the last item, we do not have a full proof. We studied many cases, and still are not sure that all possible cases are revealed.

We also constructed two examples of graphs, where there is no pair of paths with a prescribed number of different edges/nodes among exponentially many first near-shortest paths. The examples show that the effectiveness of the considered approach is bounded (even for planar graphs), since it implies the worst-case time complexity is exponential in the prescribed distance between the two paths.

Version (a) of Example 1 is a planar graph parametrized by $\bar{q}$, where for any $q \leq \bar{q}$, among the first $\Omega((n/\bar{q})^q)$ near-shortest paths, the maximal edge/node distances between two paths are $4q$ edges and $2q$ nodes. Example 1a has the following structure:

– It has $n = \bar{q}(r+1)+1$ nodes, for arbitrary $\bar{q}, r \geq 1$.
– Each path from s to t has $2\bar{q}$ edges.
– The shortest path p^* is composed of $\bar{q}$ pairs of consecutive edges, so that each one of those pairs can be replaced by any one out of $r-1$ other edge pairs.
– The edge weights are integers, and the maximal edge weight is about $r^{\bar{q}}$.

See Figure 1 for an illustration of such a graph with the maximal edge distance of $4q$ and the node distance of $2q$ among all pairs of the first 9^q near-shortest paths, for any $q \leq \bar{q}$.

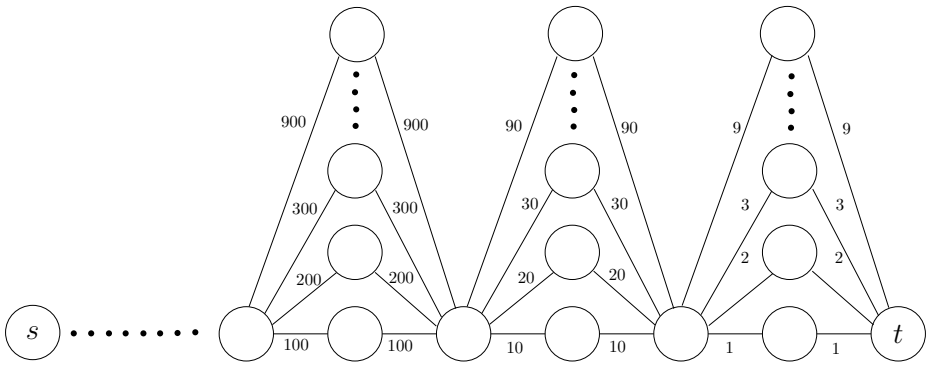


Fig. 1. Example 1a with $r = 9$ and $\bar{q}$ "towers" between s and t.

Version (b) of Example 1 has a simpler structure and eliminates the exponential edge weights by the cost of a bit weaker exponential properties. The

graph on $n = 3n' + 1$ nodes is also planar, constructed as a concatenation of n' diamonds as in Figure 2. There, among the first $\sum_{i=0}^{q} C_{n'}^{q} = \Omega(((n'-q+1)/q)^q)$ near-shortest paths, the maximal edge/node distances between two paths are $8q$ edges and $2q$ nodes, for any $q \leq \lfloor n'/2 \rfloor$.

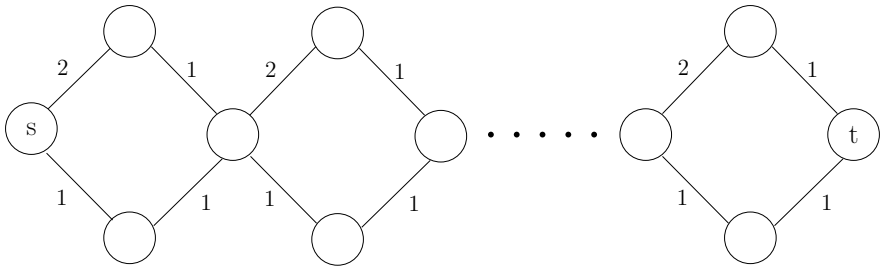


Fig. 2. Example 1b.

Example 2, also parametrized by $\bar{q}$, shows how an arbitrary graph in a wide graph class can be "spoiled" locally so that among the first $\sum_{i=1}^{q} \prod_{j=0}^{i-1} (\bar{q}-j)+1 = \Omega((\bar{q}-q+1)^q)$ near-shortest paths, the maximal edge/node distances between two paths are $2q+2$ edges and $\min\{2q; \bar{q}\}$ nodes, for any $q \leq \bar{q}$.

Let $G_0 = (V, E, w)$ be an arbitrary weighted graph with a *single shortest path* from s to t. Denote that path by p^* and the second shortest path by p_2^*. We set $\epsilon = (w(p_2^*) - w(p^*))/(\bar{q}+2) > 0$. Let v be an arbitrary node at p^*, breaking p^* into p_1 and p_2. We split v into two nodes v' and v'', so that p_1 finishes at v', p_2 starts from v'', and all edges originally incident to v, except for that lying on p_1, are now incident to v''. We now add a complete graph, whose nodes are v', v'' and $\bar{q}$ new nodes and whose edges are of weight ϵ each, to the obtained graph. See Figure 3 for an illustration.

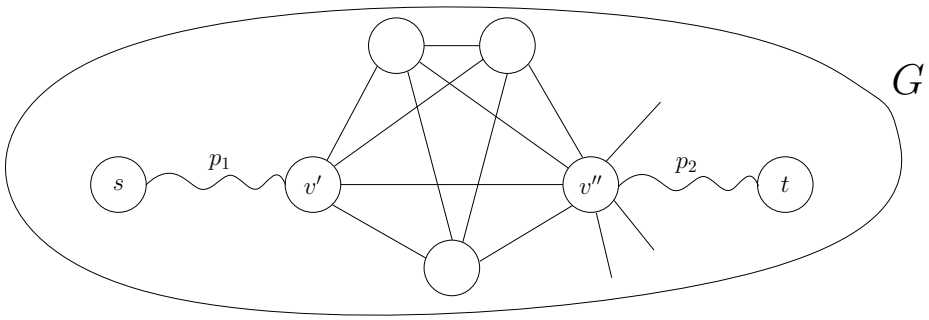


Fig. 3. Example 2 with $\bar{q} = 3$.

Denote the resulting graph by G. The shortest path from s to t in G is $p_1 \circ (v', v'') \circ p_2$ of weight $w(p^*) + \epsilon$. The near-shortest paths in G, in order, are of the form $p_1 \circ p \circ p_2$, where p goes over all paths from v' to v'' of lengths 2, after that 3, and so on up to $\bar{q} + 1$ (note that the weights of all of those paths are strictly less than $w(p_2^*)$). The claimed exponential properties of G are easy to validate.

Note that the boundary case of Example 2 with $s = t$ is just a complete graph on $\bar{q} + 1$ vertices with all edge weights 1.

Another variant of Example 2 arises if we similarly "spoil" G_0 by inserting either version of Example 1 between nodes v' and v'', with the edge weights proportionally decreased, instead of the complete graph as above. Importantly, if the original graph G_0 is planar, then the resulting graph G will also be planar. The properties of this variant are also exponential, though a bit weaker than when using the complete graph.

2.2 Partially Disjoint Shortest Paths

In this section, we study finding partially edge-disjoint paths among the (exactly) shortest paths. The restricted scope allows achieving new interesting results. Our main tool is the *subgraph of shortest paths* $\tilde{G}$ introduced in [8, Section 4].[3] For any graph $G = (V, E, w)$ and its nodes s and t, its subgraph $\tilde{G} = \tilde{G}(s, t)$ is composed of the nodes and edges of all shortest paths from s to t, keeping their weights. If G is undirected, then its edges are directed along the shortest path(s) going through them, so the subgraph of shortest paths $\tilde{G}$ is always directed. Its main properties are as follows. We denote by $d(u, v)$ the distance (=the length of the shortest path) from node u to node v.

- Graph $\tilde{G}$ is acyclic. For any node u of $\tilde{G}$, $d(s, u) + d(u, t) = d(s, t)$. For any edge (u, v) of $\tilde{G}$, $d(s, u) + w(u, v) + d(v, t) = d(s, t)$.
- A path from s to t in G is shortest if and only if it belongs to $\tilde{G}$.
- Any path in $\tilde{G}$ is shortest between its end-nodes in G.
- If v is reachable from u in $\tilde{G}$, then all shortest paths from u to v in G are contained in $\tilde{G}$.

Since we need only the subgraph of shortest paths for studying the shortest paths from s to t, we assume $G = \tilde{G}$ in the follows.

At-first, assume that our goal is finding the maximal number of disjoint shortest paths. Let us build flow network $N_1 = (G, s, t, u_1)$ by assigning capacity $u_1(e) = 1$ to each edge e of G. The following statement is shown in [8, Section 4].

[3] This subgraph may be considered as a generalization of the layered network introduced in [7]. A layered network $L = L(G)$ is a subgraph of the given graph G, such that the set of *all paths* from s to t in L coincides with the set of *all shortest paths* from s to t in G.

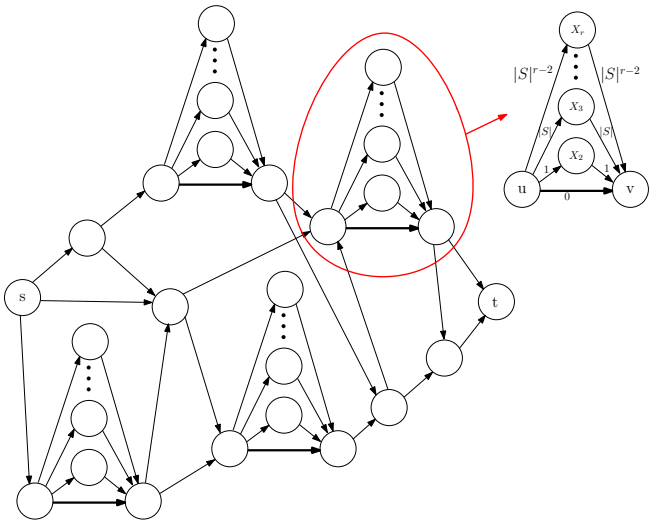


Fig. 4. The construction of Proposition 4. The thick edges belong to S. In each gadget, all edge capacities are 1 and edge costs are as shown in the zoomed copy.

Proposition 1. *The maximal number of disjoint shortest paths is equal to the size of the maximal flow f_{max} in N_1. The set of such paths can then be found by the flow decomposition of f_{max}.*

Let us assume now that a *set of sensitive edges* $S \subseteq E$ is distinguished in G. Let us define flow network $N_2 = (G, s, t, u_2)$ by assigning capacity $u_2(e) = 1$ to each edge $e \in S$ and $u_2(e) = \infty$ to each other edge of G.

Proposition 2. *The maximal number of shortest paths disjoint at the edges in S is equal to the size of the maximal flow f_{max} in N_2. The set of such paths can then be found by the flow decomposition of f_{max}.*

Recall that the case of node capacities can be reduced to that of edge capacities. Therefore, this and the following statements related to S can be extended to the case of the *set $S \subseteq V$ of sensitive nodes* in G.

Assume now that any edge of S may be *overloaded by at most two paths* going along it, and we look for the set of r shortest paths minimizing the number of overloaded edges. Let us define flow network with edge-costs $N_3 = (G, s, t, u_3, c_3)$ by taking N_2, assigning edge costs zero to all its edges, and for any edge $(u, v) \in S$, adding a new node x_2 and a pair of edges (u, x_2) and (x_2, v) of capacity 1 and of cost 1.

Proposition 3. *The set of r shortest paths overloading any edge of S by at most two paths going along it and minimizing the number of overloaded edges can be constructed by finding the min-cost flow $f_{mincost}$ of size r in N_3 and applying to it the flow decomposition.*

Now, assume that any edge of S may be *overloaded by any number of paths* going along it, and the objective is to minimize the maximum overload overall sensitive edges. That is, the loss of overloading even a single sensitive edge by k paths is more than overloading all sensitive edges by $k-1$ paths, for any $k \geq 2$ (the prioritized loss). We look for the set of r shortest paths minimizing the total loss of overloading the sensitive edges. Let us define flow network with edge-costs $N_4 = (G, s, t, u_4, c_4)$ by taking N_3, and for any edge $(u, v) \in S$ and any $i : 3 \leq i \leq r$, adding a new node x_i and a pair of edges (u, x_i) and (x_i, v) of capacity 1 and of cost $|S|^{i-2}$. See Figure 4 for illustration.

Proposition 4. *The set of r shortest paths minimizing the total prioritized loss of overloading sensitive edges can be constructed by finding the min-cost flow $f_{mincost}$ of size r in N_4 and applying to it the flow decomposition.*

3 Concluding Remarks

We have also investigated the k near shortest trees problem not being aware of Sedeño-Noda and González-Martín [22]. We came with a more algorithmic solution that is less efficient, but may bring more understanding to the construction, we differ details to the full version.

References

1. Akhmedov, M.: Faster 2-disjoint-shortest-paths algorithm. In: Computer Science - Theory and Applications - 15th International Computer Science Symposium in Russia, CSR 2020, Yekaterinburg, Russia, June 29 - July 3, 2020, Proceedings. pp. 103–116 (2020)
2. Bentert, M., Nichterlein, A., Renken, M., Zschoche, P.: Using a geometric lens to find k disjoint shortest paths. In: 48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference). pp. 26:1–26:14 (2021)
3. Bérczi, K., Kobayashi, Y.: The directed disjoint shortest paths problem. In: 25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria. pp. 13:1–13:13 (2017)
4. Chondrogiannis, T., Bouros, P., Gamper, J., Leser, U.: Alternative routing: k-shortest paths with limited overlap. In: Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems, Bellevue, WA, USA, November 3-6, 2015. pp. 68:1–68:4 (2015)
5. Chondrogiannis, T., Bouros, P., Gamper, J., Leser, U.: Exact and approximate algorithms for finding k-shortest paths with limited overlap. In: Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017. pp. 414–425 (2017)
6. Deng, Y., Guo, L., Huang, P.: Exact algorithms for finding partial edge-disjoint paths. In: Computing and Combinatorics - 24th International Conference, COCOON 2018, Qing Dao, China, July 2-4, 2018, Proceedings. pp. 14–25 (2018)
7. Dinitz, Y.: Algorithm for solution of a problem of maximum flow in a network with power estimation. In: Soviet Math. Doklady. vol. 11, pp. 1277–1280. Springer (1970)

8. Dinitz, Y., Dolev, S., Kumar, M.: Polynomial time k-shortest multi-criteria prioritized and all-criteria-disjoint paths. In: Cyber Security Cryptography and Machine Learning - 5th International Symposium, CSCML 2021, Be'er Sheva, Israel, 2021, Proceedings. Lecture Notes in Computer Science, Springer (2021)
9. Eilam-Tzoreff, T.: The disjoint shortest paths problem. Discret. Appl. Math. **85**(2), 113–138 (1998)
10. Erdmann, M.A., Lozano-Pérez, T.: On multiple moving objects. Algorithmica **2**, 477–521 (1987)
11. Gajjar, K., Jha, A.V., Kumar, M., Lahiri, A.: Reconfiguring shortest paths in graphs. CoRR **abs/2112.07499** (2021), https://arxiv.org/abs/2112.07499
12. Guo, L., Deng, Y., Liao, K., He, Q., Sellis, T., Hu, Z.: A fast algorithm for optimally finding partially disjoint shortest paths. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden. pp. 1456–1462 (2018)
13. Halperin, D., Latombe, J., Wilson, R.H.: A general framework for assembly planning: The motion space approach. Algorithmica **26**(3-4), 577–601 (2000)
14. Hopcroft, J.E., Wilfong, G.T.: Reducing multiple object motion planning to graph searching. SIAM J. Comput. **15**(3), 768–785 (1986)
15. Jennings, J., Whelan, G., Evans, W.: Cooperative search and rescue with a team of mobile robots. In: 1997 8th International Conference on Advanced Robotics. Proceedings. ICAR'97. pp. 193–200 (1997)
16. Lochet, W.: A polynomial time algorithm for the k-disjoint shortest paths problem. In: Marx, D. (ed.) Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021. pp. 169–178. SIAM (2021)
17. Mataric, M.J., Nilsson, M., Simsarin, K.T.: Cooperative multi-robot box-pushing. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 1995, August 5 - 9, 1995, Pittsburgh, PA, USA. pp. 556–561 (1995)
18. Papadimitriou, C.H., Raghavan, P., Sudan, M., Tamaki, H.: Motion planning on a graph (extended abstract). In: 35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994. pp. 511–520. IEEE Computer Society (1994)
19. Robertson, N., Seymour, P.D.: Graph minors .xiii. the disjoint paths problem. J. Comb. Theory, Ser. B **63**(1), 65–110 (1995)
20. Rodríguez, S., Amato, N.M.: Behavior-based evacuation planning. In: IEEE International Conference on Robotics and Automation, ICRA 2010, Anchorage, Alaska, USA, 3-7 May 2010. pp. 350–355 (2010)
21. Rus, D., Donald, B.R., Jennings, J.: Moving furniture with teams of autonomous robots. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 1995, August 5 - 9, 1995, Pittsburgh, PA, USA. pp. 235–242 (1995)
22. Sedeño-Noda, A., González-Martín, C.: On the K shortest path trees problem. Eur. J. Oper. Res. **202**(3), 628–635 (2010)
23. Smith, B.S., Egerstedt, M., Howard, A.M.: Automatic deployment and formation control of decentralized multi-agent networks. In: 2008 IEEE International Conference on Robotics and Automation, ICRA 2008, May 19-23, 2008, Pasadena, California, USA. pp. 134–139 (2008)
24. Yen, J.Y.: Finding the k shortest loopless paths in a network. Management Science **17**(11), 712–716 (1971)

Exponentially Fast Collaborative Broadcasting in Space*

(Preliminary Version)

Shlomi Dolev[1], Sergey Frenkel[2], Manish Kumar[1], and Christian Scheideler[3]

[1] Ben-Gurion University of the Negev, Israel
[2] Federal Research Center "Computer Science and Control"
Russian Acad. of Sc., Russia
[3] Universität Paderborn, Germany
{dolev@cs,manishk@post}.bgu.ac.il,
fsergei@mail.ru, scheideler@upb.de

Abstract. Broadcast is one of the most important service in the wireless sensor networks (WSNs), where nodes propagate the message across the whole network in all directions. We consider the three-dimensional broadcasting problem for n points uniformly distributed in the space and investigate the exponential growth of three-dimensional ball using propagation in each step. We consider some assumptions such as no signal interference and points being independent.

Keywords: Collaborative broadcast · Three-dimensional environment · Superposition.

1 Introduction and Related Work

In communication networks, the broadcast is a fundamental service that allows nodes to send messages to all other nodes in the network. In the context of the three-dimensional environment such as LEO satellites, Wireless sensor networks (WSNs), Mobile Ad-Hoc Networks (MANET), drug delivery in the body, etc., all communications happen over a wireless medium. These networks have limited energy and computational power. So an efficient and fast broadcasting algorithm is important for overall network performance.

Definition: Let the points $\omega_1, \omega_2, ..., \omega_n$ be uniformly distributed as a binomial point process in the space Ω of a very large but limited volume $|\Omega|$ with binomial parameter $p = Prob(\omega_n \in \Omega)$, interpreted as the process density ρ in the given space. Each point of the process radiates energy E into the surrounding space. Point P is located in this space at distance d (understood as a minimum among all possible segments between P and a point L) from a sphere bounded the ball

* This research was (partially) funded by a grant from the German Research Funding (DFG, Grant #8767581199), the Rita Altura trust chair in computer science, and by the Lynne and William Frankel Center for Computer Science

$B(O, r)$ with radius r (Figure 1). We consider the model of a "growing ball" that exists in space constantly discretely (step by step) increasing its radius with each step $k = 1, 2, ..., n$ with the factor $f = (1 + \epsilon)^k$.

Goal: Find the relation between k and r, in order to check that amount energy in ~~the~~ P, where $E_P > E_{thr}$. E_{thr} is the energy threshold value to study the influence of the relation between k and r on the conditions for exceeding the threshold of the superposition of the emission of points inside the growing ball. The threshold energy is the sensor's own characteristic located at a point and accordingly, its achievement should be ensured by the parameters of the region of space in which the radiating points are located. Epsilon ϵ is expressed as a function $f(\rho, E_{thr})$.

In this work, we focus on exponentially fast broadcasting in a 3D environment. Broadcasting in the 2D environment case was initially studied for the graph-based model, where nodes behave as a transmitter and in a given time step nodes transmit the message to all of their outgoing neighbors in the same time step. A flooding algorithm achieves the optimal bound of the diameter of the network (see [4] for a survey).

Broadcasting can be classified into deterministic and probabilistic approaches. The deterministic approach identifies a subset of nodes and assigns them for forwarding messages. These subsets of the nodes can be cluster-heads in the network [7]. Probabilistic broadcast approache~~s~~, called *gossip*, ~~where~~ nodes in the network forward the message with a pre-specified probability [3].

Sirkeci-Mergen et al. [6] propose a multi-stage cooperative broadcast algorithm where nodes are uniformly distributed in a disk. Their algorithm works in multi-stage, in the first stage, the center node of the disk transmits the message and all nodes that receive the message~~,~~ are considered as level one. In the next stage, level one nodes retransmit the message. In this way set of nodes keeps growing in ~~the~~ outward direction. Schindelhauer et al. [5] reduce the stages of the approach in Sirkeci-Mergen et al [6] and introduce that in each round, all informed nodes send a single message cooperatively instead of alternate rounds.

Dolev et al. [2] consider a self-stabilizing scenario where nodes broadcast *beeps* signals in a specific channel assigned from a set of $O(\log n)$ channels and node locks synchronize using wireless Multiple Input Multiple Output (MIMO), in $O(\log n)$ time units.

In contrast, we consider particles that are uniformly distributed rather than fixed communication graphs, and use energy propagation, whether carried by electromagnetic fields or quantum tunneling (as in photosynthesis [8]) to base our exponential growth in time.

2 Model of the Energy Radiation from a Ball

Let us first present the general scheme of the ball radiation analysis.

Let L be some point inside the ball, the distance of which to the point P must be determined (Fig.1). This distance is equal to the distance of the image

of this point in a flat circle obtained by a section of the ball by a plane passing through three points P, L, O. As is known, only one unique plane passes through three points.

Condition 1. We are not considering signals interference, the propagation of power to the point P does not depend on the propagation angles of the signals, ~~and~~ while all points inside the ball are independent.

Condition 2. The energy emitted by each of point within the ball has the same value E, and each i-th the point contributes the energy in the point P, $E_p^i = E/d_i^a$, where d_i is the distance between a point i and P,

where a is the fading exponent. It is well-known that for the electromagnetic wave $a = 2$ in a far zone, and 1 or 3 in close one, depending on the medium of propagation[1]. Then the amount energy in the point P is estimated as:

$$E_P = \sum_{i=1,n} E_P^i$$

where n is a number of points within the ball $B(O, r)$.

Taking into account that th~~at~~ wave can be propagated in a very specific medium, further, we consider various fading exponents.

3 Exponential Growth

We have to estimate the boundaries $E_P > E_{thr}$ on a step k using the propagation model $E_i/d_j(x_i, k)^a$, where $d(x_i, k)$ is distance between a point x_i, and P on k-th step (round), understood as the length of the segment between point P and the corresponding point x_i (corresponding to a point L in the previous Section).

The obvious bound for the distance $d(x_i, k)$ is

$$d(x_i, k) < d_k + 2r_k$$

where r_k and d_k are the ball radius and the P distance on the k-th round. If $r_k = r(1 + \epsilon)^k$, the distance can be estimated as $d_k \approx d/(1 + \epsilon)^k$, where d, r are the initial values of distance of P from the sphere, and the radius of the sphere.

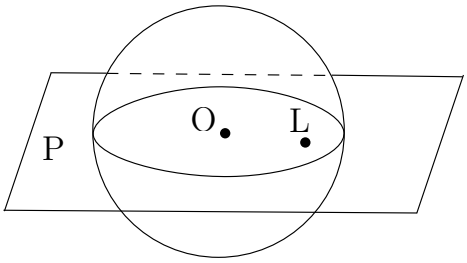


Fig. 1. Ball in the space and section by a plane

Taking into account that for $\epsilon \ll 1$ it is true that $(1+\epsilon)^k \approx (1+k\epsilon)$, we may receive various bounds.

Let the energy at point P ~~is~~ a simple superposition of the energy at all points, i.e. E_P is proportional to the number of points in the ball at the k-th step. Taking into account the definition of uniform distributed points density within the ball as ρ = number of points/V, V is a volume, let us receive the number n_k of points on the k-th step of the ball growth. Since for the ball of the radius r, $V = (4/3)\pi r^3$, we get for the k-th ball with radius $r_k = r(1+\epsilon)^k$

$$n_k = 4/3\rho\pi r^3(1+3k\epsilon)$$

where we use the fact that $(1+\epsilon)^k = 1+k\epsilon$, if $\epsilon \ll 1$

~~But~~ in this case, the energy E_p depends on the current radius, and hence on k. Indeed, at a constant density, the larger the radius, the more points in the ball segments closest to the boundary of the sphere, and the greater their contribution to energy at the point in question. The only thing that can be done to make ϵ independent of k is to find a range of values that provide a threshold value for any k.

~~Let's, f~~or example, assume that the relevant value of $a = 3$. Taking into account our bounds for the distance, we can get:

$$\epsilon \geq 1/3((E_{thr}(d+2r)^a/E \times 1.33\pi\rho) - 1)$$

where E is the energy emanating by each of the points within the ball.

The formula is obtained under fairly broad assumptions about the distance of the outer point from the inner ones (which are used in the previous section) and a linear approximation, assuming that the epsilon is small enough, and $3k\epsilon \gg k^3\epsilon^3$.

It is easy to see that this bound for ϵ correctly reflects the physics.

Indeed, the more energy reflected by a point (node) within a current ball, the less impact of change (growth) in the ball value can ~~arise~~ the energy ~~in the~~ point P exceeding the threshold. The same conclusion about the density r and vice versa, the more initial distance d of the point P from the original spherical bound of the ball, the more growth factor should be used.

References

1. Bienkowski, P., Trzaska, H.: Electromagnetic measurements in the near field. SciTech Publishing (2011)
2. Dolev, S., Narayanan, R.P., Scheideler, C., Schindelhauer, C.: Logarithmic time MIMO based self-stabilizing clock synchronization. In: NANOCOM '21: The Eighth Annual ACM International Conference on Nanoscale Computing and Communication, Virtual Event, Italy, September 7 - 9, 2021. pp. 30:1–30:2 (2021)
3. Kyasanur, P., Choudhury, R.R., Gupta, I.: Smart gossip: An adaptive gossip-based broadcasting service for sensor networks. In: IEEE 3rd International Conference on Mobile Adhoc and Sensor Systems, MASS 2006, 9-12 October 2006, Vancouver, BC, Canada. pp. 91–100 (2006)

4. Peleg, D.: Time-efficient broadcasting in radio networks: A review. In: Distributed Computing and Internet Technology, 4th International Conference, ICDCIT 2007, Bangalore, India, December 17-20, Proceedings. pp. 1–18 (2007)
5. Schindelhauer, C., Oak, A., Janson, T.: Collaborative broadcast in (log log n) rounds. In: Algorithms for Sensor Systems - 15th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS 2019, Munich, Germany, September 12-13, 2019, Revised Selected Papers. pp. 119–136 (2019)
6. Sirkeci-Mergen, B., Scaglione, A., Mergen, G.: Asymptotic analysis of multistage cooperative broadcast in wireless networks. IEEE Trans. Inf. Theory **52**(6), 2531–2550 (2006)
7. Stojmenovic, I., Seddigh, M., Zunic, J.D.: Dominating sets and neighbor elimination-based broadcasting algorithms in wireless networks. IEEE Trans. Parallel Distributed Syst. **13**(1), 14–25 (2002)
8. Van Grondelle, R., Novoderezhkin, V.I.: Quantum effects in photosynthesis. Procedia Chemistry **3**(1), 198–210 (2011), 22nd Solvay Conference on Chemistry

Common Public Knowledge for Enhancing Machine Learning Data Sets (Preliminary Version)

Shlomi Dolev and Arnon Ilani*.

Ben-Gurion University of the Negev, Be'er Sheva, Israel
{dolev@cs,arnonil@post}.bgu.ac.il

Abstract. In this paper, we demonstrate the benefit of adding multi-source knowledge based on established public knowledge such as the knowledge found in Wikipedia into existing datasets for improving machine learning results on everyday tasks (e.g., classification). We explore various added values for better classification of unseen data, improving classification on classes the model is not trained on.
Specifically, we demonstrate the need to use common public knowledge in a challenge in the scope of Forest Cover Type Prediction initiated by the Roosevelt National Forest of Northern Colorado [1].

Keywords: Ontology · Machine learning · Random forests · Feature engineering · World knowledge · Forest Management · Tree cover type.

1 Introduction

The central theme of this research is to show how public knowledge not included in the data set can be beneficial in improving machine learning results for a variety of tasks. Our goal is to demonstrate how public scientific/social/historical knowledge accumulated over the years can enhance a data set used in many specific machine learning processes. The same as the exact sciences aim to formulate mathematical rules to calculate the outcomes of an experiment. The findings of these experiments and analytical formulation can assist the machine learning process, adding insight to the data set. An established source is currently a loose definition of a knowledge base found in public. For example, these knowledge sources can be Wikipedia or other professional/historical ontologies[2] and taxonomies.

The main goal is to show immediate benefit from using such common/public outside the dataset knowledge. The benefit is obtained by adding orthogonal knowledge not present in the data set that can improve classification or other tasks. Results compared to using only the original dataset. A secondary goal is to provide guidelines to what knowledge is more rewarding than others and find parameters for knowledge by considering the data source.

* Corresponding auther

The rest of this paper is structured as follows. Section 2 discusses related work. Section 3 discusses the research flow in general. Section 4 presents classification over the Covertype dataset. Section 5 discusses outside knowledge sources. Section 6 presents classification results and Section 7 concludes the paper.

2 Related Work

Next, we describe recent research done especially considering results on the Forest Cover Type dataset obtained from [3]:

Table 1: Research results on the Forest Cover Type dataset over the years

Reference	Year	Leading method	Accuracy
Kishore, Rahul R. et al. [4]	2016	Random forests algorithm	87.66%
De Almeida et al. [5]	2016	Dynse K-E92 method	80.5%
Krawczyk and Wozniak. [6]	2015	One-class classification model with different incremental learning and forgetting procedures	75.76%
Rojanavasu et al. [7]	2009	sUpervised Classifier System (UCS)	72.90%
Narayan et al. [8]	2006	k-dimensional tree with Repeated Bisection technique	64.32%
Prudhomme and Lallich. [9]	2005	Kohonen Opt technique with normalization of the attributes	67.80%
Garcke and Griebel. [10]	2005	sparse grid	70.30%
Castro et al. [11]	2005	Fuzzy ARTMAP (FAM) with data partitioning in regions	76.00%
Oza. [12]	2005	Bagging with multilayer perceptrons	77.87%
Secretan et al. [13]	2005	No Matchtracking Fuzzy ARTMAP	79.28%
Pal and Mitra. [14]	2004	Rough-fuzzy algorithm	67.01%
Liu et al. [15]	2004	k-NN classification with IOC algorithm	88.00%
Koggalage and Halgamuge. [16]	2004	Support vector machine classifier, classification for class 1, 2 and 5 only	89.72%
Liu et al. [17]	2004	Decision tree classifier	90.00%
Mitra et al. [18]	2002	k-NN Algorithm	67.75%
Yang and Webb. [19]	2002	Naïve Bayes with nondisjoint discretization method	68.60%
Demiriz et al. [20]	2004	AdaBoost	74.75%
Frank et al. [21]	2002	Loglikelihood pruning	82.50%
Furnkranz. [22]	2001	R3	66.80%
Lazarevic and Obradovic. [23]	2001	Progressive sampling technique with a boosting algorithm	71.00%

Lazarevic and Obradovic. [24]	2001	Distributed boosting technique	73.20%
Kaburlasos and Petridis. [25]	2000	Backpropagation	70.00%
Blackard. [26]	1999	Artificial neural network	70.58%
Arvind Kumar and Nishant Sinha. [3]	2020	Random forests algorithm	94.60%

The next section presents the research flow we have developed

3 Research flow

The research workflow can be described by a few logical steps:

(a) Finding a well-detailed challenge with as much context of the experiment as possible. This is an important precursor to the next steps since related knowledge is necessarily out of the scope of what was directly measured.
(b) The dataset needs to have meaningful preferably without raw measurement features. Raw measurements such as pixels and frequencies by themselves do not have context. That is why a more composite feature type is preferred.
(c) Outside knowledge hunt. The main tool to seek related knowledge is running web queries and finding publicly available knowledge. A new NLP-based text search tool[1] can be used and Google search as well.
(d) Prepare a new dataset with added knowledge and compare task results on the original compared to the modified dataset.

3.1 Research use cases

In principle, we would like to provide several use cases with improved results simply by adding outside knowledge sources. Use cases may differ in various ways yet all benefit from added-features, otherwise missing from the original experiment. The first use case we can show results for is the Forest Cover Type Prediction challenge.

3.2 What we do with an outside knowledge source

The experiments done in this paper show how outside knowledge is used in various ways.

Detailed below is a work-in-progress list of ways to utilize outside knowledge:

1. Construct new features based on one or more existing features

[1] https://spike.apps.allenai.org/datasets

  - To fine-tune existing features. For example in the forest cover type using the soil type with 40 types in total to create three new features one of them is Available Water Storage which has a real value primitive type. The effect is extending the dynamic range for the model to learn finer details in the context of soil type which already exists in the original dataset.
  - Compute a new feature. If-then-else type of new features.
2. Construct new features based on existing classes
  - e.g., mapping from tree cover type class (e.g. Spruce or Lodgepole) to a "Shade Tolerant" feature value.

One goal is to predict unseen[2] data-category for a feature. Trying to show that outside sources can improve prediction over unseen data-categories hence prediction can rely more on outside knowledge than hard-to-collect measurements.

4 Classification over the Covertype dataset

The Covertype dataset[3] is a forest cover type dataset for the classification of tree cover types given various measurements. The dataset is tabular with both numerical and categorical primitives types. The dataset contains 581,012 instances[4], 7 tree cover types (target classes) and 12 features: Elevation, Aspect, Slop, Horizontal Distance To Hydrology, Vertical Distance To Hydrology, Horizontal Distance To Roadways, Hillshade 9am, Hillshade Noon, Hillshade 3pm, Horizontal Distance To Fire Point, Wilderness Area, Soil Type:

1. Spruce
2. Lodgepole pine
3. Ponderosa pine
4. Cottonwood/willow
5. Aspen
6. Douglas fir
7. Krummholz

A detailed description and analysis of this dataset can be found in a modeling study[27, 5] from 2017. In this work, a limited dataset was considered with only three original features: Elevation, Wilderness Area, Soil Type.

The following Feature engineering section describes the process of adding outside knowledge.

[2] artificially unseen in the train set.
[3] https://archive-beta.ics.uci.edu/ml/datasets/covertype
[4] an instance is a forest cover type for a 30m by 30m cell taken from USDA Forest Service.
[5] United States Forest Service Improving Management of Forest Cover Modeling Study 100189521,by Douglas Fraser, May 4, 2017.

The model chosen for the classification task is Random Forest classifier from sklearn.

After adding obtained features the next step is to divide the dataset between train and test sets. As described earlier the goal is to train a model on seen-categories and test on an unseen category. Wilderness Area feature has four categories:

- Rawah Wilderness Area
- Neota Wilderness Area
- Comanche Peak Wilderness Area
- Cache la Poudre Wilderness Area

To conclude the end goal is to show classification improvement of unseen-wilderness-area datapoints on the dataset with outside knowledge compared to classification on the dataset without.

4.1 Feature engineering

In the cover type dataset, the "Soil Type" feature has a range of 40 discrete values following the USFS ELU Code as described in Fig 1. For each instance in the dataset three new values were added from the USDA-NCSS soil survey corresponding to the "Soil Type" value as described in Fig 2.

The Shade Tolerant feature is a simple category-type determined by tree cover type which corresponds to the dataset target classes conveniently.

These are the four Shade Tolerant categories:

- Shade intolerant
- Intermediate shade tolerant
- Shade tolerant.
- Shade tolerant undefined. [6]

The same goes for the next two features: Fire Return Interval Range Min (in short: FRIR_Min) and Fire Return Interval Range Max (in short: FRIR_Max). Both are mapped conveniently to correspond to the target classes.

These are FRIR min values:

- Spruce = 35
- Douglas fir = 25
- Lodgepole pine = 25
- Cottonwood/willow = 18.8 [7]
- Aspen = 7
- Ponderosa pine = 2

[6] For the Krummholz tree cover type since we did not find information regarding it.
[7] In the absence of information, we took the average of the other five values.

– Krummholz = 18.8 [8]

These are FRIR max values:

– Spruce = 200
– Douglas fir = 100
– Lodgepole pine = 340
– Cottonwood/willow = 158 [9]
– Aspen = 120
– Ponderosa pine = 30
– Krummholz = 158 [10]

Ending up with a total of six new features to add:

– Dominant Geomorphic Position -- categorical features corresponding to the soil geomorphic position e.g mountain slopes, drainage-ways, terraces, etc. 15 categories in total.
– Available Water Storage -- real value with two digits after the decimal point.
– Water Source Percentage -- integer value in [0,100] range.
– Shade Tolerant -- four categories in total [Shade tolerant, Intermediate shade tolerant, Shade intolerant, Shade tolerant undefined].
– Fire Return Interval Range Min -- real value with one digit after the decimal point.
– Fire Return Interval Range Max -- real value with one digit after the decimal point.

4.2 Training and testing a Random Forest model

The Random Forest model is chosen with hyperparameters selection model_selection RandomizedSearchCV from sklearn.

Random Forest model training settings:

– "n_estimators": np.arange(100, 1500, 100)
– "max_depth": np.arange(1, 20)
– "criterion": ["gini", "entropy"]
– n_iter=7,
– cross validation = 3

Running both train + test twice on a dataset with and a dataset without added-features. The previous step is done four times each time choosing (and splitting the dataset accordingly) a different "unseen" wilderness area to be the test set and the remaining three wilderness areas as the train set.

Performance measured is the accuracy of multi-class classification. Comparing accuracy achieved without added-features vs accuracy achieved with added-features. Using classification_report from sklearn.metrics

[8] In the absence of information, we took the average of the other five values.
[9] In the absence of information, we took the average of the other five values.
[10] In the absence of information, we took the average of the other five values.

Soil Type	USFS ELU Code	Description
1	2702	Cathedral family - Rock outcrop complex, extremely stony
2	2703	Vanet - Ratake families complex, very stony
3	2704	Haploborolis - Rock outcrop complex, rubbly
4	2705	Ratake family - Rock outcrop complex, rubbly
5	2706	Vanet family - Rock outcrop complex complex, rubbly
6	2717	Vanet - Wetmore families - Rock outcrop complex, stony
7	3501	Gothic family
8	3502	Supervisor - Limber families complex
9	4201	Troutville family, very stony
10	4703	Bullwark - Catamount families - Rock outcrop complex, rubbly
11	4704	Bullwark - Catamount families - Rock land complex, rubbly
12	4744	Legault family - Rock land complex, stony
13	4758	Catamount family - Rock land - Bullwark family complex, rubbly
14	5101	Pachic Argiborolis - Aquolis complex
15	5151	unspecified in the USFS Soil and ELU Survey
16	6101	Cryaquolis - Cryoborolis complex
17	6102	Gateview family - Cryaquolis complex
18	6731	Rogert family, very stony
19	7101	Typic Cryaquolis - Borohemists complex
20	7102	Typic Cryaquepts - Typic Cryaquolls complex
21	7103	Typic Cryaquolls - Leighcan family, till substratum complex
22	7201	Leighcan family, till substratum, extremely bouldery
23	7202	Leighcan family, till substratum - Typic Cryaquolls complex
24	7700	Leighcan family, extremely stony
25	7701	Leighcan family, warm, extremely stony
26	7702	Granile - Catamount families complex, very stony
27	7709	Leighcan family, warm - Rock outcrop complex, extremely stony
28	7710	Leighcan family - Rock outcrop complex, extremely stony
29	7745	Como - Legault families complex, extremely stony
30	7746	Como family - Rock land - Legault family complex, extremely stony
31	7755	Leighcan - Catamount families complex, extremely stony
32	7756	Catamount family - Rock outcrop - Leighcan family complex, extremely stony
33	7757	Leighcan - Catamount families - Rock outcrop complex, extremely stony
34	7790	Cryorthents - Rock land complex, extremely stony
35	8703	Cryumbrepts - Rock outcrop - Cryaquepts complex
36	8707	Bross family - Rock land - Cryumbrepts complex, extremely stony
37	8708	Rock outcrop - Cryumbrepts - Cryorthents complex, extremely stony
38	8771	Leighcan - Moran families - Cryaquolls complex, extremely stony
39	8772	Moran family - Cryorthents - Leighcan family complex, extremely stony
40	8776	Moran family - Cryorthents - Rock land complex, extremely stony

Fig. 1. USFS ELU Code soil type [28]

Soil type	USFS ELU Code	Dominant Geomorphic Position	Available Water Storage	Water Source Percentage
1	2702	mountain slopes	2.26	0
2	2703	mountain slopes	2.21	0
3	2704	mountain slopes	4.24	0
4	2705	mountain slopes	2.8	0
5	2706	mountain slopes	1.5	0
6	2717	mountain slopes	2.71	0
7	3501	structural benches	11.81	0
8	3502	mountain slopes benches	9.3	0
9	4201	moraines	7.99	0
10	4703	mountain slopes	4.33	0
11	4704	mountain slopes	4.22	0
12	4744	benches	1.86	0
13	4758	mountain slopes mountainsides	3.11	0
14	5101	stream terraces alluvial flats	10.82	0
15	5151	none	0	0
16	6101	drainageways terraces	14.22	0
17	6102	drainageways terraces	11.5	0
18	6731	mountainsides	1.7	0
19	7101	drainageways	27.56	5
20	7102	drainageways	12.59	5
21	7103	mountain slopes drainageways	13.08	2
22	7201	mountain slopes	5.88	0
23	7202	moraines drainageways	9.69	5
24	7700	mountain slopes	5.88	0
25	7701	mountain slopes	5.88	0
26	7702	mountain slopes benches	5.58	0
27	7709	mountain slopes	5.88	0
28	7710	mountain slopes	5.88	0
29	7745	mountain slopes benches	4.99	0
30	7746	mountain slopes mountainsides	5.55	0
31	7755	mountain slopes	4.18	0
32	7756	mountain slopes	3.81	0
33	7757	mountain slopes	4.67	0
34	7790	mountain slopes mountainsides	2.88	0
35	8703	solifluction lobes glacial-valley floors	6.13	0
36	8707	mountain slopes	5	0
37	8708	solifluction lobes mountain slopes	3.78	5
38	8771	mountain slopes drainageways	10.29	0
39	8772	mountain slopes	5.52	0
40	8776	mountain slopes	6.22	0

Fig. 2. Soil type obtained features [29]

5 Outside knowledge sources

Three outside knowledge sources were found and used:

1. The USDA-NCSS soil survey [11] from which features: Dominant Geomorphic Position, Available Water Storage, Water Source Percentage were chosen.
   – "This app was developed by the California Soil Resource Lab at UC Davis and UC-ANR in collaboration with the USDA Natural Resources Conservation Service." [29]
2. Wikipedia's page: List of tree species by Shade Tolerance[30] [12] feature was taken.
3. The Fire Effects Information System (FEIS)[31] [13] from which features: Fire Return Interval Range Min and Fire Return Interval Range Max were chosen.
   – "The Fire Effects Information System is an online collection of reviews of the scientific literature about fire effects on plants and animals and about fire regimes of plant communities in the United States. FEIS reviews are based on thorough literature searches, often supplemented with insights from field scientists and managers. FEIS provides reviews that are efficient to use, thoroughly documented, and defensible. Approximately 15 to 30 new or revised reviews are published in FEIS each year" [32]

Notice that an outside knowledge source has some characteristics:

– Add knowledge over a feature or a class. In our case, the USDA-NCSS soil survey relies on the Soil Type feature and the other two: Shade Tolerance and Fire Return Interval Range rely on the class type.
– Can be categorical or numerical.
– Can be complete or incomplete. Complete means that a bijective function exists between the outside knowledge source set and the independent set in the dataset. The Fire Return Interval Range set to Tree Cover classes set is incomplete and some adjustments were needed to fill in for the missing mapping. Mainly assign the average value for the missing mappings.
– Can in principle have a more complex relationship between existing datapoints (both features and classes) hence: a more complex mapping function needed to engineer a new feature. In our case the mapping is a simple assignment function.

Some fun facts on the outside knowledge sources: (later we may formulate outside knowledge properties into a more concrete ranking formula)

– It is written: "Pinus contorta, with the common names lodgepole pine and shore pine, and also known as twisted pine,and contorta pine" [14]. In the

[11] https://casoilresource.lawr.ucdavis.edu/gmap/
[12] https://en.wikipedia.org/wiki/List_of_tree_species_by_shade_tolerance
[13] https://www.fs.fed.us/database/feis/plants/tree/pinconl/all.html#201
[14] https://en.wikipedia.org/wiki/Pinus_contorta

target class, we have "lodgepole pine" type which in [31] the is called "Pinus contorta". Hence finding synonyms are basic step towards matching knowledge sources for the same-entity with different names.
- In [31] we can also find knowledge about other geographical areas such as Glacier National Park, Montana which is about 900-940 miles away, also other areas such as Yellowstone National Park, Wyoming which is about 500 miles away. Hence this outside knowledge source has a broader perspective than the areas given by the dataset.
- In [32] we learn that several central agencies manage the scientific work presented by the FEIS publication. Those agencies are: United States Department of Agriculture, Forest Service, United States Department of Interior, Bureau of Indian Affairs, Bureau of Land Management, Fish and Wildlife Service, and National Park Service. This gives much credibility compared to a Wikipedia page.
- In [30] there are 7 sub-types of Spruce: Engelmann Spruce, Sitka Spruce, Colorado Blue spruce, Norway Spruce, White Spruce, Black Spruce, Red Spruce. If the given dataset was more elaborated this knowledge source with already over 150 tree species can be reused to support more target classes.
- [29] opens up a full taxonomy of soil [15] from which we only scratched the surface. Again this is a knowledge source much more established compared to a Wikipedia page and is based on long standing research in soil science

6 Classification results

Accuracy results for :

- For unseen Rawah Wilderness Area (only present in the test set):
  - with support size (#of instances[16]) = 260796 in both cases (biggest group)
  - Without added features: 72%
  - With! added features: **100%**
  - **improvement of 28%**
- For unseen Neota Wilderness Area (only present in the test set):
  - with support size (#of instances) = 29884 in both cases (smallest group)
  - Without added features: 60%
  - With! added features: **100%**
  - **improvement of 40%**
- For unseen Comanche Peak Wilderness Area (only present in the test set):
  - with support size (#of instances) = 253364 in both cases
  - Without added features: 61%
  - With! added features: **100%**
  - **improvement of 39%**

[15] https://www.nrcs.usda.gov/wps/portal/nrcs/main/soils/survey/class/taxonomy/
[16] An instance is a forest cover type for a 30m by 30m cell taken from USDA Forest Service

– For unseen Cache la Poudre Wilderness Area (only present in the test set):
  • with support size (#of instances) = 36968 in both cases
  • Without added features: 14%
  • With! added features: 93%
  • **improvement of 79%!**

Notes:

– Improvement over **all** 4 unseen categories achieved!
– Support size equals the total unseen-category size in the full dataset.
– The Shade Tolerant feature was added as-is I did not rely (they were not included) on the existing features: Shade at 9am feature, Shade at noon feature, Shade at 3pm feature

7 Conclusions

Easy to see how accuracy improvement ranging from [28% up to 79%] on the provided dataset is significant. This is a promising first demo to show how outside knowledge can enrich existing datasets achieving better classification.

There is also a practical benefit. From the standpoint of data collection, as some existing features in the dataset were not needed to reach 100% accuracy the efforts and cost of data collection and processing of those "left-out" features can be saved.

But maybe the most interesting part is that by a new train/test division it is possible to use "seen" measurements to classify "unseen" measurements given target classes are identical. From an application standpoint, this opens up the possibility of classifying unreachable or unknown measurements per feature. e.g. classify tree cover types in a new wilderness area that was not included in the original experiment (given that other conditions are the same)

Some forward-looking research thoughts/directions:

– For the same tree cover type dataset: is it possible to classify well over two unseen wilderness areas out of a total of four wilderness areas
– For the same tree cover type dataset: is it possible to classify well over unseen soil types.
– For the same tree cover type dataset: is it possible to classify well over unseen target class (one of the cover types).
– For the same tree cover type dataset: analyze which outside knowledge contribute the most and why
– Continue to demo similar behavior in other datasets in a supervised learning setting.
– Continue to demo similar behavior in other datasets in a (self) unsupervised learning settings.

References

1. Jock Blackard. Covertype. UCI Machine Learning Repository, 1998.
2. A. Gómez-Pérez, M. Fernandez-Lopez, and O. Corcho. *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web. First Edition*. Advanced Information and Knowledge Processing. Springer London, 2006.
3. Arvind Kumar and Nishant Sinha. Classification of forest cover type using random forests algorithm. 2020.
4. Rahul R. Kishore, Shalvin S. Narayan, Sunil Lal, and Mahmood A. Rashid. Comparative accuracy of different classification algorithms for forest cover type prediction. In *2016 3rd Asia-Pacific World Congress on Computer Science and Engineering (APWC on CSE)*, pages 116–123, 2016.
5. Paulo Ricardo Lisboa de Almeida, Luiz Oliveira, Alceu de Souza Britto, and Robert Sabourin. Handling concept drifts using dynamic selection of classifiers. *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 989–995, 2016.
6. B. Krawczyk and Michał Woźniak. One-class classifiers with incremental learning and forgetting for data streams with concept drift. *Soft Computing*, 19:3387–3400, 2015.
7. Pornthep Rojanavasu, Hai Huong Dam, Hussein A. Abbass, Christopher J. Lokan, and Ouen Pinngern. A self-organized, distributed, and adaptive rule-based induction system. *IEEE Transactions on Neural Networks*, 20:446–459, 2009.
8. B. Lakshmi Narayan, C. A. Murthy, and Sankar K. Pal. Maxdiff kd-trees for data condensation. *Pattern Recognit. Lett.*, 27:187–200, 2006.
9. Elie Prudhomme and Stéphane Lallich. Quality measure based on kohonen maps for supervised learning of large high dimensional data. 01 2005.
10. Jochen Garcke and Michael Griebel. Semi-supervised learning with sparse grids. In *ICML 2005*, 2005.
11. José Castro, Michael Georgiopoulos, Jimmy Secretan, Ronald F. Demara, Georgios C. Anagnostopoulos, and Avelino J. Gonzalez. Parallelization of fuzzy artmap to improve its convergence speed: The network partitioning approach and the data partitioning approach. *Nonlinear Analysis-theory Methods & Applications*, 63, 2005.
12. Nikunj C. Oza and Stuart J. Russell. Online bagging and boosting. *2005 IEEE International Conference on Systems, Man and Cybernetics*, 3:2340–2345 Vol. 3, 2005.
13. Jimmy Secretan, José Castro, Michael Georgiopoulos, J. Tapia, Amit Chadha, B. Huber, Georgios C. Anagnostopoulos, and Samuel Richie. Parallelizing the fuzzy artmap algorithm on a beowulf cluster. *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, 1:475–480 vol. 1, 2005.
14. Sankar K. Pal and Pabitra Mitra. Case generation using rough sets with fuzzy representation. *IEEE Transactions on Knowledge and Data Engineering*, 16:293–300, 2004.
15. Ting Liu, Ke Yang, and Andrew W. Moore. The ioc algorithm: efficient many-class non-parametric classification for high-dimensional data. *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2004.
16. Ravindra Koggalage and Saman K. Halgamuge. Reducing the number of training samples for fast support vector machine classification. 2004.

17. Xiaomei Liu, K. Bowyer, and Lawrence O. Hall. Decision trees work better than feed-forward back-prop neural nets for a specific class of problems. *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)*, 6:5969–5974 vol.6, 2004.
18. Pabitra Mitra, C. A. Murthy, and Sankar K. Pal. Unsupervised feature selection using feature similarity. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24:301–312, 2002.
19. Ying Yang and Geoffrey I. Webb. Non-disjoint discretization for naive-bayes classifiers. In *ICML*, 2002.
20. Ayhan Demiriz, Kristin P. Bennett, and John Shawe-Taylor. Linear programming boosting via column generation. *Machine Learning*, 46:225–254, 2004.
21. Eibe Frank, Geoff Holmes, Richard Kirkby, and Mark A. Hall. Racing committees for large datasets. In *Discovery Science*, 2002.
22. Johannes Fürnkranz. Round robin rule learning. In *ICML*, 2001.
23. Aleksandar Lazarevic and Zoran Obradovic. Data reduction using multiple models integration. In *PKDD*, 2001.
24. Aleksandar Lazarevic and Zoran Obradovic. The distributed boosting algorithm. In *KDD '01*, 2001.
25. Vassilis G. Kaburlasos and Vassilios Petridis. Fuzzy lattice neurocomputing (fln) models. *Neural networks : the official journal of the International Neural Network Society*, 13 10:1145–69, 2000.
26. J. A. Blackard and Denis J. Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture*, 24:131–151, 1999.
27. Douglas Fraser. Improving management of forest cover modeling study 100189521. *United States Forest Service*, 2017.
28. Douglas Fraser. Improving management of forest cover modeling study 100189521. *United States Forest Service*, page 22, 2017.
29. The usda-ncss soil survey. *https://casoilresource.lawr.ucdavis.edu/gmap*.
30. List of tree species by shade tolerance. *https://en.wikipedia.org/wiki/List_of_tree_species_by_shade_tolerance*.
31. Michelle D. Anderson. Feis species review: Pinus contorta var. latifolia (pinconl). 2003.
32. About the fire effects information system (feis). *https://www.fs.fed.us/database/feis/AboutFEIS/about.html*.

Self Masking for Hardening Inversions

Paweł Cyprys[1], Shlomi Dolev[1], and Shlomo Moran[2]

[1] Ben-Gurion University of the Negev
[2] Technion Israel Institute of Technology
August 16, 2022

Abstract. The question whether one way functions (i.e., functions that are easy to compute but hard to invert) exist is arguably one of the central problems in complexity theory, both from theoretical and practical aspects. While proving that such functions exist could be hard, there were quite a few attempts to provide functions which are one way "in practice", namely, they are easy to compute, but there are no known polynomial time algorithms that compute their (generalized) inverse (or that computing their inverse is as hard as notoriously difficult tasks, like factoring very large integers).
In this paper we study a different approach. We provide a simple heuristic, called self masking, which converts a given polynomial time computable function f into a self masked version $[f]$, which satisfies the following: for a random input x, $[f]^{-1}([f](x)) = f^{-1}(f(x))$ w.h.p., but a part of $f(x)$, which is essential for computing $f^{-1}(f(x))$ is *masked* in $[f](x)$. Intuitively, this masking makes it hard to convert an efficient algorithm which computes f^{-1} to an efficient algorithm which computes $[f]^{-1}$, since the masked parts are available to f but not to $[f]$.
We apply this technique on variants of the subset sum problem which were studied in the context of one way functions, and obtain functions which, to the best of our knowledge, cannot be inverted in polynomial time by published techniques.

1 Introduction

The question whether one way functions (i.e., functions that are easy to compute but hard to invert) exist is arguably one of the central problems in complexity theory, both from theoretical and practical aspects. e.g., it is known that the existence of one way functions implies, and is implied by, the existence of pseudo random number generators (see e.g. [6] for a constructive proof of this equivalence).

While proving that one way functions exist could be hard (since it would settle affirmatively the conjecture that $P \neq NP$), there were quite a few attempts to provide functions which are one way "in practice" – namely, they are easy to compute, but there are no known polynomial time algorithms which compute their (generalized) inverses.

In this paper we suggest a heuristic, called *self masking*, to cope with published attacks on previous attempts to construct one way functions. Specifically,

the self masking versions of polynomial time computable functions "hide" in the outputs of these functions parts which are essential for computing their inverse.

1.1 Preliminaries

To make the presentation self contained and as short as possible, we present only definitions which are explicitly used in our analysis. For a more comprehensive background on one way functions and related applications see, e.g., [7; 6].

The notation $x \in_{\mathcal{U}} D$ indicates that x is a member of the (finite) set D, and that for probabilistic analysis we assume a uniform distribution on D.

Following [6], we define one way functions using the notion of *polynomial time function ensembles.*

Definition 1 *A polynomial time function ensemble $f = (f_k)_{k=1}^{\infty}$ is a polynomial time computable function that, for a strictly increasing sequence $(n_k)_{k=1}^{\infty}$ and a sequence $(m_k)_{k=1}^{\infty}$, f_k maps $\{0,1\}^{n_k}$ to $\{0,1\}^{m_k}$. Both n_k and m_k are bounded by a polynomial in k and are computable in time polynomial in k. The domain of f_k is denoted by $D_k = \{0,1\}^{n_k}$.*[3]

Definition 2 *Let $f = (f_k)_{k=1}^{\infty}$ be a polynomial time function ensemble. Then f is* one way function *if for any polynomial time algorithm AL, and for all but finitely many k's, the probability that $AL(f_k(x)) \in f_k^{-1}(f_k(x))$ for $x \in_{\mathcal{U}} D_k$ is negligible (i.e., asymptotically smaller than $|x|^{-c}$ for any $c > 0$) .*

1.2 Previous work

Quite a few attempts to construct one way functions - typically in the context of public key cryptosystems - are based on the hardness of variants of the subset sum problem. However, algorithmic attacks which compute the inverses of the suggested functions in expected polynomial time were later found for all these attempts.

The public key cryptosystem of Merkle and Hellman [10] uses an easy to solve variant of the subset sum problem, in which the input sequence is super increasing, which is transformed to a sequence in which the super increasing structure is concealed. This cryptosystem was first broken by Shamir in [12], and subsequently more sophisticated variants of it were broken too [2].

Super increasing sequences are a special case of *low density* instances of the subset sum problem. These low density instances were also solved efficiently [8; 1; 3]. A comprehensive survey of these methods and of the corresponding attacks can be found in [11].

[3] For definiteness, inputs whose length ℓ is different from m_k for all k are mapped to 1^{ℓ}.

1.3 Contribution

The basic variant of the self masking technique replaces a (polynomial time computable) function f by a self masking version, denoted $[f]$, as follows: Let $y = f(x)$ for arbitrary x in the domain of f, and let $|x|$ denote the length of x. Then a self masked version $[y] = [f](x)$ is obtained by replacing two "critical" substrings, z_1 and z_2, of y, of length $|x|^{\Omega(1)}$, by $z_1 \oplus z_2$ [4]. Intuitively, z_1 and z_2 are critical in the sense that they are essential for computing $f^{-1}(y)$.

An immediate concern raised by this method is that it might significantly increase the number of preimages associated with the masked output value $[f](x)$, e.g. that $[f]^{-1}([f](x))$ may contain exponentially many preimages of $[f](x)$ even if $f^{-1}(f(x))$ contains only few elements. We cope with this difficulty by showing that, by carefully selecting the parameters of the transformation, this is not the case, and in fact that we can guarantee that, w.h.p., $[f]^{-1}([f](x)) = \{x\}$, i.e. $[f]$ is univalent.

We demonstrate this technique on functions associated with variants of the subset sum problem, which were widely used in the context of one way functions (see eg [10; 8; 7; 9]).

Organization of the paper. Section 2 introduces the self masked subset sum problem, and proves that this problem is NP hard.
Section 3 presents function ensembles associated with the self masked subset sum problem, and present conditions under which the resulted functions are univalent w.h.p.
Section 4 demonstrates that applying the self masking technique on super increasing instances of the subset sum problem produces function which cannot be inverted by the known attacks on cryptosystems based on super increasing sequences.
Section 5 extends this result by showing that applying the self masking technique on low density instances of the subset sum problem provides functions which cannot be inverted by the known algorithms for solving low density instances of the (unmasked) subset sum problem.
Section 6, which is only sketched in this version, discusses extension of the self masking technique to high density instances of the subset sum problem.
Finally, Section 7 summarizes the results of this paper and discusses possible extensions.

2 Subset sum with self masking

The subset sum problem of dimensions k and ℓ, to be denoted $SS(k, \ell)$, is defined as follows: Let $\mathcal{A}_{k,\ell} = \{(a_1, \ldots, a_k) : a_i \in [0, 2^\ell - 1]\}$. An input to $SS(k, \ell)$ is a pair (A, b), where $A \in \mathcal{A}_{k,\ell}$ and b is an additional integer. It is required to

[4] $z_1 \oplus z_2$ denotes bitwise XOR of the binary representations of z_1 and z_2; leading zeros are assumed when these representations are of different lengths.

decide if there is a binary vector $\alpha = (\alpha_1, \ldots, \alpha_k)$ s.t. $A\alpha^T = \sum \alpha_i a_i = b$.

We use the subset sum problem to define the following function ensemble $(f_{k,\ell})_{k,\ell \in \mathbb{N}}$: For each k and ℓ, $n_{k,\ell} = k(\ell+1)$, $m_{k,\ell} = \ell(k + \lceil log(k) \rceil)$. An input $x \in \{0,1\}^{n_{k,\ell}}$ represents a sequence $x = (A, \alpha) = ((a_1, \ldots, a_k), \alpha)$, where each a_i is encoded by ℓ bits, and α is a binary vector with k bits. $f_{k,\ell}(x) = y$ is given by

$$f_{k,\ell}(x) = f_{k,\ell}(A, \alpha) = (A, A\alpha^T) = y \ , \tag{1}$$

where $A\alpha^T < k2^\ell$ is encoded by $\ell \lceil log(k) \rceil$ bits.

Note that $f_{k,\ell}(x)$ is necessarily a solvable instance of the subset sum problem, and $f^{-1}_{k,\ell}(f_{k,\ell}(x))$ is the nonempty set of the solutions to this instance.

The self masking version of subset sum, denoted *self masked subset sum*, consists of two independent instances of the problem, which mask each other (so it actually uses the 2-dimensional subset sum problem [5]).

Specifically, the input is a triplet (A_1, A_2, b), where A_1 and A_2 are k dimensional vectors of positive integers, and b is s positive integer. It is needed to decide if there is a binary k-vector α and integers b_1, b_2, s.t.

$$A_1\alpha^T = b_1, \ A_2\alpha^T = b_2, \ \text{ and } \ b = b_1 \oplus b_2.$$

Lemma 1. *The self masked subset sum problem is NP-Hard.*

Proof. We prove the lemma by presenting a polynomial time reduction from the subset sum problem to the self-masked subset sum problem. Let (A, b) be an input to the subset sum of dimensions k and ℓ (for arbitrary k and ℓ). We reduce (A, b) to an input (C, D, e) to the self masked subset sum, where $C = A$ and D and e are defined as follows: Let $n = \lceil \log(\sum_{i=1}^{k} a_i) \rceil$. Then $D = 2^n C = (2^n a_1, \ldots, 2^n a_k)$, and $e = (2^n + 1)b$.

Since n is linear in the input length, the reduction can be performed in polynomial time. To prove its correctness, we need to show that there is a binary vector α satisfying $A\alpha^T = b$ if and only if there is a binary vector β satisfying $C\beta^T \oplus D\beta^T = e$.

Let $\alpha = (\alpha_1, \ldots, \alpha_k)$ be an arbitrary non-zero binary vector. Observe that in the binary representation of the integer $D\alpha^T$, the n least significant bits are all zeros, while in the binary representation of $C\alpha^T$ (with possible leading zeros), the only non-zero bits are among the n least significant bits. This implies that, in this case, the XOR operation coincides with integer addition, i.e.

$$C\alpha^T \oplus D\alpha^T = C\alpha^T + D\alpha^T = (2^n + 1)C\alpha^T = (2^n + 1)A\alpha^T.$$

We conclude that if $A\alpha^T = b$ for some α, then $C\alpha^T \oplus D\alpha^T = (2^n + 1)b = e$, and vice versa - if, for some β, $C\beta^T \oplus D\beta^T = e$, then $A\beta^T = e/(2^n + 1) = b$. This completes the correctness proof.

3 Function ensembles associated with self masked subset sum

Given k and ℓ, the function ensemble of dimensions k and ℓ associated with the self masked subset sum is denoted by $[f]_{k,\ell}$. An input x to $[f]_{k,\ell}$ is a triple (A_1, A_2, α), and

$$[f]_{k,\ell}(x) = [f]_{k,\ell}(A_1, A_2, \alpha) = \left(A_1, A_2, A_1\alpha^T \oplus A_2\alpha^T\right) \tag{2}$$

That is, in $[f]_{k,\ell}(x)$ the values of b_1 and b_2 mask each other by $b_1 \oplus b_2$.

Lemma 2. *Assume a uniform distribution on $\mathcal{A}_{k,\ell}$, and let $\alpha \in \{0,1\}^k \setminus \{0^k\}$. Then the random variable r_α on $\mathcal{A}_{k,\ell}$ defined by*

$$\forall A \in \mathcal{A}_{k,\ell}, \quad r_\alpha(A) = A\alpha^T (\mathrm{mod}\ 2^\ell).$$

defines the uniform distribution on $[0, 2^\ell - 1]$.

Proof. We need to show that for each integer $c \in [0, 2^l - 1]$, it holds that $Prob[r(A) = c] = 2^{-\ell}$. For this we assume WLOG that $\alpha_1 = 1$, and we let β be the vector obtained from α by setting α_1 to 0. Then $r(A) = A\alpha^T = A\beta^T + a_1$. Hence we get, using arithmetic modulus 2^ℓ:

$$\begin{aligned} Prob[r_\alpha(A) = c] &= \sum_{j \in [0, 2^\ell - 1]} \left(Prob[A\beta^T = j] \cdot Prob[a_1 = (c - j)]\right) \\ &= \left(\sum_{j \in [0, 2^\ell - 1]} (Prob[A\beta^T = j]\right) \cdot 2^{-\ell} \ = \ 2^{-\ell}, \end{aligned}$$

where the second equality holds since for all j, $Prob[a_1 = c - j \pmod{2^\ell}] = 2^{-\ell}$. □

Our proof uses the following variant of lemma 2

Lemma 3. *Assume a uniform distribution on $\mathcal{A}_{k,\ell}$, and let $\alpha, \beta \in \{0,1\}^k$ s.t. $\alpha \neq \beta$. Then the random variable $r_{\alpha,\beta}$ on $\mathcal{A}_{k,\ell}$ defined by*

$$\forall A \in \mathcal{A}_{k,\ell}, \quad r_{\alpha,\beta}(A) = [A\alpha^T (\mathrm{mod}\ 2^\ell)] \ \oplus \ [A\beta^T (\mathrm{mod}\ 2^\ell)]$$

defines the uniform distribution on $[0, 2^\ell - 1]$.

Proof. Assume WLOG that $\alpha_1 = 0$ and $\beta_1 = 1$. Let $c = (c_2, \ldots, c_k)$, where the c_i's are arbitrary elements in $[0, \ldots, 2^\ell - 1]$. Let $\mathcal{A}_c$ be the subset of all vectors $(a_1, \ldots, a_k) \in \mathcal{A}_{k,\ell}$ in which $a_2 = c_2, \ldots, a_k = c_k$. Then, on $\mathcal{A}_c$, $A\alpha^T$ is fixed (since $\alpha_1 = 0$ and hence it is independent of the value of a_1), and $A\beta^T$ mod (2^ℓ) distributes uniformly on $[1, \ldots, 2^\ell - 1]$ (since a_1 distributes uniformly in $[0, 2^\ell - 1]$). So the lemma holds for $\mathcal{A}_c$. The lemma follows by observing that $\mathcal{A}_{k,\ell}$ is a disjoint union of $\mathcal{A}_c$, where c varies over all $2^{\ell(k-1)}$ possible combinations of $(k-1)$ tuples.

Lemma 4. *Let* $(A_1, A_2) \in_{\mathcal{U}} [\mathcal{A}_{k,\ell}]^2$*, and let* $\alpha \in \{0,1\}^k$*. Then the probability that there exists* $\beta \in \{0,1\}^k, \beta \neq \alpha$*, s.t.*

$$A_1\alpha^T (\text{mod } 2^\ell) \oplus A_2\alpha^T (\text{mod } 2^\ell) = A_1\beta^T (\text{mod } 2^\ell) \oplus A_2\beta^T (\text{mod } 2^\ell) \qquad (3)$$

is at most $2^{k-\ell}$*.*

Proof. Fix A_1 for now. Let $\beta \in \{0,1\}^k, \beta \neq \alpha$ be given. Denote for brevity $A_1\alpha^T = b_1$ and $A_1\beta^T = b_2$. Then (3) can be written as: $b_1 \oplus A_2\alpha^T = b_2 \oplus A_2\beta^T$, which is equivalent to $b_1 \oplus b_2 = A_2\alpha^T \oplus A_2\beta^T$. By Lemma 3, the probabilty of this last equality to hold for a random A_2 is $2^{-\ell}$. The lemma for fixed A_1 follows by applying the union bound on all $\beta \in \{0,1\}^k \setminus \{\alpha\}$. Since A_1 was arbitrary, the lemma is proven.

As an immediate application of lemma 4 we get:

Corollary 1 *Let* c *be a positive constant. If* $\ell > k + c \log n_{k,\ell} = k + c \log(k(\ell+1))$*, the probability that two random inputs* x_1, x_2 *to* $f_{k,\ell}$ *satisfy* $f_{k,\ell}(x_1) = f_{k,\ell}(x_2)$ *is smaller than* $(n_{k,\ell})^{-c}$*.*

Note that the premises of Corollary 1 hold for almost all ℓ provided that $\ell \geq (1+\varepsilon)k$ for some fixed $\varepsilon > 0$ - i.e. for low density instances of the subset sum problem.

4 Subset sum with super increasing sequences

A sequence $A = (a_1, \ldots, a_k)$ is *super increasing* if:

$$\text{for } i = 2, \ldots k, \quad \sum_{j=1}^{i-1} a_j \ < \ a_i \ .$$

A subset sum instance (A, b) is easily solved in polynomial time when A is super increasing: start with an empty subset S, and at each stage add to S the largest element in A which is not yet in S, provided that the sum of the elements in S does not exceed b. Nevertheless, few cryptographic schemes are based on solving instances with super increasing sequences, by concealing their super increasing nature. [11] provides a detailed survey of these methods, and then describes the efficient attacks that eventually broke them. In this section we observe that these attacks must use a value which is hidden by the self masking technique, implying that the self masked version of subset sum with super increasing sequences are likely to be immune to these attacks.

The super increasing variant of subset sum of dimensions k and ℓ, denoted $SS^{si}(k,\ell)$, is defined by associating with each input sequence $A = (a_1, \ldots, a_k) \in \mathcal{A}_{k,\ell}$ a super increasing sequence $A^{si} = (a_1^{si} \ldots, a_k^{si})$, where $a_1^{si} = a_1, a_2^{si} = 2^\ell + a_2, a_3^{si} = 3 \cdot 2^\ell + a_3$, and in general $a_i^{si} = (2^{i-1} - 1)2^\ell + a_i$. It is easy to

check that A^{si} is super increasing. The functions $f^{si}_{k,\ell}$ are obtained from $f_{k,\ell}$ in Equation 1, by replacing $A\alpha^T$ by $A^{si}\alpha^T$:

$$f^{si}_{k,\ell}(x) = f^{si}_{k,\ell}(A,\alpha) = (A, A^{si}\alpha^T)$$

Since A^{si} is super increasing, inverting the function $f^{si}_{k,\ell}$ by finding the unique vector α satisfying $A^{si}\alpha^T = b$, as outlined above, is easy. We now argue that, for k and ℓ satisfying the premises of Corollary 1, the self masking version of $f^{si}_{k,\ell}$ is likely to be immune to this inversion method. For this, we observe that $f^{si}_{k,\ell}$ is univalent w.h.p.:

For all A and α it holds that $A\alpha^T(\bmod\ 2^\ell) = A^{si}\alpha^T(\bmod\ 2^\ell)$. Hence Lemma 4 remains valid if, in eq. (3), we replace $A_1(A_2)$ by $A^{si}_1(A^{si}_2$ resp.). and hence the following analogue of Corollary 1 for super increasing sequences holds.

Corollary 2 *If $\ell > k + c\log n_{k,\ell} = k + c\log(k(\ell+1))$, the probability that there are x_1, x_2 with $f^{si}_{k,\ell}(x_1) = f^{si}_{k,\ell}(x_2)$ is smaller than $(n_{k,\ell})^{-c}$.*

Let $\left[f^{si}\right]$ be the self masking version of f^{si}. Corollary 2 implies that if $\ell > k + c\log(n_{k,\ell})$, then w.h.p., $\left[f^{si}\right](A_1, A_2, \alpha) = \{(A_1, A_2, \alpha)\}$. In this scenario, inverting $\left[f^{si}\right](A_1, A_2, \alpha)$ is at least as hard as reconstructing the integers $b_1 = A^{si}_1\alpha$ and $b_2 = A^{si}_2\alpha$ from their xor $b = b_1 \oplus b_2$ and the sequences A_1, A_2. This last task appears to be be non-trivial.

Other variants based on super increasing sequences. As noted above, few cryptosystems are based on subset sum with super increasing sequences. We briefly survey them below (for a more comprehensive exposition we refer again to [11]).

The most known variant is due to Merkle and Hellman [10]: Given a super increasing sequence $A = (a_1, \ldots, a_n)$ and b, select relatively prime integers W, M, where $W < M$ and $M > b$, and then define $a'_i = Wa_i \pmod M, b' = Wb \pmod M$. The original super increasing sequence A is then replaced by a random permutation of $A' = (a'_1, \ldots, a'_n)$, and b is replaced by b'. The resulting sequence is not super increasing, and reconstructing the original super increasing sequence (when W and M are not given) is not straightforwards. This process can be iterated few times, yielding the multiply iterated Merkle Hellman system.
The first polynomial time algorithm which solves the original (singly iterated) Merkel Hellman system was given by Shamir in [12] (this attack assumes certain restrictions on the ratio between M and k, which are implied by properties of the associated cryptosystem). Shamir's attack was later followed by algorithms solving more sophisticated variants of such systems (eg [2]). These algorithms essentially reconstruct the original sequence A and b from the hidden versions A' and b', and in particular the value of b' must be given for applying these attacks. Since this value is hidden by our self masking technique, it appears that these attacks cannot be directly applied to the self masked variants of Merkle and Hellman systems, as well to their extensions.

5 The low density variant

The subset sum problem of dimensions k and ℓ, $SS(k,\ell)$, is said to be of low density if ℓ is larger than k. Polynomial time algorithms for inverting low density $f_{k,\ell}$ were first obtained in [1; 8]. These algorithms reduce the inversion of $f_{k,\ell}$ to finding a shortest vector in an integer lattice. A detailed survey of these algorithms and later improvements can be found in [11]. We briefly describe below the algorithm of [8], as described in [3].

Let $x = (A, \alpha)$ be an input to $f_{k,\ell}$, where $A = (a_1, \ldots, a_k) \in_{\mathcal{U}} \mathcal{A}_{k,\ell}$ and $\alpha = (\alpha_1, \ldots, \alpha_k) \in_{\mathcal{U}} \{0,1\}^k$. Let further $y = f_{k,\ell}(x) = (A, b)$, where $b = A\alpha^T$. The algorithm of [8] reduces the computation of $f^{-1}_{k,\ell}(y)$ to the problem of finding a shortest vector in the $k+1$ dimensional integer lattice $L(y) = L(A,b)$ defined by the basis

$$\begin{aligned} v_1 &= (1, 0, \ldots, 0, -Ka_1), \\ v_2 &= (0, 1, 0, \ldots, 0, -Ka_2) \\ &\cdots \\ v_k &= (0, \ldots, 0, 1, -Ka_k), \\ v_{k+1} &= (0, \ldots, 0, Kb). \end{aligned}$$

where K is any integer larger than $\sqrt{k}$. Given that basis, each binary vector $\beta = (\beta_1, \ldots, \beta_k)$ is mapped to a lattice-vector $w(\beta)$ given by

$$w(\beta) = \sum \beta_i v_i + v_{k+1} = \left(\beta_1, \beta_2, \ldots, \beta_k, b - A\beta^T\right);$$

Observe that $A\beta^T = b$ iff $w(\beta) = (\beta_1, \ldots, \beta_k, 0)$. The main ingredient in the correctness proof of the algorithm of [8] is showing that if $k < 1.54725\ell$, then w.h.p. α is the only vector satisfying $A\alpha^T = b$, and $w(\alpha)$ is the unique shortest vector in $L(y)$. [3] uses a similar proof technique, but reduces the vector $y = (A, b)$ to a different lattice $L'(y)$, which enables to improve the required density to $k < 1.0639\ell$. For our sake it is sufficient to note that the use of the sum $b = A\alpha^T$ in the definition of the basis vector v_{k+1} is crucial in the above reductions.

Consider now the self masking function $[f]_{k,\ell}$

$$[f]_{k,\ell}(A_1, A_2, \alpha) = (A_1, A_2, b_1 \oplus b_2), \quad \text{where} \quad b_1 = A_1\alpha^T, b_2 = A_2\alpha^T.$$

In order to compute the inverse of $[f]_{k,\ell}(A_1, A_2, \alpha) = (A_1, A_2, b_1 \oplus b_2)$ it is necessary to compute from $(A_1, A_2, b_1 \oplus b_2)$ two integers b'_1 and b'_2 s.t.: (i) $b'_1 \oplus b'_2 = b_1 \oplus b_2$, and (ii) for some binary vector α' it holds that $b'_1 = A_1\alpha'^T, b'_2 = A_2\alpha'^T$.

By Theorem 2, if $\ell > (1+\varepsilon)k$ for some positive ε, then for almost all k, $[f]_{k,\ell}$ is univalent w.h.p., meaning that w.h.p. α' must equal α. Thus in this case solving the self masked version of $SS(k,\ell)$ is at least as hard as finding the unique b_1, b_2 s.t. $b_1 \oplus b_2 = b$, from A_1, A_2 and $b_1 \oplus b_2$.

6 The high density variant

The subset sum problem of dimensions k and ℓ, is said to be of high density if $k > \ell$. In this case the corresponding self masking function $[f]_{k,\ell}$ is w.h.p.

not univalent. Nevertheless, self masking functions which are univalent w.h.p. can be obtained also for each high density variant of the subset sum problem, $SS(k,\ell)$, by using a d dimensional self masking, $\left[f^{(d)}\right]$, where $d-1 > \frac{k+c\log(n_k)}{\ell}$, as sketched below.

An input x to $f^{(d)}_{k,\ell}$ contains d independent instances of the problem, i.e. $x = (A_1, A_2, \ldots, A_d, \alpha)$, and the self masked version of $f^{(d)}$ is

$$\left[f^{(d)}\right]_{k,\ell}(x) = \left[f^{(d)}\right]_{k,\ell}(A_1, A_2, \ldots, A_d, \alpha) =$$
$$\left(A_1, A_2, \ldots, A_d, A_1\alpha^T \oplus A_2\alpha^T, A_1\alpha^T \oplus A_3\alpha^T, \ldots, A_1\alpha^T \oplus A_d\alpha^T\right),$$

Lemma 5. *Let $A_1, \ldots, A_d$ be mutually independent random vectors from $\mathcal{A}_{k,\ell}$, and let $c_2, \ldots, c_d$ be arbitrary integers. Then the probability that there exists a vector $\alpha \in \{0,1\}^k$ and integers $b_1, b_2, \ldots, b_d$ s.t. for each $i \in \{2, \ldots, d\}$ it holds that $c_i = b_1 \oplus b_i$ and $A_i \cdot \alpha^T = b_i$, is at most $2^{k-(d-1)\ell}$.*

Proof. First we note that $A_i \cdot \alpha^T = b_i$ iff $A_1\alpha^T \oplus A_i\alpha^T = c_i,\ i = 2, \ldots, d$. As in the proof of Lemma 4, we first fix A_1. Then we get that for a given α, and for each $i \in \{2, \ldots, d\}$:

$$Prob[A_1\alpha^T \oplus A_i\alpha^T = c_i \mid A_1, \alpha] \leq 2^{-\ell}.$$

Since the A_i are mutually independent, we get that for a fixed α the probability that this equality holds for all $i \in \{2, \ldots, d\}$ is at most $2^{-\ell(d-1)}$. The lemma for a fixed A_1 follows by the union bound. Since A_1 is arbitrary, the lemma holds. □

Similarly to the one dimensional case, Lemma 5 implies:

Corollary 3 *If $(d-1)\ell > k + c\log n_{k,\ell}$, the probability that two independent random inputs, x_1, x_2, to $f^{(d)}_{k,\ell}$, satisfy $f^{(d)}_{k,\ell}(x_1) = f^{(d)}_{k,\ell}(x_2)$, is smaller than $(n_{k,\ell})^{-c}$.*

7 Concluding Remarks

In this paper we introduced the self masking technique, which aims at making the inversion of various polynomial time computable functions harder (see the informal idea sketch suggested in [4]). In the basic version, $[f](x)$, the self masking version of $f(x)$, replaces two "critical" parts of $f(x)$ by their bitwise xor. A straight forwards approach for solving the resulted computational task of computing $[f]^{-1}(x)$ requires examining numerous possible pairs of candidates for the xored parts. Thus inversion is hard unless there is a way to bypass this straight forwards approach in an efficient way. Specifically, this task is likely to be difficult if, w.h.p., computing the inverse of $[f](x)$ requires to reconstruct the original critical parts from their bitwise xor, i.e. if $[f]^{-1}([f](x)) = \{x\}$. As will be discussed in the full version the invesion task remains hard when the univalence requirement is relaxed to the case when $[f]^{-1}([f](x))$ is of small cardinality. We note that, apriori, a self masking $[f]$ of f could be hard to invert even if f can be inverted in polynomial time.

We applied this technique on well studied functions based on variants of the subset sum problem, where the critical parts were the sums of two independent solvable inputs for this problem. As we discussed, these sums are indeed critical for the polynomial time inversion algorithms surveyed in [11]. Thus it appears that these inversion algorithms cannot be directly applied to the self masked versions of subset sum problems presented in this paper.

Possible extensions. It is interesting if the self masking technique can be shown to harden the inversion of other polynomial time computable functions.

The practicality of the self masking technique depends heavily on the hardness to reconstruct the self masked parts. Ideally we would like it to imitate xor with one time pad. A promising way to approach this goal is to use instances from different functions, e.g. to mask a critical part of a function defined by an instance to the subset sum problem by a critical part of an instance to a different problem.

References

1. Brickell, E.F.: Solving low density knapsacks. In: Advances in cryptology. pp. 25–37. Springer (1984)
2. Brickell, E.F.: Breaking iterated knapsacks. In: Proceedings of CRYPTO 84 on Advances in Cryptology. p. 342–358. Springer-Verlag, Berlin, Heidelberg (1985)
3. Coster, M.J., Joux, A., LaMacchia, B.A., Odlyzko, A.M., Schnorr, C.P., Stern, J.: Improved low-density subset sum algorithms. Computational complexity **2**(2), 111–128 (1992)
4. Dolev, H., Dolev, S.: Toward provable one way functions. IACR Cryptol. ePrint Arch. p. 1358 (2020), https://eprint.iacr.org/2020/1358
5. Emiris, I.Z., Karasoulou, A., Tzovas, C.: Approximating multidimensional subset sum and minkowski decomposition of polygons. Mathematics in Computer Science **11**(1), 35–48 (2017)
6. Håstad, J., Impagliazzo, R., Levin, L.A., Luby, M.: A pseudorandom generator from any one-way function. SIAM Journal of Computing **28**, 12–24 (1999)
7. Impagliazzo, R., Naor, M.: Efficient cryptographic schemes provably as secure as subset sum. Journal of cryptology **9**(4), 199–216 (1996)
8. Lagarias, J.C., Odlyzko, A.M.: Solving low-density subset sum problems. Journal of the ACM (JACM) **32**(1), 229–246 (1985)
9. Lyubashevsky, V., Palacio, A., Segev, G.: Public-key cryptographic primitives provably as secure as subset sum. In: Theory of Cryptography Conference. pp. 382–400. Springer (2010)
10. Merkle, R., Hellman, M.: Hiding information and signatures in trapdoor knapsacks. IEEE transactions on Information Theory **24**(5), 525–530 (1978)
11. Odlyzko, A.M.: The rise and fall of knapsack cryptosystems. In: In Cryptology and Computational Number Theory. pp. 75–88. A.M.S (1990)

12. Shamir, A.: A polynomial-time algorithm for breaking the basic merkle - hellman cryptosystem. IEEE Transactions on Information Theory **30**(5), 699–704 (1984). https://doi.org/10.1109/TIT.1984.1056964

Swarming with (Visual) Secret (Shared) Mission

(Preliminary Version)

Shlomi Dolev, Alexander Fok, and Michael Segal

Ben Gurion University of the Negev, Beer Sheba, Israel

Abstract. A swarm of UAVs typically moves in a coordinated manner to perform a mission, for example, searching for a target, specified by an image. On one hand, mission details must be conveyed to the swarm members. On the other hand, the mission information is sensitive information that should not be exposed to a potential adversary, say, in case some of the drones accidentally fall into the adversary's hands. Our solution is based on VES (Visual Encryption Schemes) and projections of visual bit maps rather than (quadratic) messages exchange in implementing Secure Multi Party Computation. We suggest a perfect-information-theoretic secure solution for this problem.

Keywords: Visual Secret Sharing · Visual Encryption Scheme · Secure Multi-Party Computation · Target Image Protection · UAV Swarm · Information-Theoretic Secure Solution

1 Introduction

A swarm of n UAVs typically moves in a coordinated manner to perform a mission, for example, searching for a target identified by an image. The target image is critical piece of information that has to be accessible to every drone in the swarm for mission success. However, the target image is sensitive information that should not be exposed to a potential adversary in case of any faulty scenario, for instance, if an adversary takes control over some of the drones. The suggested solution allows a swarm of drones to make a collaborative target image matching. This solution is resilient against a semi-honest adversary. In the semi-honest setting, the parties have to follow the exact prescribed protocol in the real world. This implies that they cannot change their inputs or outputs. As opposite, a malicious (Byzantine) adversary may arbitrarily deviate from the protocol execution in its attempt to actively cheat. It is also resilient against a malicious (active) adversary that controls less than one third of the amount of drones. This solution is based on Naor and Shamir's Visual Encryption Scheme (VES) for mission target image hiding, see [20]. The target image is sliced into n shares that are provided to the drones in the swarm. For a chosen parameter k, any k shares make it possible to reveal the original image while $k-1$ shares are not enough to expose any information on the image. During the mission, drones take candidate picture images from potential geographic regions and try to match them to the target image. The drones employ Secure Multi

Party Computation (MPC), see [13,1,2,11]. The use of MPC preserves the main security requirement during the candidate image matching evaluation; in every computation, any subset of up to $k-1$ drones has no information on the target image.

Security analysis against different adversary types and threat scenarios is also shown.

2 Background and Related Work

There are several approaches to cope with a secure image matching task, in distributed systems. For example, systems similar to ad hoc swarm of drones, has been considered in the past. The following section discusses several of these possible approaches.

Geometrical Image Split Approach. In this approach, the target image is split to neighborhood pieces. Each piece contains complete information about the image piece, bounded by some geometrical figure, e.g. rectangular pieces. An advantage of this approach is that most of the known image matching solutions can be applied using the image slices matching, some examples are proposed in [22,19,10,16]. The main disadvantage here is the poor security of this naive image split approach. Single image pieces can contain a key image element, e.g., part of or the entire license plate number, in the case of a car image. Such pieces, falling into adversary hands, would reveal significant information about the target.

Holographic Information Presentation Approach. In this approach, the target image is split into frequencies, leveraging holographic coding as proposed in [5,6]. The appealing property of holographic image slices encoding is that every slice contains (blurred) information about the whole image. Thus, it can be thought of as a blur image of the entire target image, or Fourier encoding in the frequency domain rather than actual pixel values. While holographic techniques are very effective in gradual data transfer and error correction recovery based on partial information, their security level is weak, allowing an adversary to gain information on the whole target image given control of some of the drones. Depending on the specific holographic coding approach, its own effectiveness can be measured by experiments utilizing image processing techniques, as proposed in [8]. The least square criterion is commonly used, a criterion that does not necessarily reflect the human image perception.

Projection Based, non Threshold Approach. Every drone projects its share of the target image to an external "safe media" (e.g., a specific location in dusted air) accessible by all, but cannot be broken into the individual projections. Alternatively, the projections might be accessible to the "secured swarm leader". The "secured swarm leader" tries to match the projected target splits with the acquired target candidate image. Once matched, the target is acquired. In this approach, the original target image is considered as a piece of sensitive, binary data that can be secretly shared among swarm drones, utilizing some known secret sharing schemes, see, e.g. [23]. The disadvantage of this approach is that during the target matching computation, the security depends on the security

abilities of the "secured swarm leader". Once an adversary compromises the swarm leader or affects the distributed leader's election mechanism, the whole scheme is compromised.

Our contribution. In this work we propose an integral solution for the drone swarming and hiding target image. Our solution relies on Visual Encryption Scheme and Secured Multi Party Computation (MPC). The Hiding Target Image Problem solution and VES usage are described in Section 3.3 Hiding Target Image Problem. In Section 3.4 we analyze Candidate Image Injection Problem, where the majority of the swarm drones should agree on the candidate image taking picture parameters (such as location, angle of taking image, resolution, etc.) to obtain identical (or almost identical) candidate images in order to apply the target image matching algorithm. The Secure Image Matching Problem is analyzed in Section 3.5. Within this section, the swarm of drones should make a collaborative decision regarding the candidate and target images to match, while keeping the target image secure. As it follows from the Section 4, the above-mentioned solution is information-theoretic secure. In Section 5 we propose solution variant that is based on VES and slides projection. This solution is much more efficient for some setups.

3 Our Approach

Below are definitions that are utilized within the scheme.

3.1 Definitions

Let T be a target image, and T_i be i's target image slide, created in accordance with VES technique, as described in Section 3.3. We denote by $[T]$ a set of all image slides $T_1, T_2, \ldots, T_n$. Let $P = \{P1, \ldots, P_n\}$ be a set of n drones. Assuming that drones perform some computation, let X_i be the results of P_i's computation, and $[X] = \{X_1, \ldots, X_n\}$ be a set of all the calculation results of the drones. Similarly, C_i represents a candidate image taken by drone P_i, and $\overline{C}_i$, represents an inverted candidate image C_i where black pixels are set to white and white pixels are set to black.

3.2 Solution Description

There is a swarm of n identical drones, every drone has a processing unit, local storage and camera to take images. We do not require any particular computational power or minimal security specification on the drone hardware. For the sake of simplicity, it can be assumed that the images consist of black and white

pixels and each pixel is treated separately.[1] The proposed solution consists of two phases:

Initialization phase – Firstly of all, the swarm of drones is initialized and the drones receive all the required information to accomplish the mission. It can be assumed that the swarm initialization is done in trusted environment, where sensitive information is protected by various means that are out of scope of this work.

Operation phase – After the **initialization phase**, the swarm drones travel in a coordinated manner toward the target searching area. Then they start acquiring candidate images trying to match them to the original target image (that they do not posses due to security considerations). In the proposed solution, the target image is never reconstructed during the swarm operation (in solution described in Section 5, the participants reconstruct and immediately forget the reconstructed image). This ensures the high privacy of the target image.

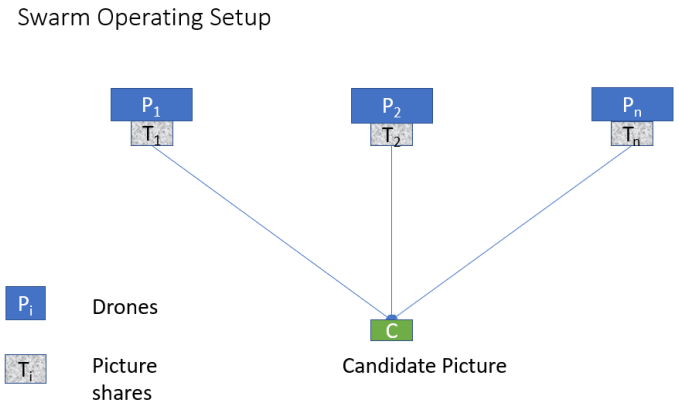


Fig. 1. UAVs Swarm Operational Setup

In our solution, we identified and addressed the following problems.

3.3 Hiding Target Image Problem

The target image is sensitive information that should not be revealed by an adversary during the swarm **operation phase**. The proposed solution should

[1] One may consider RGB images by handling, the matrix corresponding to R, with values 0 when the pixel has no red component and 1 otherwise. Similarly, to green and blue components of the pixel, thus matching three binary matrices instead of one. More sophisticated schemes for fine tuned colors can be supported by adding more matrices.

be resilient against an adversary that controls less than k drones from the swarm. In our solution, we use VES to share the target image to n slides. Shared slides are distributed among the swarm drones. By definition of VES, to recover a target image, at least k drones are required.

VES Usage

Background. There are various Visual Encryption Schemes (VES) proposed by researchers. Shamir and Naor in [20] proposed a VES based on pixel transparency sharing. They define k out of n scheme that works as follows. The target image is encoded into n slides. At least k slides are required to recover the original image. The image slide (or share) consists of the pixel subpixels (shares) that are populated as following. Target image T consists of black (1) and white (0) pixels. Every image pixel is treated and encoded separately. There are two collections of $n \times m$ matrices C_0 and C_1. To encode a white pixel, a matrix from C_0 is randomly drawn. To encode a black pixel, a matrix from C_1 is randomly drawn. Every row from the chosen matrix is the pixel share. It defines the color of m pixels in each one of the n slides.

Example: $k = 2$**.** Let C_0 and C_1 be two sets of all matrices obtained by permuting the columns of matrices as described below:

$$C_0 = \left\{ \text{Matrices obtained by permuting of the columns of } \begin{bmatrix} 1 & 0 \dots 0 \\ 1 & 0 \dots 0 \\ \dots & \\ 1 & 0 \dots 0 \end{bmatrix} \right\}$$

$$C_1 = \left\{ \text{Matrices obtained by permuting of the columns of } \begin{bmatrix} 1 & 0 \dots 0 \\ 0 & 1 \dots 0 \\ \dots & \\ 0 & 0 \dots 1 \end{bmatrix} \right\}$$

Any single share in either C_0 *or* C_1 is a random collection of one black and $n-1$ white pixels. Any two shares of a white pixel have a combined Hamming weight of 1. Any two shares of a black pixel have a combined Hamming weight of 2. To avoid a distortion of the aspect ratio of the target image, it is convenient to represent the shares as two dimensional arrays $d \times d$.

For algorithm calculation experiments, we define VES scheme with pixel share options as shown on Figure 2, that fits 2 out of 2 VES scheme. This scheme works for our solution since we do not have to keep visual aspect ratio of the VES slides (avoiding images distortion), like authors are doing in original VES paper [20]. Every target image pixel is randomly given value from left or right table. For white pixel, the same share is chosen. For black pixel - mirror share is chosen.

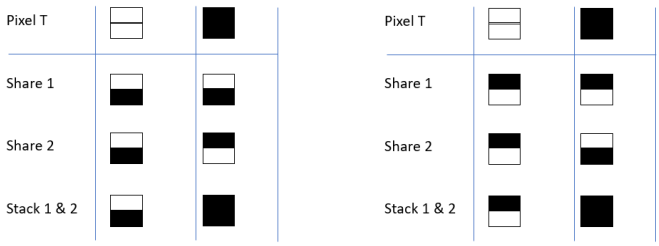


Fig. 2. Pixel Share Options

3.4 Candidate Image Injection Problem

During the swarm **operation phase**, while searching for the target, drones take images and compare them to the given target image. An adversary can perform various attacks on the swarm, resulting in the adversary taking partial or complete control over some of the swarm drones, gaining them access to the drone local storage. With this access, the adversary can eavesdrop on the drones communication, and send valid or invalid protocol messages. It is possible that an adversary controlled drone can inject images of his choice, triggering image matching computation on a malicious image, leading to false matching results and swarm action activation. Moreover, controlling a small number of drones allows an adversary to perform Image Dictionary Attack as described below in Section 4. To cope with the Candidate Image Injection Problem, the drones can agree on the image taking parameters, like coordinates, view angle, resolution, etc. Such agreement allows drones to take identical (or almost) identical images to match them with the target image. The drones can use Byzantine Agreement Solution to agree on the image taking parameters, such as coordinates. The Byzantine Agreement solution is resistant against 1/3 of the malicious drones.

3.5 Secure Image Matching Problem with no Projection

Once the agreed candidate image is captured by all swarm drones, the drones compare it to the target image slide T_i they posses. This comparison operation is done locally by each drone. The results of the comparison operation bear target image slide T_i information, thus if just shared with other drones, allow an adversary to expose an information about the target image. Moreover, given the drones local comparison results, the swarm should make a collaborative decision regarding probability of the candidate image target match. To handle the above challenges, we use MPC.

Secure Multi Party Computation. Secure Multi Party Computation maturity is increasing, evolving in its wide usage in different application domains. For example – [11,21], where computation should be done on the inputs, while

keeping the inputs private from computing nodes. Various MPC settings and performance optimizations are extensively investigated and proposed by various researches, [9,7,1,3,13,9,11,17,12], just to mention some. In our solution, drones keep the following information: T_i, C, x_i. We consider x_i, the result of local comparisons of the slide T_i and the candidate image C_i, as private data since it bears information about VES slide T_i. Thus, x_i should not be revealed to other computation nodes. This is the reason why in our algorithm x_i is secretly shared among drones, preventing an adversary controlled drone from revealing any information about x_i. It is worth to note that proposed local calculation step, allows the following MPC computation to rely on additions only, thus reducing overall communication cost.

Primitive Operators. We use the following operators to describe the solution algorithm.
AND$_\mathbf{m}$ – matrix logical **AND** operator.
First it replicates its second argument value from 1 to m pixels and then applies the logical **AND** on m pixels of both arguments. m is the VES parameter defining number of subpixels representing target image pixel in the slide. The result of **AND$_\mathbf{m}$** calculation on single pixel is bitmap of size $m \times m$.
ThresholdOr Operator – distributed pixel **OR** operator.
In the original Naor and Shamir VES scheme, in order to unveil the original image pixels, transparent slides are laid on each other. In the resulting picture, white pixels look grey, and black pixels look darker (or completely black – depending on VES parameters). Mathematically it means that the Hamming Weight of correspondent pixels should be greater than the threshold t, to interpret the pixel as black, otherwise the pixel is interpreted as white. In our solution we require operating on precise pixel white and black values. To align our solution to the VES scheme, ThresholdOr is introduced. ThresholdOr operates on every target image pixel.

The result of ThresholdOr$([x], t)$ is matrix $Y_{n\times n}$ such that :

$$Y[k,j] = \begin{cases} 0, \text{ if } \sum_{i=1}^{n} x_i[k,j] \leq t \\ 1, \text{ otherwise.} \end{cases}$$

One of the parameters of VES is the number of pixels in each transparency (slide) representing a single target image pixel – m. The above ThresholdOr operator definition is valid for $m = 1$. It can be shown that ThresholdOr calculates Hamming Weight of the subpixels representing the target image pixel in the image shares. Thus, it can be generalized for VES scheme with parameter $m > 1$.

Putting it All Together Refer to Algorithm 1 for the complete Swarming and Hiding Target Image Algorithm.
At the beginning, the swarm is initialized and, once reaching the target search area, the drones start acquiring the candidate images to match. The swarm

drones start taking candidate images based upon the agreed image taking parameters, while ensuring that the images originate from benign drones and are (at least almost) identical. When drones posses valid candidate images C_i, they use $\mathbf{AND_m}$ operator to compare the local target image share T_i with C_i and $\overline{C}_i$ (Line 21). We denote the local comparison results x_i and $\overline{x}_i$ correspondingly. In Line 25 MPC is used to match the image in a secure and private way, as follows.

INPUT-MPC. x and $\overline{x}$ are secretly shared to other drones, including themselves (Line 1).
MPC1. The operator ThresholdOr$(x_1, x_2, \ldots x_n)$ is applied to shares of x_i and $\overline{x}_i$, resulting in pixel matrices y and $\overline{y}$ distributed shares (Line 3).
MPC2. Drones calculate the *match_pixel_count* (Line 5).
MPC3. Images matching decisions are calculated based on *match_pixel_count* and parameter R (Line 9).

At the end of the *MPC* computation, all drones have a local decision about the images matching. All benign drones have the same decision.

4 Analysis

First, we explain the correctness of our scheme.

Does C match T? To answer this question, it is necessary to answer it first for every pixel in C and T: whether the pixel $T[i,j]$ is equal to pixel $C[i,j]$? The basic calculation for pixels comparison looks like this: The AND_m is applied on all m subpixels representing pixel $T[i,j]$ in every slide. The result of AND_m $(x[i,j])$ is either white or black pixel. Then ThresholdOr counts, in a distributed way, white and black subpixels in all slides and results in the matrix Y of the target image size where each $Y[i,j]$ is equal to:

1. 1 (black) if the number of corresponding black subpixels in the shares is more than the VES threshold t.
2. 0 (white) if the number of corresponding black subpixels in the shares is less than or equal to the VES threshold t.

An example calculation for $T[i,j] = 1, C[i,j] = 1$ is shown on Figure 2. Here 2 out of 2 VES is assumed, $t = 2$ and $m = 2$.
For two pixels we have four possible combinations, as shown in the Table 1 below. The only available data for the drone performing the images match calculation is $C[i,j]$ and $Y[i,j]$. Due to the security requirement, $T[i,j]$ is not available. Given the data from the Table 1, we can conclude deterministic decisions for cases 2 and 4 only. To handle cases 1 and 3, we perform additional calculations where $\overline{C}_i$ (the inverted C_i) and $\overline{Y}$ are calculated. As shown in the Table 2 below, with the information provided by this additional calculation, we can conclude deterministic decisions for cases 1 and 3. In the Algorithm 1, we count matching black pixels with Y and matching white pixels with $\overline{Y}$, which forms the final

Algorithm 1: Swarming and Hiding Target Image Algorithm - with MPC

```
   // Target image is delivered to dealer
   Input: Target image T, match pixels threshold R, VES threshold t
   Output: Swarm target image match decision
   // MPC: Inputs x_i and x̄_i
1  function MPCCalc([x_i], [x̄_i]):
2      for each P_i in [P] do
3          P_i --secret share x_i and x̄_i--> P_j // secret share x_i and x̄_i
4      Y = ThresholdOr([x], t)
5      Ȳ = ThresholdOr([x̄], t)
       // Calculate match image pixels count
6      match_pixel_count = 0
7      for Y[i,j] in Y AND Ȳ[i,j] in Ȳ do
8          if Y[i,j] == 1 OR Ȳ[i,j] == 1 then
9              match_pixel_count+ = 1
       // Report the candidate image matching result
10     if match_pixel_count > R then
11         return True
12     else
13         return False
14
15
16 function main():
       // 1. Swarm initialization
       // Dealer calculates
17     [T] = VES(T)
       // Dealer distributes to drones the image slides
18     for each T_i in [T] do
19         (T_i) --send to--> P_i
       // 2. Swarm operation
20     while True do
           // Each Drone P_i takes candidate image C_i
21         C_i = TakeAgreedCandidateImage()
           // Drones local calculation
           // Each Drone P_i compares candidate image with their slides
22         for each P_i in [P] do
23             x_i = T_i AND_m C_i
24             x̄_i = T_i AND_m C̄_i
           // Drones perform MPC computation
25         report MPCCalc([x_i],[x̄_i])
```

Case Number	$T[i,j]$	$C[i,j]$	$Y[i,j]$	$T[i,j]$ ==? $C[i,j]$
1	0	0	0	YES/NO
2	0	1	0	NO
3	1	0	0	YES/NO
4	1	1	1	YES

Table 1. Table with all four possible calculation combinations

Case Number	$T[i,j]$	$\overline{C}_i[i,j]$	$\overline{Y}[i,j]$	$T[i,j]$ ==? $\overline{C}[i,j]$
1	0	0	0	YES
2	0	1	0	Determined by Table 1
3	1	0	0	NO
4	1	1	1	Determined by Table 1

Table 2. Table with all four possible calculation combinations

images match decision.

Now, we deal with image matching precision.

How accurate is our answer to the question – does C match T? The image matching precision of the solution depends on several parameters. The image matching solution relies on the individual pixels matching in C and T. We interpret C and T as binary matrices with element values 0 or 1, of the same size. We introduce the algorithm parameter R – the number of different pixels in C and T. The solution precision can be modeled by the precision model of binary classifiers. Small R can lead to a poor images match – causing high false negative (FN) result rate. Large R can cause high false positive (FP) result rate. One can tune R according to the desired FN and FP trade-off for the specific setting. An additional approach for image matching precision improvement is applying the algorithm multiple times – starting with low resolution images T *and* C comparison and gradually increasing the images resolution. When images are similar, such approximation leads to exact images matching for low resolution images, meaning that, for the majority of pixels, $T[i,j] = C[i,j] = 1$ or $T[i,j] = C[i,j] = 0$. Algorithm parameter D can be defined as an image matching precision threshold. Parameters R and D should be tuned empirically for the specific operational environment and image settings.

Next, we analyze the security of the proposed solution.

Target Image Privacy The target image is manipulated and managed in various forms during the solution. We prove that all data that contains target image information is kept safe and an adversary that controls less than k drones can not reveal any information about the target image – at any computational step.

Target Image T is used to initiate the swarm in safe environment. The target image is neither stored nor transmitted on any drone media, thus is not available to any adversary attack. ■

Target Image Slides T_i are created with Visual Encryption Scheme. VES is information-theoretic secure. It means that any combination of less than k image slides would reveal no information about the original target image T. In all algorithm computation steps, image slides T_i are kept locally by the corresponding drones. Therefore, an adversary has no opportunity to reveal any information about the target image, unless he controls at least k drones. ■

Secure Image Matching The Secure Image Matching computation is performed by MPC engine, see Line 1 of the algorithm. The only information about the target image is kept locally by drones in VES shares. The results of the VES shares comparison with the candidate image are passed to the MPC engine and secretly shared to all the drones for computation. Thus the security of the whole solution is bounded by the chosen security setting of the MPC engine.

MPC Security Guarantees MPC security level depends on settings and available resources. The usage of a broadcast communication channel allows MPC to tolerate up to $n/2$ Byzantine drones, but assuring the reliability of a broadcast channel can be challenging for the noisy swarm operating environment. Thus, in our protocol, we assume the availability of secure communication channels among the swarm drones. We make no assumptions on the drones computation power, or their hardware security hardening requirements. In these MPC settings, the proposed solution is information-theoretic secure:

- If an adversary is able to eavesdrop communication of up to $t < n/2$ drones.
- If an adversary controls up to $t < n/3$ drones – meaning the t byzantine drones can deviate from the protocol by either sending protocol valid messages or even completely different messages, while collaborating to coordinate the attack.

Image Dictionary Attack. One of the simple, while dangerous attack scenarios that our solution should be resilient against is Image Dictionary Attack. Applying this kind of attack might allow an adversary to reveal the target image. The adversary can act as following:

- Adversary maintains a bank of high value target images – images dictionary.
- During the swarm operation, adversary acquires control over some of the swarm drones.
- Adversary feeds images from the images dictionary to the controlled drones – in order to trigger a swarm image matching operation.
- When a target image is matched by the swarm, the swarm performs a target match action.

The adversary can utilize side channel eavesdropping techniques to detect swarm target image matching, without needing to actually trigger the swarm action on the target image. It can, for example, analyze internal swarm wireless communication channels, swarm communication patterns with command and control center (C2C), etc. It is clear that in case of target image matching the communication patterns on these channels will be different compared to regular communication. For example, swarm will notify C2C about the target matching and require its authorization to act. Despite the fact that the communication channels might be encrypted and the adversary would not be able to decrypt the communicated messages or to understand the exact dialogue, by analyzing the communication metadata, the adversary might be able to detect different communication patterns like target image matching. The adversary here reached two goals:

- Revealed the swarm target image.
- Caused the swarm faulty action on the target image. Assuming the swarm has limited physical resources to perform an action on a target, e.g., limited number of bombs, the adversary can drain the swarm resources, thus limiting its ability to act on the real target.

In our solution we have several countermeasures against an adversary attempt to carry the Image Dictionary Attack. As described in Candidate image Consensus, we require the drones to reach a majority consensus to trigger the image matching activity. It can be reached by solving Approximate Byzantine Agreement detailed in Section 3.4.

Finally, we conclude with algorithm performance.

Algorithm Performance. While analyzing the solution performance, we focus on calculations performed at the swarm operation phase. The calculations required for the swarm initialization are done only once in the trusted environment. It is done as pre-processing enabling the swarm operation while on mission, thus its performance is less important for overall solution complexity.

Operational Phase Calculations. The main solution calculation consists of two steps: candidate image consensus (Section 3.4) and secure candidate image matching (Section 3.5).

Candidate Image Agreement Performance. One of the important metrics of Byzantine Agreement solution is the communication complexity. It is known that quadratic communication is required for worst case with resilience against at most $n/3$ malicious drones.

MPC Performance Background. The current solution is not linked to a specific MPC protocol. Since the solution does not depend on MPC implementation specifics, we decided to rely on known MPC protocols and not to develop a custom solution. Any known MPC construction can be applied,

for example: Goldreich-Micali-Wigderson (GMW) [12], Ben-Or, Goldwasser, and Wigderson (BGW) [4] or Beaver-Micali-Rogaway (BMR) protocol [3]. Performance wise, BGW and BMR protocols guarantee any function evaluation in a constant number of rounds, as shown in [17]. The actual MPC cost is proportional to the circuit depth multiplied by the cost of secret multiplication. The multiplication itself requires several rounds of interaction. Thus, the main performance parameters of an MPC based solution are circle depth and number of multiplication gates. As proposed by researchers in [18] design, that is based on Bristol notation, sha256 contains 22272 AND gates out of 116245 total gates. For sha256 implementation as suggested in [17], with Beaver optimization [2], 3976 computation rounds are required.

In the Algorithm 1, once drones posses an agreed upon and valid candidate image C_i, they compute X and $\overline{X}$ performing local computations – using $\mathbf{AND_m}$ operator, to compare the local target image share T_i with C and $\overline{C}$. In the Line 25 of the Algorithm 1, MPC is used to perform ThresholdOr($[x], t$) twice – once for Y and once for $\overline{Y}$ calculations. The MPC calculation itself consists of a fixed number of parameters sharing rounds and local calculations that do not require any communication, as described in Line 25. It turns out that MPC computation of ThresholdOr($[x], t$) is done in a single computation round – not requiring any data exchange among the computing drones.

5 VES Solution with Projection and Counting Device

Solution based on VES and MPC, as introduced in algorithm 1, requires secure communication channels among computation parties. Moreover, these communication channels should be reliable and support $O(n^2 \cdot l)$ communication complexity. Here n is number of processors and l is input size - in our case - number of pixels in images, that depend on camera resolution. To cope with these challenges, we want to offer an alternative solution version. The proposed solution can be applied in different setups, so we generalize the operating setup here. The drone (UAV) is going to be replaced by mobile agent (or agent), and the swarm of drones is swarm of mobile agents. The only requirement on mobile agents swarm is that the agents can communicate with the leader (or image owner - depending on setup). The candidate images can be acquired by the mobile agents or fed to the swarm from an external source.

5.1 The Simple Scheme

There is a swarm of n identical mobile agents (from now we call it agents), every agent has a processing unit, local storage and camera to take images. We do not require any particular computational power or minimal security specification on the agent hardware. For the sake of simplicity, it can be assumed that the images consist of black and white pixels, as described in Section 3.

The proposed solution consists of two phases:

Initialization phase The target image is encrypted with VES scheme with threshold $t == n$, resulting in $\{T_1, \ldots, T_n\}$ slides that reveal no information about the target image. Every agent is given T_i slide.

Operation phase – After the **initialization phase**, the agents start acquiring candidate images trying to match them to the original target image. In the simplest solution, the agent that acquired candidate image and is willing to compare it to the target image should reconstruct it from the VES slides. To do so, it can request other agents to share their secret slides T_i. When receiving the slides and reconstructing the target image, the processing agent compromises the privacy of the target image, since the agent itself, or its communication channel, can be exposed to an adversary attack. The proposed solution works with $O(n \cdot l)$ communication complexity, thus has better performance than the Algorithm 1.

5.2 Improved Scheme

We assume here the same initialization of the mobile agents swarm as in previous section. We use the same setup as in simple scheme, but we add to it a "counting device". The counting device has the following capabilities:

1. Input device that can receive pixel matrices from agents.
2. RAM that can store array of pixel matrices.

3. Counting device that can calculate ThresholdOr operator as defined in Section 3.5.
4. Output device that signal 1 if images match and 0 if not.
5. Timer T_1. It starts when first bit is received. When time $T_1 expires$, the device resets its RAM, thus forgetting all received data.

As in simple scheme, the target image is encrypted with VES scheme with threshold $t == n$, and the agents are given T_i slides of the target image. In this solution, the agent that acquired candidate image and is willing to compare it to the target image doesn't reconstruct it from the VES slides. Instead, the agents perform $\mathbf{AND_m}$ – matrix logical **AND** operator defined in Section 3.5. The agent performs $\mathbf{AND_m}$ on candidate image C_i as well as on candidate image negative $\overline{C}_i$: $X_i = T_i\ AND_m\ C_i$, $\overline{X}_i = T_i\ AND_m\ \overline{C}_i$. X_i and $\overline{X}_i$ are pixel matrices of size $c \cdot m \times d \cdot m$. Now the agents transmit the pixel matrices X_i and $\overline{X}_i$ to the counting device. The device performs ThresholdOr operator on matrices' pixels, counting pixel matches. Eventually, it makes image match decision.

Let's look at the calculation steps in greater details.

Image VES Encryption As described in Section 2, target image is encrypted with VES scheme.

Local Calculation The agents perform $\mathbf{AND_m}$ on candidate image C_i as well as on candidate image negative $\overline{C}_i$ pixels: $X_i = T_i\ AND_m\ C_i$, $\overline{X}_i = T_i\ AND_m\ \overline{C}_i$. X_i and $\overline{X}_i$ are pixel matrices of size $c \cdot m \times d \cdot m$. An example of such calculation for case when T and C compared pixels are white, is shown on Figure 3. In the example we see that X_i pixel shares keep the original VES assigned values, thus will be used later for white pixels comparison. For black pixels comparison we use negative candidate image, as shown on Figure 3.

Lemma 1. *The defined local calculation preserves VES information theoretic-secure property of secret image slides T_i.*

Proof. To prove it lets analyze all possible $\mathbf{AND_m}$ combinations in Table 3. According to the $\mathbf{AND_m}$ definition in Section 3.5, the result of applying $\mathbf{AND_m}$ operator on single pixel is $m \times m$ bitmap. Applying it to the whole slide and candidate image, results in X_i and $\overline{X}_i$ pixel matrices of size $c \cdot m \times d \cdot m$. We denote b - black pixel, w - white pixel, g - grey bitmap. Grey bitmaps represent the original VES assigned values. As shown in Table 3, applying $\mathbf{AND_m}$ for white C pixels, preserves the original VES assigned grey values for X for both T_i values (black and white), and completely blacks $\overline{X}$. Blacked bitmaps reveal no information about the secret image pixels. Grey (partial) bitmaps do not reveal any information about the secret image pixels by VES definition. It proves that X holds the VES T_i slide information theoretic-secure property. The same proof can be done when applying $\mathbf{AND_m}$ for black C pixels. In this case, the original VES assigned grey values for $\overline{X}$ are preserved for both T_i values (black and white), and X bitmaps are completely black. ■

Algorithm 2: Swarming and Hiding Target Image Algorithm - with Counting Device

```
// Target image is delivered to dealer
Input: Target image T, match pixels threshold R, VES threshold t
Output: Swarm target image match decision
// MPC: Inputs X_i and X̄_i
function CountDevice([X_i], [X̄_i]):
    Y = ThresholdOr([X], t)
    Ȳ = ThresholdOr([X̄], t)
    // Calculate match image pixels count
    match_pixel_count = 0
    for Y[i,j] in Y AND Ȳ[i,j] in Ȳ do
        if Y[i,j] == 1 OR Ȳ[i,j] == 1 then
            match_pixel_count+ = 1
    // Report the candidate image matching result
    if match_pixel_count > R then
        return True
    else
        return False

function main():
    // 1. Swarm initialization
    // Dealer calculates
    [T] = VES(T)
    // Dealer distributes to agents the image slides
    for each T_i in [T] do
        (T_i) --send to--> P_i
    // 2. Swarm operation
    while True do
        // Each Agent P_i takes candidate image C_i from local camera
        C_i = TakeAgreedCandidateImage()
        // Agents local calculation
        // Each Agent P_i compares candidate image with their slides
        for each P_i in [P] do
            X_i = T_i AND_m C_i
            X̄_i = T_i AND_m C̄_i
        // Agents transmits [X_i],[X̄_i] to counting device for pixels
           counting and images match decision
        report CountDevice([X_i],[X̄_i])
```

Case Number	C	T	X	$\overline{X}$
1	b	b	b	g
2	b	w	b	g
3	w	b	g	b
4	w	w	g	b

Table 3. Table with all four possible local calculation combinations

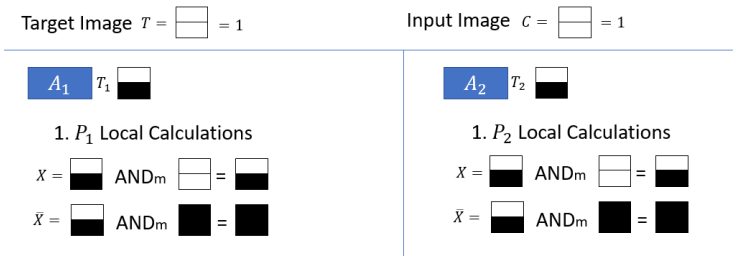


Fig. 3. Agents Local Calculation for Black Pixels

Counting Device Calculation As defined in Section 5.1, the counting device receives local calculation results - the pixel matrices X_i and $\overline{X}_i$, and performs ThresholdOr operator on matrices' pixels, counting pixel matches. An example of such calculation is shown on Figure 5 for white pixels, and on Figure 6 for black pixels. In Section 4 and in Tables 1 and 2, the correctness of the calculation is proven for all pixel combinations.

5.3 Analysis

Solution Security Security Model While analyzing the solution security we define several adversary types. There are **passive**, or **honest-but-curious adversaries**. Such adversaries can eavesdrop communication messages among the computing agents, or attack and corrupt computing agents. Passive adversaries can form coalition of up to k agents, thus they are able to view private information of up to k agents. Passive adversaries also can use the controlled agents to generate their own messages, but they have to follow the protocol. Opposite to a passive, **malicious (Byzantine)** adversaries may arbitrarily deviate from the protocol execution in their attempts to cheat. Our main security goal is to keep the original target image private during all images comparison steps. As we have proven in Section 4, an adversary that controls less than k drones can not reveal any information about the target image – at any computational step. It holds for improved solution described in Section 5.2 - for target image slides and local calculation steps. The information about the target image is revealed when pixel matrices X_i and $\overline{X}_i$ are sent (projected) to the counting device. An adversary that can attack the counting device, is able to eavesdrop the pixel matrices that bear secret target image slides T_i and reconstruct the T. So the solution model is bounded by security properties of the counting device.

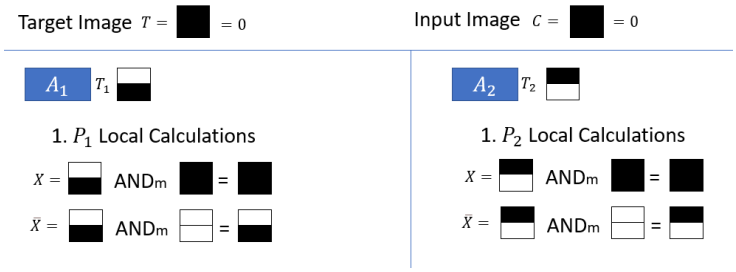


Fig. 4. Agents Local Calculation for White Pixels

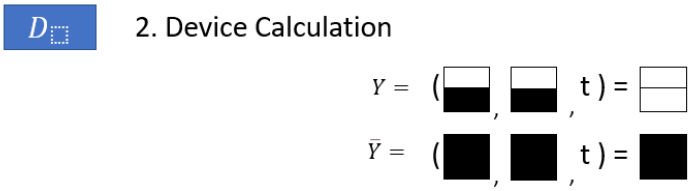


Fig. 5. Device Calculation for White Pixels

Solution Complexity In Section 6, we have shown that since Algorithm 1 uses MPC, it requires quadratic communication complexity. Calculations of Algorithm 2 require single data transfer - pixel matrices from agents to the counting device. It can be done in linear time - $O(n)$, where n - is the number of mobile agents. Additional advantage of the Algorithm 2 is that it doesn't require existence and maintenance of secure peer-to-peer communication channels among the agents, as MPC settings. It significantly improves not just communication complexity, but also overall agents swarm resilience in dynamic, noisy environments - for example swarm of UAVs, or agents operating over public communication infrastructure, e.g. - mobile phones.

Protocol and data structure optimizations Pixel matrices X, and $\overline{X}$ are calculated locally by local agents and than transmitted for global calculations - by all agents in Algorithm 1 for MPC, and by the counting device in Algorithm 2 for threshold calculation. These matrices can be represented in compact format, thus further saving communication bandwidth and required memory. As we have shown in Section 23, when we compare white C pixel with secret image slide T_i, its pixel is blacked in bitmap X for all T_i values. In the same way, when we compare black C pixel with secret image slide T_i, its pixel is blacked in bitmap $\overline{X}$ for all T_i values. We can use this property and store the local calculation results in single pixel matrix XX, thus saving half communication bandwidth and storage volume. Additional optimization is related to how VES pixel shares (bitmaps) are represented in slides. Every pixel share is a matrix of black and white sub-

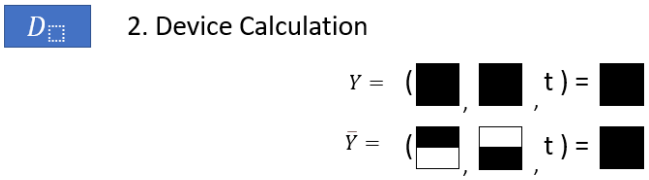


Fig. 6. Device Calculation for Black Pixels

pixels selected from predefined shares collection, for example shown on Figure 2. Without loosing any of VES functionality, we can decide that we actually keep only black subpixels. The complete bitmap, containing white subpixels, can be recreated on the fly, when needed for local calculations. This optimization saves half storage volume and communication bandwidth.

6 Conclusions and Extensions

The solution introduced in Section 3.5 allows a swarm of drones to make a collaborative target image matching. In Section 3.3, we have shown that the solution is resilient against a semi-honest adversary as well as against a malicious adversary that controls less than k drones. The solution feasibility is discussed in Section 4. Usage of Naor and Shamir Visual Encryption Scheme in conjunction with distributed, MPC computation model ensures a high solution security level and resilience against a malicious adversary controlling up to $k-1$ swarm drones. It turns out that our solution is surprisingly efficient while using MPC model, requiring only a small, fixed number of computation rounds, with local calculations that do not involve any data exchange among the drones during the computations, but inputs secret sharing. The Candidate Image Consensus Problem (Section 3.4) can be further researched in order to ensure that an adversary can not cause false activation of the swarm. The solution image matching precision, as described in Section 3.5, can be enhanced in the future with the usage of Hausdorff distance [14] for comparing image elements of target and candidate images. The current solution is not limited to the usage of Shamir and Naor visual encryption scheme, thus the use of alternative visual encryption constructions like suggested in [24,15] can improve the target image hiding capability. The solution scope can be extended to gray scale and color images. A significant security challenge of target image updates during the swarm operation can enhance practical implementation aspects of the solution. Imposing various hardware optimizations on drones hardware – like GPU usage or minimal communication channel requirements, can be evaluated in order to improve candidate image matching time.

An alternative VES based solution introduced in Section 5.2. This solution improves VES and MPC based solution communication complexity - from $O(n^2 \cdot l)$ to $O(n)$ and doesn't require secure communication channels among com-

putation parties. Combined with communication protocol and data structure optimizations proposed in Section 5.3, the communication and storage requirements are equal to number of pixels in candidate images, that depend on camera resolution. Security model of this solution is weaker, but, due to its advantages, it can be applied in various scenarios. An interesting application can be - peer to peer private storage of personal images. Here the images owner distributes images slides to peers (agents). In this setup, the agents are not trusted. Only the images owner has counting device and thus, the ability to reconstruct the secret images. When the image owner wants to compare the secret image to another image, applies the Algorithm 2 with its own counting device.

An alternative computational secure methodology can be encoding the target and candidate images with cryptographic hash, e.g. sha256. The hashes can be used for individual images match, without revealing the image to the participants and therefore can support the dynamic set of participants that copy the hashed value and also begin the verification.

7 Experiments

Several experiments that were performed with Algorithm 2 simulator demonstrate the algorithm operation. In the simulations, we initialized VES with 2 slides, and image pixels represented with four shares. The code, together with original images, can be found in this git public repository.

7.1 Experiment1 $bsquare40x40\_wqs20x20$

Black Square 40×40 compared with identical black square 40×40, with embedded 20×20 white square. as can be reconstructed from the pixel matrices X

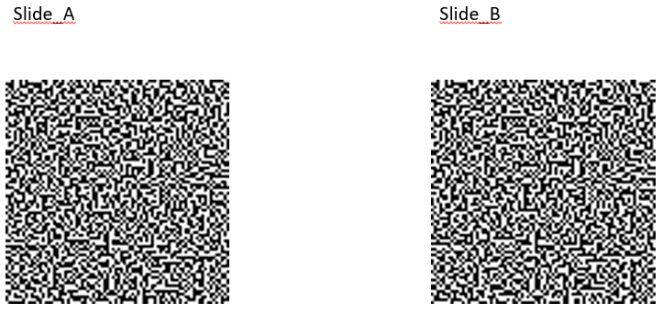


Fig. 7. Target Image Slides

transmitted to the counting device. These images are not reconstructed during the normal algorithm operation.

In the below images we see X_i and $\overline{X}_i$ as locally calculated by the agents, applying $\mathbf{AND_m}$ on T_i, C_i and $\overline{C}_i$. Note that the left image *Slide_A_P* is almost identical with candidate image C_i and the right image *Slide_A_N* is almost identical with $\overline{C}_i$. The only differences are that white pixels of the original image become grey for C_i and black pixels of the original image become grey for $\overline{C}_i$. We omitted here images demonstrating calculations with slide B since they are identical to slide A calculations.

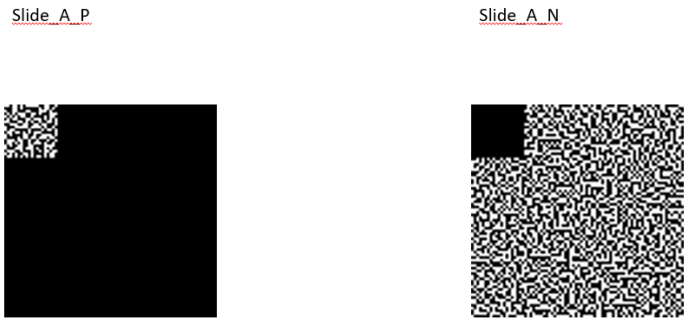


Fig. 8. Slide A AND with Candidate Image

In the below images we see positive and negative candidate image as can be reconstructed from the pixel matrices X transmitted to the counting device. These images are not reconstructed during the normal algorithm operation.

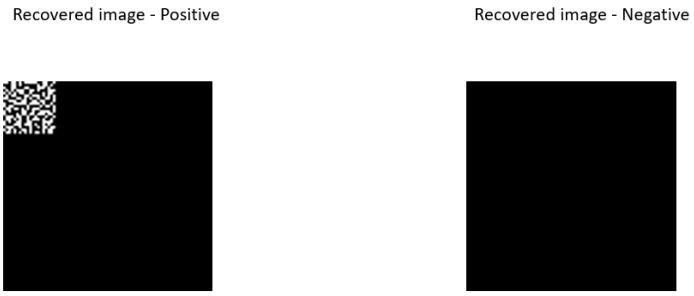


Fig. 9. Recovered images

The compare images result image is:
Positive image: match pixel count: 100 out of 1600 pixels, match percentage: 6.25%

Negative image: match pixel count: 1200 out of 1600 pixels, match percentage: 75.00%

7.2 Experiment2 *LennaBWOrig_bs*

Lenna original image is compared with Lenna image with an addition of black square on Lenna's shoulder.

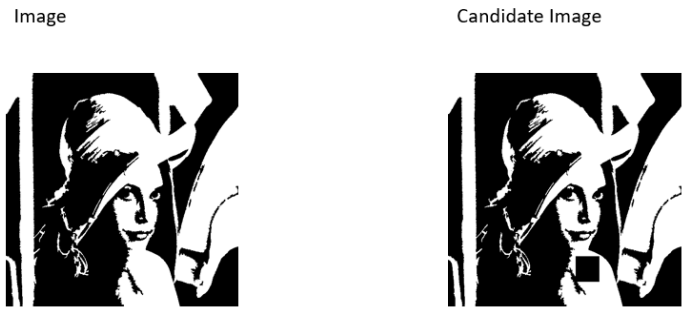


Fig. 10. Target Image Slides

In the below images we see X_i and $\overline{X}_i$ as locally calculated by the agents, applying $\mathbf{AND_m}$ on T_i, C_i and $\overline{C}_i$. Note that the left image *Slide_A_P* is almost identical with candidate image C_i and the right image *Slide_A_N* is almost identical with $\overline{C}_i$. The only differences are that white pixels of the original image become grey for C_i and black pixels of the original image become grey for $\overline{C}_i$. This is interesting visual aspect of the calculation. The pixel matrices bear visual representation of candidate image, but reveal no information about the secret target image. We omitted here images demonstrating calculations with slide B since they are identical to slide A.

In the below images we see positive and negative candidate image as can be reconstructed from the pixel matrices X transmitted to the counting device. These images are not reconstructed during the normal algorithm operation, and are shown here just for algorithm description purpose.

The compare images result is:
Positive image: match pixel count: 111562 out of 262144 pixels, match percentage: 42.56%
Negative image: match pixel count: 147693 out of 262144 pixels, match percentage: 56.34%

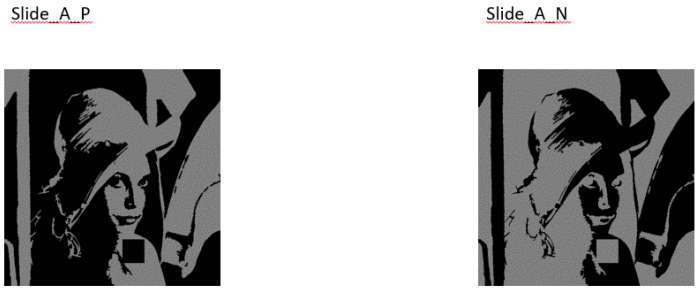


Fig. 11. Slide A AND with Candidate Image

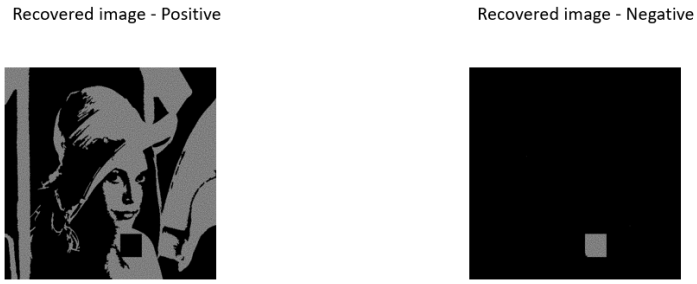


Fig. 12. Recovered images

References

1. Albrecht, M.R., Grassi, L., Perrin, L., Ramacher, S., Rechberger, C., Rotaru, D., Roy, A., Schofnegger, M.: Feistel structures for mpc, and more. In: European Symposium on Research in Computer Security. pp. 151–171. Springer (2019)
2. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Annual International Cryptology Conference. pp. 420–432. Springer (1991)
3. Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols. In: Proceedings of the twenty-second annual ACM symposium on Theory of computing. pp. 503–513 (1990)
4. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation, p. 351–371. Association for Computing Machinery, New York, NY, USA (2019), https://doi.org/10.1145/3335741.3335756
5. Berend, D., Dolev, S., Frenkel, S., Hanemann, A.: Towards holographic "brain" memory based on randomization and walsh–hadamard transformation. Neural Networks **77**, 87–94 (2016)
6. Bruckstein, A.M., Holt, R.J., Netravali, A.N.: Holographic representations of images. IEEE Transactions on Image Processing **7**(11), 1583–1597 (1998)

7. Chaum, D., Crépeau, C., Damgard, I.: Multiparty unconditionally secure protocols. In: Proceedings of the twentieth annual ACM symposium on Theory of computing. pp. 11–19 (1988)
8. Dolev, S., Frenkel, S.: Multiplication free holographic coding. In: 2010 IEEE 26-th Convention of Electrical and Electronics Engineers in Israel. pp. 000146–000150. IEEE (2010)
9. Gennaro, R., Rabin, M.O., Rabin, T.: Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In: Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing. p. 101–111. PODC '98, Association for Computing Machinery, New York, NY, USA (1998). https://doi.org/10.1145/277697.277716, https://doi.org/10.1145/277697.277716
10. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 580–587 (2014)
11. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing. p. 218–229. STOC '87, Association for Computing Machinery, New York, NY, USA (1987). https://doi.org/10.1145/28395.28420, https://doi.org/10.1145/28395.28420
12. Goldreich, O., Micali, S., Wigderson, A.: How to Play Any Mental Game, or a Completeness Theorem for Protocols with Honest Majority, p. 307–328. Association for Computing Machinery, New York, NY, USA (2019), https://doi.org/10.1145/3335741.3335755
13. Goldwasser, S., Ben-Or, M., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computing. In: Proc. of the 20th STOC. pp. 1–10 (1988)
14. Huttenlocher, D., Klanderman, G., Rucklidge, W.: Comparing images using the hausdorff distance. IEEE Transactions on Pattern Analysis and Machine Intelligence **15**(9), 850–863 (1993). https://doi.org/10.1109/34.232073
15. Jiao, S., Feng, J., Gao, Y., Lei, T., Yuan, X.: Visual cryptography in single-pixel imaging. Opt. Express **28**(5), 7301–7313 (Mar 2020). https://doi.org/10.1364/OE.383240, http://opg.optica.org/oe/abstract.cfm?URI=oe-28-5-7301
16. Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., Fei-Fei, L.: Large-scale video classification with convolutional neural networks. In: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. pp. 1725–1732 (2014)
17. Keller, M., Scholl, P., Smart, N.P.: An architecture for practical actively secure mpc with dishonest majority. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security. p. 549–560. CCS '13, Association for Computing Machinery, New York, NY, USA (2013). https://doi.org/10.1145/2508859.2516744, https://doi.org/10.1145/2508859.2516744
18. Lapets, A., Howe, W., Getchell, B., Jansen, F.: An embedded domain-specific language for logical circuit descriptions with applications to garbled circuits. IACR Cryptol. ePrint Arch. **2020**, 1604 (2020)
19. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3431–3440 (2015)

20. Naor, M., Shamir, A.: Visual cryptography. In: Workshop on the Theory and Application of of Cryptographic Techniques. pp. 1–12. Springer (1994)
21. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority. In: Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing. p. 73–85. STOC '89, Association for Computing Machinery, New York, NY, USA (1989). https://doi.org/10.1145/73007.73014, https://doi.org/10.1145/73007.73014
22. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 779–788 (2016)
23. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)
24. Verheul, E.R., Van Tilborg, H.C.: Constructions and properties of k out of n visual secret sharing schemes. Designs, Codes and Cryptography **11**(2), 179–196 (1997)

CPU- and GPU-based Distributed Sampling in Dirichlet Process Mixtures for Large-scale Analysis

Or Dinari*[1] dinari@post.bgu.ac.il
Raz Zamir*[1] razzam@post.bgu.ac.il
John W. Fisher III[2] fisher@csail.mit.edu
Oren Freifeld[1] orenfr@cs.bgu.ac.il

[1]Computer Science, Ben-Gurion University, Beer-Sheva, Israel
[2]MIT CSAIL, Cambridge MA, USA

Abstract

In the realm of unsupervised learning, Bayesian nonparametric mixture models, exemplified by the Dirichlet Process Mixture Model (DPMM), provide a principled approach for adapting the complexity of the model to the data. Such models are particularly useful in clustering tasks where the number of clusters is unknown. Despite their potential and mathematical elegance, however, DPMMs have yet to become a mainstream tool widely adopted by practitioners. This is arguably due to a misconception that these models scale poorly as well as the lack of high-performance (and user-friendly) software tools that can handle large datasets efficiently. In this paper we bridge this practical gap by proposing a new, easy-to-use, statistical software package for scalable DPMM inference. More concretely, we provide efficient and easily-modifiable implementations for high-performance distributed sampling-based inference in DPMMs where the user is free to choose between either a multiple-machine, multiple-core, CPU implementation (written in Julia) and a multiple-stream GPU implementation (written in CUDA/C++). Both the CPU and GPU implementations come with a common (and optional) python wrapper, providing the user with a single point of entry with the same interface. On the algorithmic side, our implementations leverage a leading DPMM sampler from [1]. While Chang and Fisher III's imple-

mentation (written in MATLAB/C++) used only CPU and was designed for a single multi-core machine, the packages we proposed here distribute the computations efficiently across either multiple multi-core machines or across mutiple GPU streams. This leads to speedups, alleviates memory and storage limitations, and lets us fit DPMMs to significantly larger datasets and of higher dimensionality than was possible previously by either [1] or other DPMM methods. Our open-source code (GPLv2 licensed) is publicly available on `github.com`.

1 Introduction

In unsupervised learning, Bayesian Nonparametric (BNP) mixture models, exemplified by the Dirichlet-Process Mixture Model (DPMM), provide a principled approach for Bayesian modeling while adapting the model complexity to the data. This contrasts with finite mixture models whose complexity is determined manually or via model-selection methods. To fix ideas, an important DPMM example is the Dirichlet-Process Gaussian Mixture Model (DPGMM), a Bayesian ∞-dimensional extension of the classical Gaussian Mixture Model (GMM). Despite their potential, however, and although researchers have used them successfully in numerous applications during the last two decades, DPMMs still do not enjoy wide popularity among practitioners, largely due to computational bottlenecks that exist in current algorithms and/or implementations. In particular, one of the missing pieces is the availability of software tools that: 1) can efficiently handle DPMM inference in large datasets; 2) are user-friendly and can also be easily modified.

We argue that in order *for DPMMs to become a practical choice for large-scale data analysis, implementations of DPMM inference must leverage parallel- and distributed-computing resources* (in an analogy, consider how advances in GPU computing and GPU software contributed to the success of deep learning). This is because of not only potential speedups but also memory and storage considerations. For example, this is especially true in distributed mobile robotic sensing applications where multiple autonomous agents working together have limited computational and communication resources. As another motivating example, consider unsupervised data-analysis tasks in large and high-dimensional computer-vision datasets.

In other words, while DPMMs are theoretically ideal for handling unlabeled datasets, current implementations of DPMM inference do not scale well with the size of the dataset and/or the dimensionality.

Package	Processor	Description
CUDA/C++	GPU	The fastest package for high N (number of data points) and d (data dimensions) on a single machine; supports multiple GPU streams.
Julia	CPU	Supports both multiple cores and multiple machines.
Python	Either CPU or GPU	Wrapper to the CUDA/C++ and Julia packages

Table 1: Overview of the proposed packages

This is partly since most existing implementations are serial and they do not harness the power of distributed computing. This does not mean that there do not exist distributed *algorithms* for DPMM inference. *There is, however, a large practical gap between designing such an algorithm and having it implemented efficiently in a way that fully utilizes the available computing resources*. Thus, the very few publicly-available implementations of such distributed algorithms are fairly limited in their capabilities (as well as their expressiveness; *e.g.*, some support only isotropic Gaussians [2], *etc.*). Our work closes this gap by providing effective and scalable statistical software for typical large-scale inference.

Concretely, we propose two solutions from which the users can choose according to their needs, constraints, and available computing resources. The first proposed solution, implemented purely in CPU, is based on distributing computations across multiple cores as well as multiple machines. The second proposed solution, based mostly on GPU, relies on distributing computations across multiple GPU streams in a single machine. See Table 1 for more details.

More generally (than the topic of DPMM inference), distributed implementations, at least in traditional programming languages used for such implementations, tend to be hard to debug, read, and modify. This clashes with the usual workflow of algorithm development. As a remedy, in our recent workshop paper [3], which constitutes a preliminary and partial version of this paper, we proposed a Julia implementation for distributed sampling-based DPMM inference. Since the publication of [3] we have improved the performance of our Julia implementation, have added its GPU counterpart in CUDA/C++, and added an optional Python wrapper for both the CPU and GPU implementations.

To summarize, in this paper we explain how to use, via either Julia or CUDA/C++, an efficient distributed DPMM inference for large-scale analysis. Particularly, based on a leading parallel Markov Chain Monte Carlo (MCMC) inference algorithm [1] (to be discussed in section 2) – originally implemented in C++ for a single multi-core CPU single machine in a highly-specialized fashion using a shared-memory model – we provide novel, more scalable, and easier-to-use-or-modify implementations that leverage either the latest Nvidia's asynch memory allocation API for GPU or Julia's capabilities for distributing CPU computations efficiently across multiple multi-core machines using a distributed memory model. This leads to speedups, alleviates memory and storage limitations, and lets us infer DPMMs from significantly larger and higher-dimensional datasets than was previously possible by either [1] or other DPMM inference methods. Our Julia and CUDA/C++ implementations are also accompanied by an optional Python wrapper which hides the Julia & CUDA/C++ code from the user and provides a single point of entry that lets the user employ, in the same settings and with the same code, either one of our CPU and GPU packages.

2 Models and the Inference Algorithm

2.1 Preliminaries: Finite Mixture Models and Clustering

Let d and K be two positive integers and let $\boldsymbol{x}$ be a generic point in $\mathbb{R}^d$. A K-component Finite Mixture Model (FMM), also known as a parametric mixture model, is a probabilistic model whose associated d-dimensional probability density function (pdf) is

$$p(\boldsymbol{x};\theta) = \sum_{k=1}^{K} \pi_k f(\boldsymbol{x};\theta_k) \tag{1}$$

where $\theta = (\pi_k, \theta_k)_{k=1}^K$, $(\pi_k)_{k=1}^K$ form a convex combination (namely, $\sum_{k=1}^K \pi_k = 1$ and $\pi_k \geq 0\ \forall k$), and $f(\boldsymbol{x};\theta_k)$ is a d-dimensional pdf parameterized by θ_k. The $(f(\cdot;\theta_k))_{k=1}^K$ functions are called the mixture *components* while $(\pi_k)_{k=1}^K$ are called the mixture *proportions* (or *weights*).

Example 1 *In the (finite) Gaussian Mixture Model (GMM), $\theta_k = (\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ and $f(\boldsymbol{x};\theta_k) = \mathcal{N}(\boldsymbol{x};\boldsymbol{\mu}_k,\Sigma_k)$ where the latter is a multivariate Gaussian pdf evaluated at $\boldsymbol{x}$ and parameterized by a mean vector, $\boldsymbol{\mu}_k \in \mathbb{R}^d$, and a Symmetric Positive-Definite (SPD) $d \times d$ covariance matrix, $\boldsymbol{\Sigma}_k$. In other words, in this example*

$$f(\boldsymbol{x};\theta_k) = \mathcal{N}(\boldsymbol{x};\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k) \triangleq (2\pi)^{-d/2}(\det\boldsymbol{\Sigma}_k)^{-1/2}\exp(-\tfrac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_k))\,. \tag{2}$$

The case where $\boldsymbol{x}$ takes values in a discrete set is similar, except that each component is a probability mass function (pmf), not a pdf. A popular example is a mixture of categorical distributions or, more generally, multinomial distributions.

Let $\boldsymbol{X} = (\boldsymbol{x}_i)_{i=1}^N$ stand for N data points and again let $K > 0$ be an integer. *Clustering* is the task of partitioning $\boldsymbol{X}$ into K parts, called *clusters* and denoted by $\boldsymbol{C} = (C_k)_{k=1}^K$. Let z_i denote the latent point-to-cluster assignment of $\boldsymbol{x}_i$; *i.e.*, cluster k is defined as $C_k = (\boldsymbol{x}_i)_{i:z_i=k}$. In the standard formulation of FMM-based clustering, the data points are assumed to be independent and identically-distributed (*i.i.d.*) draws from an FMM and one typically tries to maximize the corresponding likelihood function, $p(\boldsymbol{X}|\theta)$, over θ. The most popular approach for (locally-) maximizing that likelihood is via *an* Expectation-Maximization (EM) algorithm [4]. As the time-honored Bayesian formulation helps avoiding problems such as over-fitting and enables encoding prior knowledge, the FMM also has a Bayesian (but still finite) variant, where θ is assumed to be random and drawn from a suitable prior [5] (specifically, the prior over $(\pi_k)_{k=1}^K$ is usually a Dirichlet *distribution*, not to be confused with a Dirichlet *process*). In which case, one targets the *posterior* distribution, $p(\theta|\boldsymbol{X})$, rather than the likelihood; *e.g.*, one can try to maximize $p(\theta|\boldsymbol{X})$ over θ, sample θ from it, compute posterior expectations, *etc*.

2.2 The Dirichlet Process Mixture Model (DPMM)

Below we provide a brief and *informal* introduction to the DPMM, focusing only on the essentials required for understanding this manuscript and make it self-contained as possible. For a comprehensive and formal mathematical treatment, see [6]. For an applied data-analysis perspective, see [7]. For a gentle introduction with machine-learning and/or computer-vision readers in mind, see the theses by [8] or [9].

The DPMM is a BNP extension of the FMM [10]. Loosely speaking, a DPMM entertains the notion of a mixture of infinitely-many *components*.

The weights, $\boldsymbol{\pi} = (\pi_k)_{k=1}^\infty$, are drawn from a Griffiths-Engen-McCloskey (GEM) stick-breaking process with a concentration parameter $\alpha > 0$ [11], while the components, $(\theta_k)_{k=1}^\infty$, are drawn from a prior as in the Bayesian FMM case.

Example 2 *The Dirichlet Process GMM (DPGMM) is a GMM with infinitely-many Gaussians.*

Like the FMM, the DPMM is often used for clustering. With a DPMM, however, the number of *clusters*, K, is not assumed to be known; rather, it is viewed as a latent random variable whose value is inferred with the rest of the model.

While in an FMM the number of clusters and the number of components are typically equal, and are thus both denoted by a single symbol, K, with a DPMM the situation is different: although there are infinitely-many components, the (latent and random) number of clusters, K, is finite (particularly, $K \leq N$). The inferred value of K depends on α (the higher α is, the more clusters are expected), on the complexity of the data, and the (hyper-parameters of the) prior over the components (that said, when N is large, the last two factors are usually more influential than α).

Example 3 *Recall that the standard prior for Gaussian components is a Normal Inverse-Wishart (NIW) distribution [5]. If the NIW's hyper-parameters are set to strongly favor small covariances (hence small clusters), then this will implicitly favor a large K. Likewise, if the NIW prior strongly favors larger covariances (hence large clusters), then K will tend to be small. In the lack of prior knowledge, however, the NIW prior can be set to be very weak* (i.e., *high uncertainty), letting the data speak for itself.*

For simplicity, our text below implicitly assumes that all the random vectors involved have either a pdf or a pmf. One known mathematical construction of the DPMM uses the following distributions:

$$\boldsymbol{\pi}|\alpha \sim \mathrm{GEM}(\boldsymbol{\pi}; \alpha), \tag{3}$$

$$\theta_k|\lambda \overset{i.i.d.}{\sim} f_\theta(\theta_k; \lambda), \qquad \forall k \in \{1, 2, \ldots\}, \tag{4}$$

$$z_i|\boldsymbol{\pi} \overset{i.i.d.}{\sim} \mathrm{Cat}(z_i; \boldsymbol{\pi}) \qquad \forall i \in \{1, 2, \ldots, N\}, \tag{5}$$

$$\boldsymbol{x}_i|z_i, \theta_{z_i} \sim f_{\boldsymbol{x}}(\boldsymbol{x}_i; \theta_{z_i}), \qquad \forall i \in \{1, 2, \ldots, N\}\,. \tag{6}$$

Here, $f_\theta(\cdot; \lambda)$ is the pdf or pmf (associated with some base measure) parameterized by λ, the infinite-length vector $\boldsymbol{\pi} = (\pi_k)_{k=1}^{\infty}$ is drawn from a GEM stick-breaking process with a concentration parameter $\alpha > 0$ (particularly, $\pi_k > 0$ for every k and $\sum_{k=1}^{\infty} \pi_k = 1$) while θ_k is drawn from f_θ. Each of the N *i.i.d.* observations $(\boldsymbol{x}_i)_{i=1}^{N}$ is generated by first drawing a label, $z_i \in \mathbb{Z}^+$, from $\boldsymbol{\pi}$ (*i.e.*, Cat is the categorical distribution), and then $\boldsymbol{x}_i$ is drawn from (a pdf or a pmf) $f_{\boldsymbol{x}}$ parameterized by θ_{z_i}. Informally,

$$\boldsymbol{x}_i \overset{i.i.d.}{\sim} \sum_{k=1}^{\infty} \pi_k f_{\boldsymbol{x}}(\boldsymbol{x}_i; \theta_k)\,. \tag{7}$$

Here too, each $f_{\boldsymbol{x}}(\cdot;\theta_k)$ is called a *component* and we make no distinction between a component, $f_{\boldsymbol{x}}(\cdot,\theta_k)$, and its parameter, θ_k. The so-called labels $(z_i)_{i=1}^N$ encode the observation-to-component assignments. A cluster is a collection of points sharing a label; *i.e.*, $\boldsymbol{x}_i$ is in cluster k, denoted by C_k, if and only if $z_i = k$. Let (the random variable) K be the number of unique labels: $K = |\{k : z_i = k \text{ for some } i \in \{1,\ldots,N\}\}|$; *i.e.*, K is also the number of clusters and is bounded above by N. Typically, and as assumed in this manuscript, f_θ is chosen to be a conjugate prior [5] to $f_{\boldsymbol{x}}$. The latent variables here are K, $(\theta_k)_{k=1}^\infty$, $\boldsymbol{\pi}$, and $(z_i)_{k=1}^N$. For more details (and other constructions), see [8].

Example 4 *In the case of Gaussian components, where $f_\theta(\cdot;\lambda)$ is an NIW pdf, we have*

$$f_\theta(\overbrace{\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k}^{\theta_k};\overbrace{\kappa,\boldsymbol{m},\nu,\boldsymbol{\Psi}}^{\lambda}) = \mathrm{NIW}(\boldsymbol{\mu},\boldsymbol{\Sigma};\kappa,\boldsymbol{m},\nu,\boldsymbol{\Psi}) \triangleq \mathcal{N}(\boldsymbol{\mu};\boldsymbol{m},\tfrac{1}{\kappa}\boldsymbol{\Sigma})\mathcal{W}^{-1}(\boldsymbol{\Sigma};\nu,\boldsymbol{\Psi}) \tag{8}$$

where $\mathcal{W}^{-1}(\boldsymbol{\Sigma};\nu,\boldsymbol{\Psi})$ is the Inverse-Wishart distribution (over $d \times d$ SPD matrices), the hyperparamers are

$$\lambda = (\boldsymbol{m},\boldsymbol{\Psi},\kappa,\nu) \tag{9}$$

where $\boldsymbol{m} \in \mathbb{R}^d$, the $d \times d$ matrix $\boldsymbol{\Psi}$ is SPD, and the two real numbers κ and ν satisfy $\kappa > 0$ and $\nu > d-1$ (do not confuse κ ("kappa") with k, the index of the component).

2.3 Inference via Chang and Fisher III's DPMM Sampler

We now briefly review a DPMM sampler proposed by [1]. That sampler consists of a restricted Gibbs sampler [12] and a split/merge framework [13] which together form an ergodic Markov chain. The operations in each step of that sampler are highly parallelizable. Importantly, the splits and merges let the sampler make *large moves* along the (posterior) probability surface as in such operations multiple labels change their label *simultaneously* to the same different label. This is unlike what happens, *e.g.*, in methods that must change each label separately from the others. We now describe the essential details.

The augmented space. The latent variables, $(\theta_k)_{k=1}^\infty$, $\boldsymbol{\pi}$, and $(z_i)_{k=1}^N$, are augmented with auxiliary variables. For each component θ_k two subcomponents (conceptually thought of as l="left" and r="right) are added, $\bar{\theta}_{k,l}, \bar{\theta}_{k,r}$,

with subcomponent weights $\bar{\boldsymbol{\pi}}_k = (\bar{\pi}_{k,l}, \bar{\pi}_{k,r})$. Implicitly, this means that every cluster C_k is augmented with two subclusters, $\bar{C}_{k,l}$ and $\bar{C}_{k,r}$. For each cluster label z_i, an additional *subcluster label*, $\bar{z}_i \in \{l, r\}$, is added; *i.e.*, subcluster $\bar{C}_{k,l} \subset C_k$ consists of all the points in C_k whose subcluster label is l (the other subcluster, $\bar{C}_{k,r}$, is defined similarly).

The restricted Gibbs sampler. This restricted sampler is not allowed to change (the current estimate of) K; rather, it can change only the parameters of the existing clusters and subclusters, and when sampling the labels, it can assign an observation only to an existing cluster. For each instantiated component k, changing

$$\theta_k, \bar{\theta}_{k,l}, \text{ and } \bar{\theta}_{k,r} \tag{10}$$

is done using

$$p(\theta_k|C_k;\lambda), p(\bar{\theta}_{k,l}|\bar{C}_{k,l};\lambda), \text{ and } p(\bar{\theta}_{k,r}|\bar{C}_{k,r};\lambda)\,, \tag{11}$$

respectively, where the latter three are the conditional distributions of the cluster or subcluster parameters given the cluster or subclusters (and the prior hyperparamers, λ) . In section 4, we will dive deeper into the details of the restricted Gibbs sampler.

The split/merge framework. Splits and merges allow the sampler to change K using the Metropolis-Hastings framework [14]. Particularly, the auxiliary variables are used to propose splitting an existing cluster or merging two exiting ones. When a split is accepted, each of the newly-born clusters is augmented with two new subclusters. The Hastings ratio of a split is [1]:

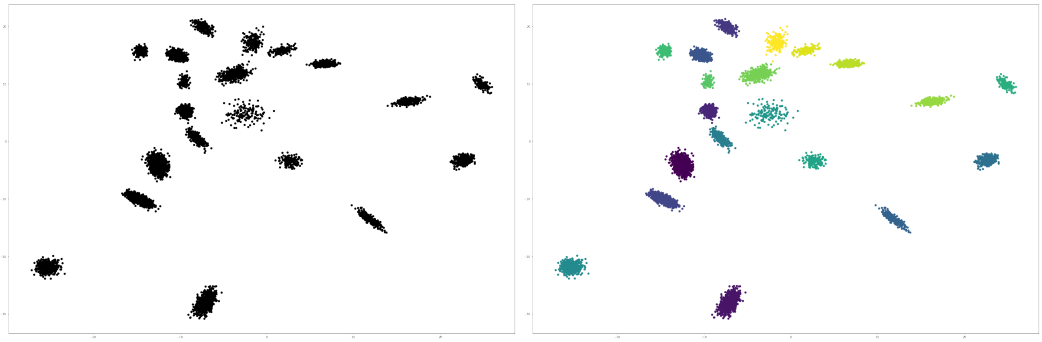
$$\mathrm{H}_{\text{split}} = \frac{\alpha\Gamma(N_{k,l})f_{\boldsymbol{x}}(\bar{C}_{k,l};\lambda)\Gamma(N_{k,r})f_{\boldsymbol{x}}(\bar{C}_{k,2};\lambda)}{\Gamma(N_k)f_{\boldsymbol{x}}(C_k;\lambda)} \tag{12}$$

where Γ is the Gamma function, N_k, $N_{k,l}$ and $N_{k,r}$ are the numbers of points in C_k, $\bar{C}_{k,l}$ and $\bar{C}_{k,r}$, respectively, while

$$f_{\boldsymbol{x}}(C_k;\lambda), f_{\boldsymbol{x}}(\bar{C}_{k,l};\lambda), \text{ and } f_{\boldsymbol{x}}(\bar{C}_{k,r};\lambda) \tag{13}$$

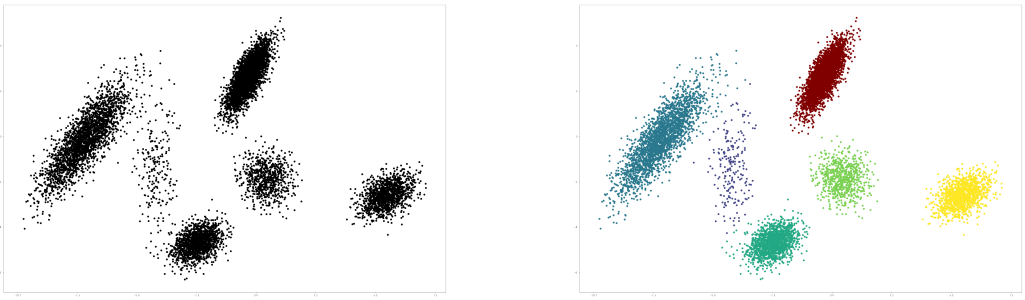
represent the *marginal* likelihood of C_k, $\bar{C}_{k,l}$ and $\bar{C}_{k,r}$ respectively. Concrete expressions for the marginal likelihood, in the case of Gaussian or Multinomial components (the component types considered in our experiments) appear in [9].

Finally, a *merge* proposal is based on taking two existing clusters and proposing merging them into one. The corresponding Hastings ratio is



(a) Unlabeled data (b) DPMMSubClusters Detection

Figure 1: 20 clusters detected by DPMMSubClusters



(a) Unlabeled data (b) DPMMSubClusters Detection

Figure 2: 6 clusters detected by DPMMSubClusters

$H_{merge} = 1/H_{split}$ where $\bar{C}_{k,l}$ and $\bar{C}_{k,r}$ are replaced with the two clusters, and C_k is replaced with the result of the merge. For the derivation behind H_{split} and H_{merge}, see [1].

Importantly, and as any successful DPMM inference method, Chang and Fisher III's sampler can detect different numbers of clusters according to the complexity of the dataset. For example, in Figure 1 we demonstrate unlabeled data with 20 clusters while in Figure 2 we show data consisting of 6 clusters. Using our implementation (the topic of the next section), the sampler correctly detected the different numbers of clusters in each dataset, while using the same code and the same hyperparamters.

3 Software

The proposed software supports three different software languages (Julia, CUDA/C++, Python), two operating systems (Windows & Linux) as well as multiple interface options. In terms of performance, the proposed software is faster than other publicly-available packages that exist today and that we were able to test. Particularly, as long as the product of N (number of data points) times d (the dimension of the data) is not too low, then our GPU version is consistently at least a few times faster than any other implementation we are aware of.

3.1 Installation

Our software (licensed under GPLv2) is available on GitHub. The URLs for our packages appear in Table 2. In order to compile the CUDA/C++ package the user will need C++14 (or higher) and CUDA 11.2 (or higher). For visualization purposes (*i.e.*, plotting points in 2D), the CUDA/C++ windows version also requires **OpenCV**. However, this visualization is used only for debugging purposes so the installation of **OpenCV** is not mandatory for using our implementation. For the CPU package the user will need Julia 1.5 (or higher). For Python we used 3.8. The installation steps below were tested successfully on Windows 10 (Visual Studio 2019), Ubuntu 18.04, and Ubuntu 21.04.

(1) Install CUDA version 11.2 (or higher) from
https://developer.nvidia.com/CUDA-downloads

(2) git clone https://github.com/BGU-CS-VIL/DPMMSubClusters_GPU

(3) Install Julia from: https://julialang.org/downloads/platform

(4) Add our DPMMSubCluster package from within a Julia terminal via Julia package manager:
```
]  add DPMMSubClusters
```

(5) Add our dpmmpython package in python:
pip install dpmmpython

(6) Add Environment Variables:

- On Linux:

(a) Add "CUDA_VERSION" with the value of the version of your CUDA installation (*e.g.*, 11.5).

(b) Add to the "PATH" environment variable the path to the Julia executable (*e.g.*, in .bashrc add: export PATH =$PATH:$HOME/julia/julia-1.6.0/bin).

(c) Make sure that CUDA_PATH exist. If it is missing add it with a path to CUDA (*e.g.*, export CUDA_PATH=/usr/local/cuda-11.6/).

(d) Make sure that the relevant CUDA paths are included in $PATH and $LD_LIBRARY_PATH (*e.g.*, export PATH=/usr/local/cuda-11.6/bin:$PATH, export LD_LIBRARY_PATH=/usr/local/cuda-11.6/lib64:$LD_LIBRARY_PATH).

- On Windows:

(a) Add "CUDA_VERSION" with the value of the version of your CUDA installation (*e.g.*, 11.5).

(b) Add to the "PATH" environment variable the path to the Julia executable (*e.g.*, C:\Users\¡USER¿\AppData\Local\Programs\Julia\Julia-1.6.0\bin).

(c) Make sure that CUDA_PATH exists. If it is missing add it with a path to CUDA (*e.g.*, C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.6).

(7) Install cmake if necessary.

(8) Install PyJulia from within a Python terminal:
```
import julia;julia.install();
```

(9) For Windows only (optional, used on for debugging purposes): Install **OpenCV**

(a) run Git Bash

(b) cd ¡YOUR_PATH_TO_DPMMSubClusters_GPU¿/DPMMSubClusters

(c) ./installOCV.sh

Package	GitHub
CUDA/C++	https://github.com/BGU-CS-VIL/DPMMSubClusters_GPU
Julia	https://github.com/BGU-CS-VIL/DPMMSubClusters.jl
Python	https://github.com/BGU-CS-VIL/dpmmpython

Table 2: GitHub URL for each package.

3.2 Building

For Windows for the CUDA/C++ package both of the build options below are viable. For Linux use Option 2.
Option 1: DPMMSubClusters.sln - Solution file for Visual Studio 2019
Option 2: CMakeLists.txt

(1) Run in the terminal:

```
cd <YOUR_PATH_TO_DPMMSubClusters_GPU>/DPMMSubClusters
mkdir build
cd build
cmake -S ../
```

(2) Build:
Windows:

```
cmake --build . --config Release --target ALL_BUILD
```

Linux:

```
cmake --build . --config Release --target all
```

3.3 Post Build

Add Environment Variable:

- On Linux:
  Add "DPMM_GPU_FULL_PATH_TO_PACKAGE_IN_LINUX" with the value of the path to the binary of the package DPMMSubClusters_-GPU.
  The path is: <YOUR_PATH_TO_DPMMSubClusters_GPU>/DPMMSubClusters/DPMMSubClusters.
- On Windows:
  Add "DPMM_GPU_FULL_PATH_TO_PACKAGE_IN_WINDOWS" with the value of the path to the exe of the package DPMMSubClusters_-GPU.

The path is: <YOUR_PATH_TO_DPMMSubClusters_GPU>\DPMMSubClusters\build\Release\DPMMSubClusters.exe.

3.4 Executing

Below are listed several options to run DPMM inference using our software. Usually the best performance will be achieved by running the CUDA/C++ version which uses the GPU (when available).

3.4.1 From Julia code - CPU

The following sample code generates a synthetic GMM dataset with $N = 10^5$ points, dimension $d = 2$ and $K = 10$ clusters, and then fits a DPMM to the data (without knowing K or the other parameters of the GMM) using the sampler.

```
using DPMMSubClusters

x,labels,clusters =
    DPMMSubClusters.generate_gaussian_data(10^5, 2, 10, 100.0)
hyper_params =
    DPMMSubClusters.niw_hyperparams(Float32(1.0),
                                    zeros(Float32,2),
                                    Float32(5),
                                    Matrix{Float32}(I, 2, 2)*1)
DPMMSubClusters.dp_parallel(x, hyper_params, Float32(100000.0),
                            100, 1, nothing, true, false, 15,
                            labels)
```

3.4.2 From a C++ code – GPU

`dp_parallel()` in '`dp_parallel_sampling_class`' is the function that should be called in order to run the program. The sample code below will first generate a synthetic random dataset and will then run the sampler to fit a DPMM to it.

```
srand(12345);
data_generators data_generators;
MatrixXd  x;
std::shared_ptr<LabelsType> labels =
    std::make_shared<LabelsType>();
double** tmean;
```

```
double** tcov;
int N = (int)pow(10, 5);
int D = 2;
int numClusters = 2;
int numIters = 100;

data_generators.generate_gaussian_data(N, D, numClusters, 100.0,
                                       x, labels, tmean, tcov);
std::shared_ptr<hyperparams> hyper_params =
    std::make_shared<niw_hyperparams>(1.0, VectorXd::Zero(D), 5,
                                      MatrixXd::Identity(D, D));
dp_parallel_sampling_class dps(N, x, 0, prior_type::Gaussian);
ModelInfo dp = dps.dp_parallel(hyper_params, N, numIters, 1, true,
                               false, false, 15, labels);
```

3.4.3 From the command line – GPU

Running the CUDA/C++ program can be done from the command line (in both Linux and Windows). There are a few parameters that can be used to run the program. In order to set the hyperparams the `params_path` parameter can be used. The value of the parameter should be a path for a Json file which includes the hyperparams (*i.e.* alpha or hyper_params for the prior) . In order to use this parameter follow this syntax:

`--params_path=<PATH_TO_JSON_FILE_WITH_MODEL_PARAMS>`

There are few more parameters like `model_path` for the path to a npy file which include the model and `result_path` for the path of the output results.

Our code has support for both Gaussian and Multinomial distributions. It can be easily adapted to other component distributions, *e.g.*, Poisson, as long as they belong to an exponential family. The default distribution is Gaussian. To specify a distribution other than a Gaussian, use the `prior_type` parameter. For example:

`--prior_type="Multinomial"`

The Json file containing the model parameters can contain many parameters that can be controlled by the user. A few examples are: alpha, prior, number of iterations, burn_out and kernel. The full list of parameters can be seen in the function `init()` in '`global_params`'. The result file is a Json file which by default contains the predicted labels, the weights, the Normalized

Mutual Information (NMI) score and the running time per iteration. A few other parameters can be added to the result file. Samples for these additional parameters are commented out in the main.cpp file.

3.4.4 From Python code – CPU and GPU

The dpmmwrapper.py file contains an example for how to run either the GPU or CPU packages from Python (look for the code in the function `main`). In the following example we are generating a GMM synthetic dataset for 10^5 point, 2 dimensions and 10 clusters. We are running it on the GPU.

```
from julia.api import Julia
jl = Julia(compiled_modules=False)
from dpmmpython.dpmmwrapper import DPMMPython
from dpmmpython.priors import niw

data, gt = DPMMPython.generate_gaussian_data(sample_count=10000,
                                             dim=2, k=10,
                                             var=100.0)
prior = niw(kappa = 1, mu = np.zeros(2), nu = 3, psi = np.eye(2))
labels, clusters, results = DPMMPython.fit(data = data,
                                    alpha = 100,
                                    prior = prior, verbose = True,
                                    gt = gt,
                                    gpu = True)
```

Here, the variable "results" depends on the backend: for the CUDA/C++ backend it will be 'null', while for the Julia backbone "results" will contain other information (see the documentation of our Julia 'fit' function). The "gt" variable stands for the ground-truth labels.

3.4.5 From a Jupiter notebook – CPU and GPU

The notebook dpmmwrapper.ipynb can be executed to test different packages including our GPU and CPU packages on multiple datasets.

3.5 Test

Our CUDA/C++ package was built with the Test Development Driven (TDD) methodology. The Windows version comes with 53 unit tests which cover more than 60% of the code. Usually it takes less than 3 minutes to run all

tests. Those tests are for both Gaussian and Multinomial distributions, and include the different CUDA kernels (for matrix multiplication) mentioned in section 4 below. We used the GoogleTest framework to write those tests and they were validated with a Visual Studio 2019 test engine.

4 The Proposed Implementation

4.1 Design and Implementation

In our implementation(s) after we initialize all the required objects, we start running the iterations of the sampler. For each iteration in `group_step()` we execute the algorithm which is described below in detail. For concreteness, below we will refer to the CUDA/C++ code during the algorithm's description and not to the Julia code. However, the structure of the code in both packages is similar enough to be followed as long as the reader is familiar with any of those languages.

We use the same notation as in [1] where N is the number of data points and K is the (current estimate of the) number of clusters.

- For each iteration of the restricted Gibbs sampling:
  (a) Sample cluster weights $\pi_1, \pi_2, \ldots, \pi_K$:

$$(\pi_1, \ldots, \pi_K, \tilde{\pi}_{K+1}) \sim \mathrm{Dir}(N_1, \ldots, N_K, \alpha). \tag{14}$$

  In our code we built a vector `points_count` that holds the number of points for each cluster and apply a Dirichlet-distribution sampling at once for all clusters:

```
dirichlet_distribution<std::mt19937> d(points_count);
std::vector<double> dirichlet = d(*globalParams->gen);
```

  (b) Sample sub-cluster weights from a 2D Dirichlet distribution:

$$(\bar{\pi}_{kl}, \bar{\pi}_{kr}) \sim \mathrm{Dir}(N_{kl} + \alpha/2, N_{kr} + \alpha/2), \quad \forall k \in \{1, \ldots, K\}. \tag{15}$$

  In order to propose meaningful splits that are likely to be accepted, the algorithm uses auxiliary variables such that each cluster consists of 2 sub-clusters (conceptually thought of as "left" and "right"). $\bar{\pi}_k = \{\bar{\pi}_{kl}, \bar{\pi}_{kr}\}$ denote the weights of the sub-clusters of cluster k. The code is implemented in `sample_cluster_params()` in 'shared_actions'.

(c) Sample cluster parameters:

$$\theta_k \propto f_{\boldsymbol{x}}(\boldsymbol{x}_{\mathcal{I}_k}; \theta_k) f_\theta(\theta_k; \lambda), \quad \forall k :\in \{1, \ldots, K\}. \tag{16}$$

Here, $f_{\boldsymbol{x}}(X; \theta_k)$ is the likelihood of a set of data points under the parameter θ_k, $f_\theta(\theta; \lambda)$ is the likelihood of the parameter θ under the prior $f_\theta(\cdot; \lambda)$, $\propto$ denotes sampling proportional to the right-hand side of the equation, and $\mathcal{I}_k = \{i : z_i = k\}$ is the set of indices of that points labels as belonging to cluster k. Each distribution (*e.g.*, a Gaussian or a Multinomial) has its own `sample_distribution()` function which calculates the cluster's parameters. The prior classes for the Gaussian and Multinomial distributions are '`niw`' and the '`multinomial_prior`' respectively which inherit from the '`prior`' class. The likelihood is calculated in each class within the function `log_marginal_likelihood()`. In order to extend and support to more distributions new classes which inherit from '`prior`' may be added.

(d) Sample sub-cluster parameters:

$$\bar{\theta}_{kh} \propto f_{\boldsymbol{x}}(\boldsymbol{x}_{\mathcal{I}_{kh}}; \bar{\theta}_{kh}) f_\theta(\bar{\theta}_{kh}; \lambda), \quad \forall k :\in \{1, \ldots, K\} \quad , \quad h \in \{l, r\}. \tag{17}$$

$\bar{\theta}_k = \{\bar{\theta}_{kl}, \bar{\theta}_{kr}\}$ denotes the parameters of the sub-clusters of cluster k. In the code we used the same functions that we used for the calculation the cluster's parameters. We maintain the following objects in memory: a `std::vector` of `local_cluster` type where each item in that vector holds the cluster and sub-cluster parameters (*e.g.*, $\bar{\theta}_k$, μ_k, Σ_k,$\bar{\theta}_{kl}$, $\bar{\theta}_{kr}$, $\bar{\pi}_{kl}$ and $\bar{\pi}_{kr}$ for the k^{th} Gaussian), the point counts (*i.e.*, N_k, N_{kl}, N_{kr}) and the sufficient statistics as well as the cluster weights (*i.e.*, $\pi_1, \pi_2, \ldots, \pi_k$).

(e) Sample cluster assignments for each point:

$$z_i \propto \sum_{k=1}^{K} \pi_k f_{\boldsymbol{x}}(\boldsymbol{x}_i; \theta_k) \mathbb{1}(z_i = k), \quad \forall i \in \{1, \ldots, N\}. \tag{18}$$

To sample from a probability distribution efficiently we implemented a GPU kernel in
`sample_by_probability()` based on a C algorithm [15]. To summarize all k of a point we wrote the `dcolwise_dot_all_kernel()` kernel. These kernels are working in parallel on all components. Each component is working in a different GPU

stream while in each stream the points are aggregated to blocks that are running in parallel. We set a number of 512 threads per block. This parameter can be fine-tuned to optimize the performance based on the GPU hardware being used.

(f) Sample sub-cluster assignments for each point:

$$\bar{z}_i \propto \sum_{h\in\{l,r\}} \pi_{z_i h} f_{\boldsymbol{x}}(\boldsymbol{x}_i; \bar{\theta}_{z_i h}) \mathbb{1}(\bar{z}_i = h), \quad \forall i \in \{1, \ldots, N\}. \tag{19}$$

$\bar{z}_i \in \{l, r\}$ variables are (conceptually thought of as "left" and "right") indicate which of the sub-cluster the $i^t h$ point is assigned to. We have a thin version of the cluster and sub-cluster parameters `thin_cluster_params`. For a single machine this structure is unneeded since the data already exists in '`local_cluster`'. However, we maintain it not only for consistency with the Julia package but also because this option may be valuable in the future to scale our CUDA/C++ implementation to multiple machines.

For efficiency we are not copying the data; rather, we use `std::shared_ptr` between the structures. We have also chunks of the data per GPU in the device memory as part of the '`gpuCapability`' structure. The structure contains the following 3 properties: List of points $\boldsymbol{x}_i$ (`d_points`), chunks of the cluster assignments z_i (`d_labels`) and sub-cluster assignments $\bar{z}_i$ (`d_sub_labels`).

```
struct gpuCapability
{
        int* d_labels;
        int* d_sub_labels;
        double* d_points;
};

class cudaKernel
{
protected:
        std::map<int, gpuCapability> gpuCapabilities;
};
```

- Propose and Accept Splits:
  By sampling the sub-clusters, one is able to propose and accept meaningful splits that divide a cluster into its 2 sub-clusters.

  (a) Propose to split cluster k into its 2 sub-clusters for all $k \in \{1, 2, \ldots, K\}$.

(b) Calculate the Hastings ratio H and accept the split with probability $\min(1, H)$:

$$H_{\text{split}} = \frac{\alpha\Gamma(N_{kl})f_{\boldsymbol{x}}(\boldsymbol{x}_{\mathcal{I}_{kl}};\lambda)\cdot\Gamma(N_{kr})f_{\boldsymbol{x}}(\boldsymbol{x}_{\mathcal{I}_{kr}};\lambda)}{\Gamma(N_k)f_{\boldsymbol{x}}(\boldsymbol{x}_{\mathcal{I}_k};\lambda)} \tag{20}$$

The proposal is done in the function `should_split_local()` and the split itself is done in the kernel `split_cluster_local_worker()`.

- Propose and Accept Merges:
  Merges are proposed by merging 2 sub-clusters into one with each of the original clusters becoming a sub-cluster of the new merged cluster.
  (a) Propose to merge clusters k_1, k_2 for all pairs $k_1, k_2 \in \{1, 2, \ldots, K\}$.
  (b) Calculate the Hastings ratio H_{merge},

$$H_{\text{merge}} = \frac{\Gamma(N_{k_1}+N_{k_2})}{\alpha\Gamma(N_{k_1})\Gamma(N_{k_2})}\frac{p(\boldsymbol{x}|\hat{z})}{p(\boldsymbol{x}|z)} \times \frac{\Gamma(\alpha)}{\Gamma(\alpha+N_{k_1}+N_{k_2})} \times \frac{\Gamma(\frac{\alpha}{2}+N_{k_1})\Gamma(\frac{\alpha}{2}+N_{k_2})}{\Gamma(\frac{\alpha}{2})\Gamma(\frac{\alpha}{2})}, \tag{21}$$

and accept the merge with probability $\min(1, H_{\text{merge}})$. The proposal is done in function `should_merge()` and the merge itself is done in the kernel `merge_clusters_worker()`.

4.2 Optimizing Processing

Two kernels for optimization
In order to optimize the performance of the matrix multiplication which is required in our algorithm we used two different CUDA kernels and decide which one to use based on the size of the $d \times N$ matrix (where d is the dimension of the data and N is the number of points). We added a built-in capability to automatically select, at run-time, the best kernel automatically based on d, N and the GPU capabilities. For more optimization in case that the best kernel is known we provide an option to set the kernel which can save some time at the beginning of the run (especially with a high N and a high d). We measured the optimal kernel on an NVidia Quadro RTX 4000 card. On matrices whose size was below 640,000 size ($d \times N$), Kernel #1 is optimized and above this number Kernel #2 is optimized. Kernel #1 was written in a native way for CUDA to multiply two matrices. Kernel #2 was written with cublas API which is more optimal for big matrices.

Data Structure
To optimize the operations that needed to be done on the matrices and in the device, all the data in the GPU and in Eigen's structure are stored in a column-major order. In general, in places which we needed to perform operations on all data points, we iterate on the dimensions and run (in parallel) each point calculation on a different GPU core.

4.3 Parallelism

The sampler from [1] was designed with massive parallelization in mind. Thus, most of its operations are parallelizable. Particularly, sampling cluster parameters is parallelizable over the clusters, sampling assignments can be computed independently for each point, and cluster splits can be proposed in parallel. Thus, a multi-core implementation is quite straightforward, with the caveat that one needs to be careful with merges; *i.e.*, to prevent more than 2 clusters merge into the same single cluster; *e.g.*, if clusters 1 and 2 are merged at the same time when clusters 2 and 3 are merged, this would imply the three clusters (1,2, and 3) would be merged into a single one. However, this would be inconsistent with the underlying model. Recall that [1] released a multi-core single-machine C++ implementation (for CPU). The nature of the algorithm, however, let us build an implementation that goes beyond their original implementation, by allowing parallelization across multiple machines, not just multiple CPU cores. In turn, this enables us to not only leverage more computing power and gain speedups but also provide practical benefits in terms of memory and storage. For example, our Julia implementation can be used within a distributed network of weak agents (*e.g.*, small robots collecting data). It also never transfers data (which is expensive and slow); rather, we transfer only sufficient statistics and parameters. Thus, the proposed implementation is also well suited for a network of low-bandwidth communication. Likewise, on a single machine, we are able to use the GPU's cores powers and Nvidia streams and asynchronous calls.

4.3.1 Multiple GPU Streams

Most of the code was written as separated individual streams. Each stream contains sequences of operations that execute in issue-order on the GPU. CUDA operations in different streams may run concurrently. In many places we tied the stream to a specific cluster. Thus, we could run close to $O(1)$ instead of $O(K)$. We took advantage of CUDA's asynchronous memory allocation API which was exposed lately in version 11. We were able to

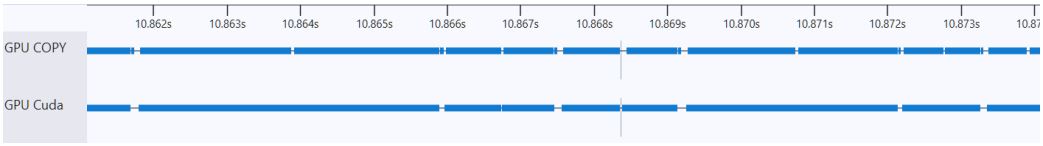


Figure 3: CUDA running in multi streams. Copy and Kernels are working in parallel. Blue color indicates an active kernel. The horizontal axis stands for time (in [sec]).

allocate and deallocate the parameters as part of the sequence of operations and in this way to improve the concurrency of other kernels that were running on other streams. In Figure 3 we show an example of memory allocation operations that were performed on the GPU in parallel and at the same time that the kernel operation was running.

4.3.2 Multiple GPU Cards

We tested the performance of our implementation by parallelizing the processing across multiple GPUs. The logic that we adopt was to parallelize the parts that we were running with multi-streams on multi-GPUs. We implemented a container with all available GPUs: `std::map<int, gpuCapability> gpuCapabilities`. In the places that we sample the cluster and sub-cluster parameters and where the sufficient statistics are been calculated, instead of creating a stream on the same GPU we took the next available GPU by calling to the function `pick_any_device` and created a stream on the current selected device. Our hypothesis was that in places which K is large there should be more impact. Our observation when we tested 2 GPUs (of type Quadro RTX 4000) was that the performance is not better. Consequently, we disabled that option by default in the `cudaKernel::init` function using the following line: `numGPU = 1`. On a different hardware and different GPU types, however, it is possible that this option will yield better results. Thus, the user may want to experiment with codenumGPU ¿ 1, depending on the available hardware.

4.4 Runtime Complexity

We now go throughout the CUDA/C++ runtime complexity step by step based on the algorithm. The symbol G below denotes the number of GPU cores.

For each iteration of a restricted Gibbs sampling: Sampling the cluster and sub-cluster parameters (including weights) takes constant time for

each cluster. In Julia it is parallelized over P processes on the master. This takes $O(K \times d/P)$ time. In CUDA/C++ we observed that using `'omp parallel for'` (for K) when sampling the cluster parameters is slower than performing it sequentially. The root cause is the fact that Eigen (which we rely on for matrix manipulation) already uses multiple CPU cores and it also uses the GPU efficiently in each one of its threads. Thus, running over the K clusters in parallel is slower than sequentially. Therefore the complexity of CUDA/C++ is $O(K \times d)$. Sampling the cluster weights is also done in constant time. Copying cluster and sub-cluster weights and parameters from host to device is parallelized over CUDA streams over K clusters. The process is parallelized over K clusters and over G GPU cores so it takes $O(K \times d/G)$. Sampling the cluster assignments takes $O(N \times K \times T/G)$ time, where T depends on the observation model, *e.g.* for multivariate Gaussian observation model with NIW prior $T = d^2$, wherein for a multinomial observation model with Dirichlet prior $T = d$. Sampling the sub-cluster assignments takes $O(N \times T/G)$. Updating the cluster and sub-cluster sufficient statistics can be split up into 3 steps. The first step is for all streams to calculate the sufficient statistics for the data it is in charge of which is common to all kinds of distributions. This step takes $O(N/G)$ time. The second step is to calculate the sufficient statistics which is unique per the distribution. It is done in parallel to each sub-cluster (l, r) and to the cluster. This step takes $O(N \times T/G)$ time. The third step is to aggregate across all streams. This step takes $O(K \times d)$ time. Overall, this takes $O(N \times K \times T/G) + O(N \times T/G) + O(K)$ time. So total: $O(N \times K \times T/G)$.

Splits: Proposing splits by looking at each cluster is $O(K)$, noting that we can drop the T here as we use previously calculated values. Processing all the accepted splits requires updating the sufficient statistics which could take at the worst case $O(N/G) + O(K)$ if all clusters are split.

Merges: Proposing merges by inspecting each cluster pair is $O(K^2)$, as in the splits, we can drop the T, by using the previously calculated values. Processing all the accepted merges also requires updating sufficient statistics. The worst case (*i.e.*, if all clusters are merged) is thus $O(N) + O(K)$.

To summarize, the total runtime complexity is $O(K) + O(N \times K \times T/G) = O(N \times K \times T/G)$

4.5 Memory Complexity

We now look at the amount of memory used on the GPU. The data is stored as one array, so we have $O(d \times N)$ on the GPU. The amount of data for the labels and sub labels is $O(N)$. Each stream also has a copy of the cluster and sub-cluster parameters which takes $O(K)$ space. We also have to aggregate sufficient statistics for each cluster after sampling the assignments, taking $O(K \times T)$. Hence, we have a total memory usage of $O(d \times N)$. Since usually $N \gg K$, the memory overhead is insignificant in comparison to the data itself.

5 Illustrations

5.1 Synthetic Data: Dirichlet Process Gaussian Mixture Model (DPGMM)

We tested our implementation on small and large synthetic GMM datasets. We wrote a class `data_generators` to generate the synthetic datasets. We ran 112 tests with different parameters: N (10^3,10^4,10^5,10^6), d (2,4,8,16,32,64,128) and K (4,8,16,32). For each test we ran 100 iterations (sufficed for convergence in all methods). We repeated each test 10 times with the same random Gaussian synthetic data that we generated and then averaged the resulting NMI scores and running times. In each test we compared CUDA/C++, Julia, and a Bayesian Gaussian Mixture from sklearn. The latter, like our implementations, infers K with the rest of the model. However, unlike our implementations, this sklearn's model requires an upper bound on K.

During our testing we observed that, on a single machine, our CUDA/C++ implementation is faster than our Julia implementation in most of the cases. The cases in which Julia was faster were when running with a low N (up to 10K) and a low d (up to 32). When the data is larger (*e.g.*, when $N = 10^6$ or $d = 128$) the CUDA/C++ solution is 2.5 times faster than Julia on average. In general, when $N \times d \times K$ is high the CUDA/C++ solution is faster than the Julia solution. Both Julia and CUDA/C++ are faster on average than sklearn solution: on average, Julia was 2.6 times faster and the CUDA/C++ was 5.3 times faster. Moreover, in a few cases the CUDA/C++ was 35 times faster than sklearn.

In Figure 4 we demonstrated the time comparison for $N = 10^6$. In the left side when $d \leq 4$ we see that our solutions were faster than sklearn. Due to the slowness of sklearn for the cases when $d > 4$ we set in the right side

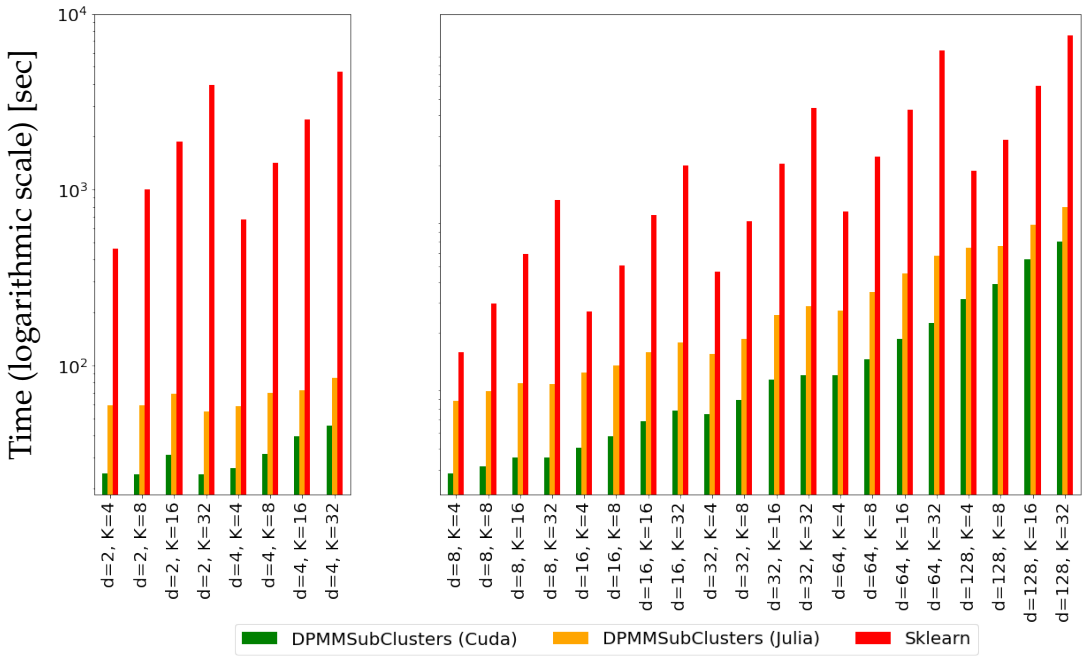


Figure 4: DPGMM synthetic data, time, $N = 10^6$. In the right panel, due to sklearn's slowness, we had to give it the unfair advantage of using the true K as the upper bound of the number of clusters.

of the figure an easier target for sklearn and set sklearn's upper bound for K to the true K (without doing it, sklearn's running time for a high d was impracitically slow, many orders of mangitude slower than our implementations). It is obvious from the figure than even when we gave a big advantage to sklearn, our implementations still beat it. For accuracy we measured the NMI for each case. NMI comparison is displayed in Figure 5. Again we split the test into two parts. The left side of the figure describes a fair comparison where all the methods were given the same conditions. The right side describes the case sklearn enjoyed the advantage of a reduced complexity achieved by using the true K as the upper bound. Despite that advantage, our solutions still yielded better results almost in all cases. All tests can be reproduced by running the dpmmwrapper.ipynb Jupiter Notebook from `https://github.com/BGU-CS-VIL/dpmmpython`. Because of the slowness of sklearn, the notebook as is might take many days. You can consider to reduce the number of permutations that we tested for N, d and K or to reduce the number of repeats.

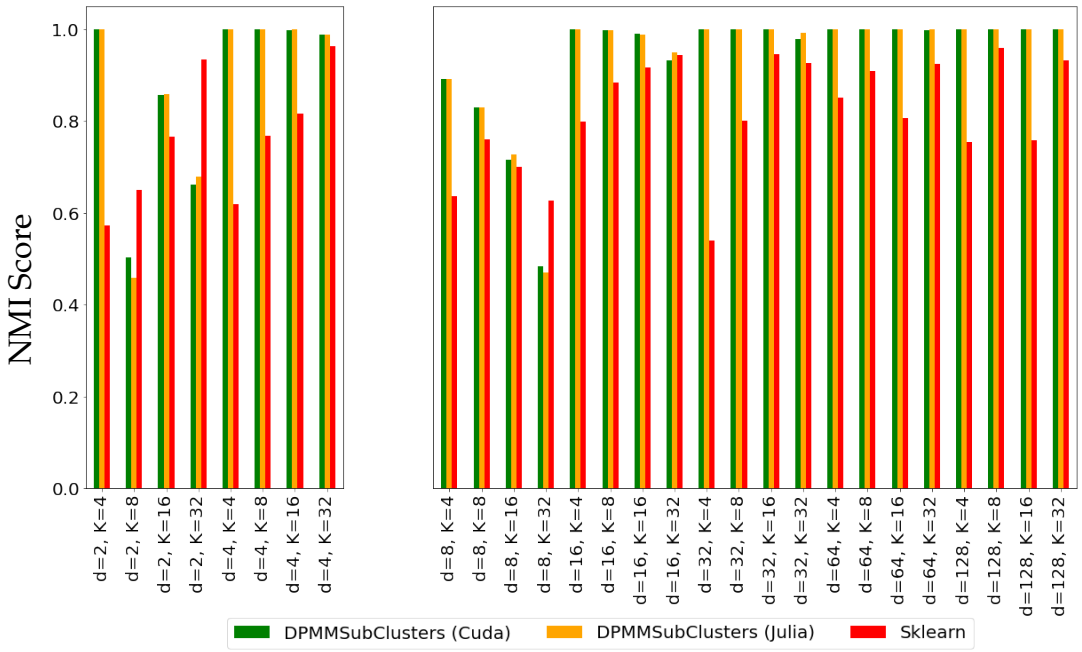


Figure 5: DPGMM synthetic data, NMI, $N = 10^6$. In the right panel, due to sklearn's slowness, we had to give it the unfair advantage of using the true K as the upper bound of the number of clusters.

5.2 Synthetic Data: Dirichlet Process Multinomial Mixture Model (DPMNMM)

We ran 72 tests with different parameters: N (10^3,10^4,10^5,10^6), d (4,8,16,32,64,128) and K (4,8,16,32) while $d \geq K$. For each test we ran 100 iterations (sufficed for convergence). We repeated 10 times each test with the same random multinomial synthetic data and then averaged the NMI results and running times. In each test we compared our CUDA/C++ and Julia solutions to each other (sklearn does not support multinomial components when the number of components is unknown). Unlike in the case of Gaussian components, here, with multinomials, we observed that our CUDA/C++ solution was uniformly faster than our Julia solution. On average the former was 5 times faster than the latter. The results (using $N = 10^6$) are displayed in Figure 6. The corresponding NMI scores apear in Figure 7.

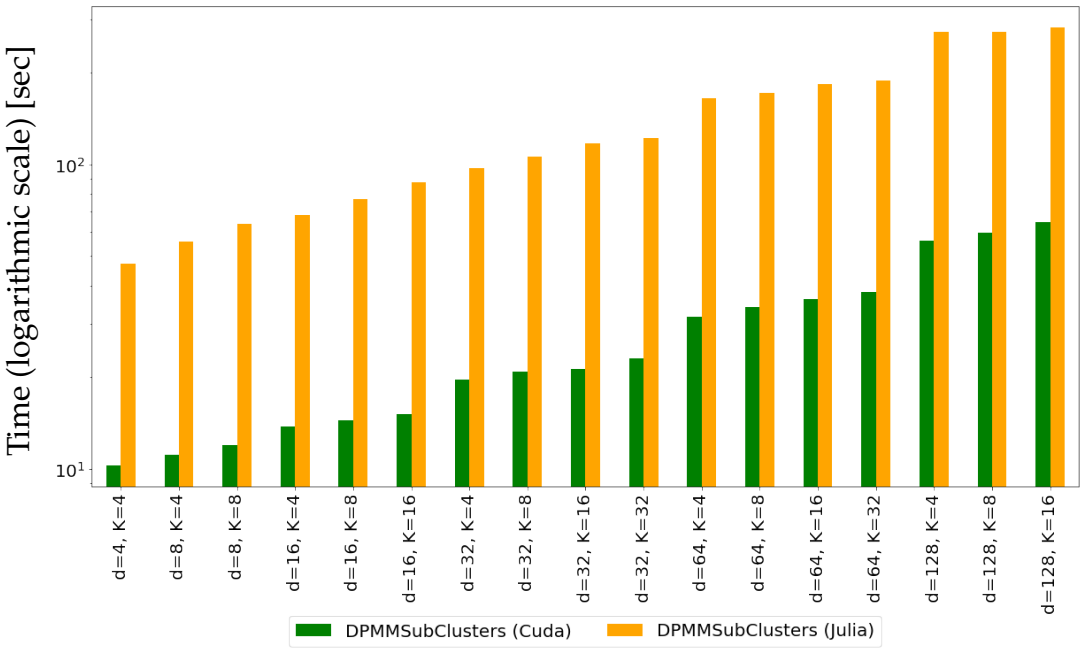


Figure 6: DPMNMM synthetic data, time, $N = 10^6$.

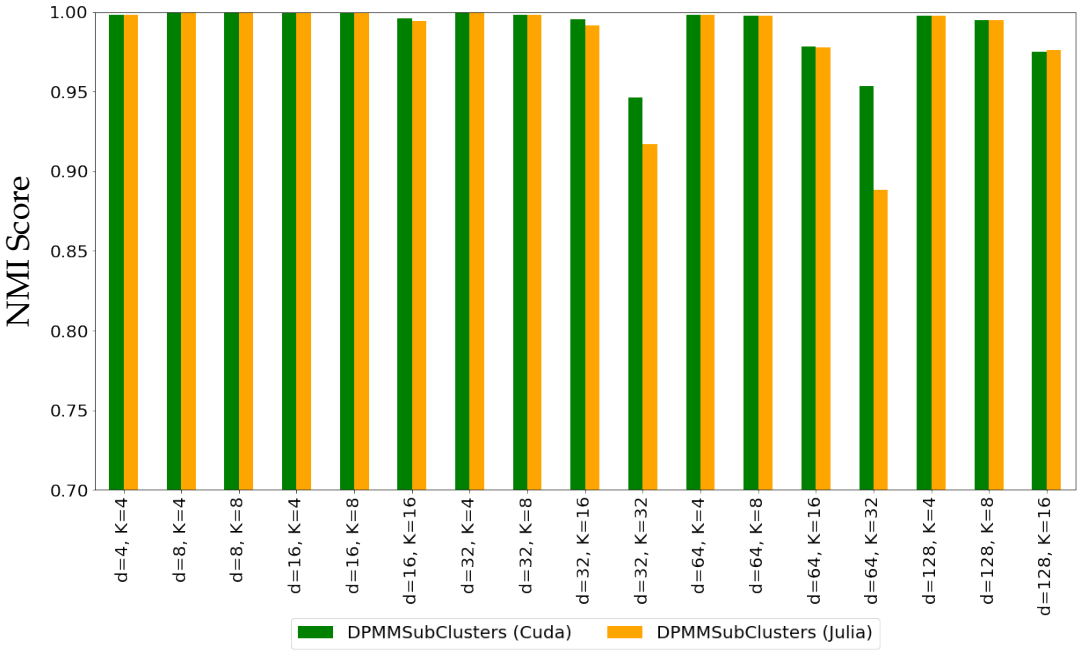


Figure 7: DPMNMM synthetic data, NMI, $N = 10^6$.

5.3 Real Data

We also tested our solution on real datasets, where, as pre-processing, we used Principal Component Analysis (PCA) to reduce the dimensionality. For DPGMM we used the following datsets: **mnist** ($N = 60000$,$d = 32$,$K = 10$); **fashion mnist** ($N = 60000, d = 32, K = 10$) and **ImageNet-100** ($N = 125000$, $d = 64$, $K = 100$). We compared our CUDA/C++ and Julia implementations with sklearn BayesianGaussianMixture. For DPMNMM we used **20newsgroups** ($N = 11314, d = 20000, K = 20$), and compared our CUDA/C++ and Julia implementations to each other (sklearn does not support multinomial components when the number of components is unknown). In the DPMNMM case, when the dimension was very high ($d = 20,000$) the CUDA/C++ package was 188 times faster than Julia. In Figure 8 we compared the running times on each dataset between the different packages. It is clear from the figure that our CUDA/C++ package is much faster from the two others, and that our Julia package is much faster than sklearn. In Figure 9 we displayed the NMI comparison the different packages. On **ImageNet-100**, our CUDA/C++ and Julia packages are almost equal sklearn (with a minor difference of 0.013). In all other cases, our solutions were more accurate than sklearn. Morover, even though, on **ImageNet-100**, our solutions did not score higher NMI values than sklearn, the K value that we predicated was much more accurate. That is, while the true K was 100, sklearn predicated $K = 500$ (which was the value of the upper bound it was given). In contrast we predicted on average $K = 96.8$ (with a standard deviation of 17.78).

6 Summary and Discussion

We extended the DPMM inference method from [1] by efficient and easily-modifiable implementations for high-performance distributed sampling-based inference. Our software can be adopted by practitioners in an easy way where the user is free to choose between either a multiple-machine, multiple-core, CPU implementation (written in Julia) and a multiple-stream GPU implementation (written in CUDA/C++). We also provide an optional Python wrapper which hides the CPU and GPU implementations as a single package with one interface. The proposed implementation supports both Gaussian and Multinomial components. However, it is also easy to extent the code to support other exponential families. We also tested the proposed implementation for learning models from real datasets as mnist, fashion mnist, ImageNet-100 and 20newgroups. We compared our solution

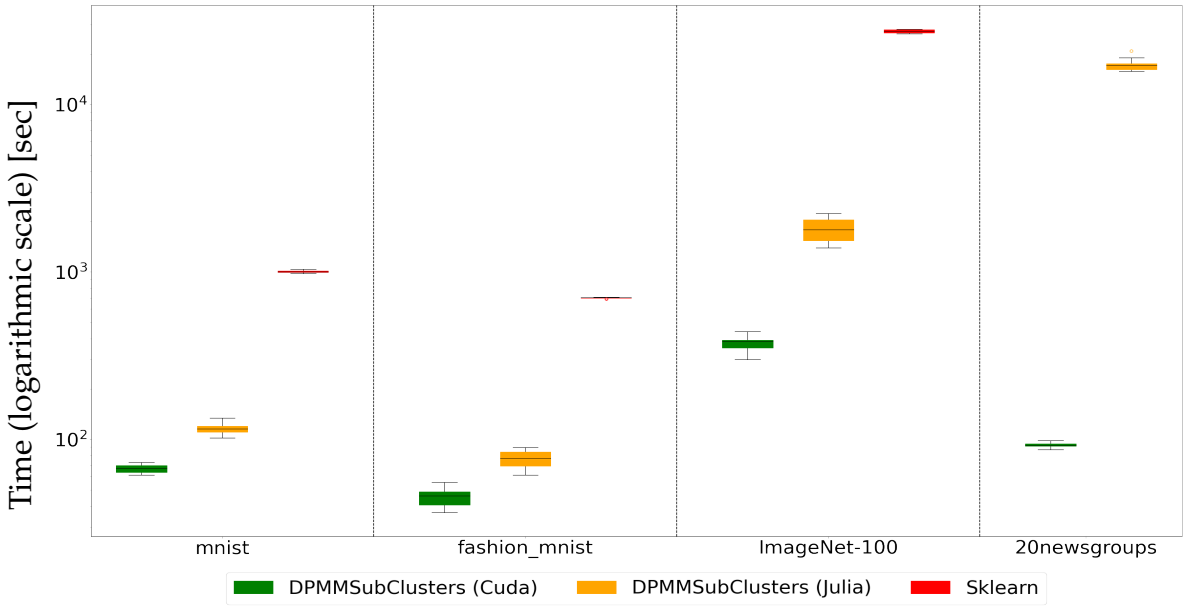


Figure 8: DPMM on real data: running times. In the 20newsgroup dataset (where the data dtype is discrete) the components are Multinomials. In the other datasets the components are Gaussians.

to sklrean and showed that for large and high dimensions our solution achieves the same, or better, accuracy while being much faster. Empirically, we found that when using high-dimensional Gaussians on a single machine our GPU package is faster than any other solution that is currently publicly available. For Multinomial components, our GPU package showed even better results which are by at least two order of magnitude faster than any other existing solution. We provided a solution for cross operating systems platforms (Windows & Linux). Our tests can be easily reproduced via a Jupiter Notebook included with our Python package.

7 Computational details

We ran our tests on two different hardware. One with strong CPU, more RAM and GPU from GTX family: i7-6800K CPU, 3.40GHz with 12 cores, 64GB RAM, GeForce GTX 1070. Second configuration with slower CPU, less RAM and GPU from RTX family: E5-2620 v3 CPU, 2.40GHz with 12 cores, 32GB RAM, Quadro RTX 4000. In both cases we used the same code. We used CUDA driver version 11. We tested on Windows 10 and Linux

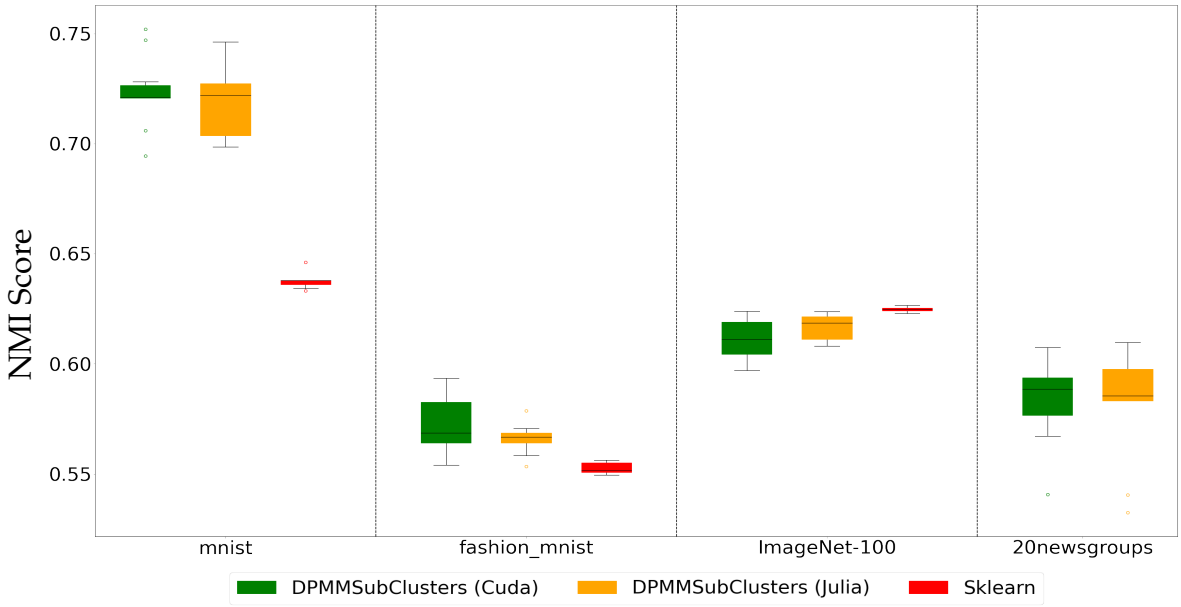


Figure 9: DPMM on real data: NMI scores. In the 20newsgroup dataset (where the data dtype is discrete) the components are Multinomials. In the other datasets the components are Gaussians.

Ubuntu 18.04 and 21.04. The Python version that we used for the wrapper was 3.8. In the CUDA/C++ package for windows we used **OpenCV** version 4.5.2 to demonstrate visualization in 2D of the clustering process iteration by iteration.

8 Open Sources

In Table 3 we specify all the open sources that we use in the **CUDA/C++** package. In Table 4 we specify all the open sources that we use in the **Julia** package.

Open Source	Usage	Link
cnpy	Read and write models from npy format	https://github.com/rogersce/cnpy
zlib	Required by cnpy	https://github.com/madler/zlib
dirichlet_distribution	Dirichlet distribution	https://github.com/gcant/dirichlet-cpp
logdet	Log determinant for Eigen using Cholesky	https://gist.github.com/redpony/fc8a0db6b20f7b1a3f23
vcflib	Log(gamma) and multinormal sampling	https://github.com/vcflib/vcflib
eigen	Matrix and vector operations	https://eigen.tuxfamily.org
stats	Sampling from an inverse-Wishart	https://www.kthohr.com/statslib.html
gcem	Required by stats	https://github.com/kthohr/gcem
jsoncpp	Read and write Json files	https://github.com/open-source-parsers/jsoncpp
MIToolbox	Calculates NMI	https://github.com/Craigacp/MIToolbox
opencv	Drawing (for debugging)	https://opencv.org/

Table 3: **C++** open-source packages used in the proposed implementation

Open Source	Usage	Link
Clustering.jl	Evaluation metrics	https://github.com/JuliaStats/Clustering.jl
DistributedArrays.jl	Distribute Computations	https://github.com/JuliaParallel/DistributedArrays.jl
Distributions.jl	Probability Distributions	https://github.com/JuliaStats/Distributions.jl
JLD2.jl	Saving and restoring checkpoints	https://github.com/JuliaIO/JLD2.jl
NPZ.jl	Compatability with Numpy data	https://github.com/fhs/NPZ.jl
SpecialFunctions.jl	Log Gamma function	https://github.com/JuliaMath/SpecialFunctions.jl

Table 4: **Julia** open-source packages used in the proposed implementation

Bibliography

[1] Chang, Jason and Fisher III, John W. Parallel sampling of DP mixture models using sub-cluster splits. In *NIPS*, 2013.

[2] Wang, Ruohui and Lin, Dahua. Scalable estimation of dirichlet process mixture models on distributed data. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 2017.

[3] Dinari, Or, Yu, Angel, Freifeld, Oren, and Fisher III, John. Distributed MCMC inference in Dirichlet process mixture models using Julia. In *IEEE CCGRID Workshop on High Performance Machine Learning*, 2019.

[4] Dempster, Arthur P, Laird, Nan M, and Rubin, Donald B. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 1977.

[5] Gelman, Andrew, Stern, Hal S, Carlin, John B, Dunson, David B, Vehtari, Aki, and Rubin, Donald B. *Bayesian data analysis*. Chapman and Hall/CRC, 2013.

[6] Ghosal, Subhashis and Van der Vaart, Aad. *Fundamentals of nonparametric Bayesian inference*. Cambridge University Press, 2017.

[7] Müller, Peter, Quintana, Fernando Andrés, Jara, Alejandro, and Hanson, Tim. *Bayesian nonparametric data analysis*. Springer, 2015.

[8] Sudderth, Erik Blaine. *Graphical models for visual object recognition and tracking*. PhD thesis, Massachusetts Institute of Technology, 2006.

[9] Chang, Jason. *Sampling in computer vision and Bayesian nonparametric mixtures*. PhD thesis, Massachusetts Institute of Technology, 2014.

[10] Antoniak, Charles E. Mixtures of Dirichlet processes with applications to bayesian nonparametric problems. *AoS*, 1974.

[11] Pitman. Combinatorial stochastic processes. Technical report, Technical Report 621, Dept. Statistics, UC Berkeley, 2002. Lecture notes, 2002.

[12] Robert, Christian and Casella, George. *Monte Carlo statistical methods*. Springer Science & Business Media, 2013.

[13] Jain, Sonia and Neal, Radford M. A split-merge markov chain monte carlo procedure for the dirichlet process mixture model. *Journal of computational and Graphical Statistics*, 13(1):158–182, 2004.

[14] Hastings, W Keith. Monte Carlo sampling methods using Markov chains and their applications. 1970.

[15] Smith, Warren D. How to sample from a probability distribution. Technical report, Technical Report DocNumber17. NEC Research, 2002.

Inferring Compositional Behavioral Models Using Transformations and Automata Learning

Tom Yaacov and Gera Weiss

Ben-Gurion University of the Negev

Model-driven engineering is an established set of tools and methodologies for empowering software engineering tasks using formal models. A central challenge embedded in these methods is how to ease the effort of building models, possibly even automate it completely by constructing models from data. Specifically, the focus of our work is on constructing automata models from system logs. This can be viewed as a grammatical inference task, the process of learning formal grammar from a set of observations. Current state-of-the-art grammatical inference methods include state merging techniques [3, 4], genetic algorithms [2], and recurrent neural networks [1].

A key aspect of building robust and reliable behavioral models, not fully realized by the above-mentioned techniques, is the ability to capture behavioral patterns in a clear and meaningful way. Consider, e.g., a modeler trying to capture the behavior of a chess player or an online store website. We would like the modeling technique to allow the capturing of patterns such as "a piece cannot move if its movement exposes the king to a direct attack" or "A user cannot add products to the cart before login". Our goal is to produce models that explicitly contain such behavioral rules in a compositional way. Furthermore, we aim for models that are easy to maintain in the sense that users can incrementally add rules and exceptions either manually or automatically. This requires our model to not only be accurate but also be explainable and allow effective analysis of its behavior. Our research exploits the growing body of research in machine learning and automata theory. Specifically, we extend current automata learning methods to identify different patterns that allow the implementation of new, far more capable, behavioral models.

To give the reader an idea of the type of challenges that we are investigating, we provide a simple illustrative example. Take the regular language $L = L_1 \cap L_2 \cap L_3$ over the alphabet $\{A, B, C\}$ where

$$L_1 = \{w\colon w \text{ contains an even number of } A\text{s}\}$$

$$L_2 = \{w\colon w \text{ contains an even number of } B\text{s}\}$$

$$L_3 = \{w\colon w \text{ contains the sub-string } BB\}$$

The goal of this exercise is to be able to construct this model from examples, i.e., a set of words labeled by a Boolean flag indicating whether the word is in L. Current state-of-the-art methods, e.g., in automata learning can construct an automaton for L, but this automaton is for the intersection of the three languages which means that it is relatively big (12 states) and it does not reflect

the compositional structure of L, as illustrated in Figure 1. Instead, we propose to apply transformations to the words and then use automata learning algorithms to learn automata for the transformed languages. Specifically, if we apply the transformations T_1 that deletes all letters that are not A, we can learn a two-state automaton for L_1, as shown in Figure 2. Currently, we propose to use a rich set of transformation functions, to apply each to the data-set, and ones that yielded small automata. This is reasonable, for example, for explaining software failures that are usually produced by tests with specific patterns. In future research, we plan to use genetic algorithms (GA) for evolving transformations that are grown as combinations and mutations of successful ones.

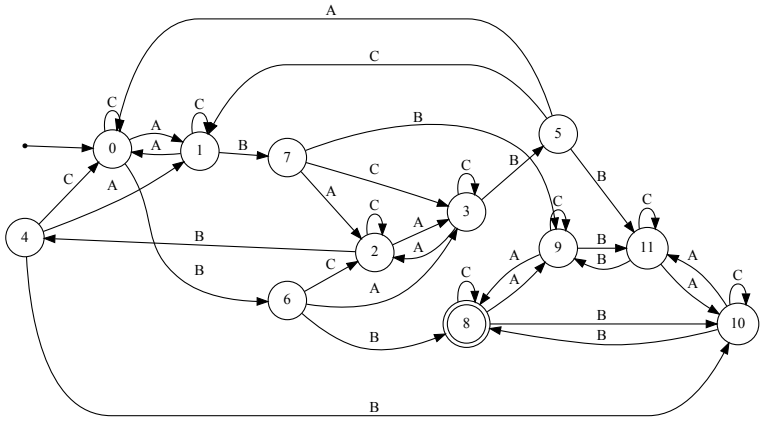
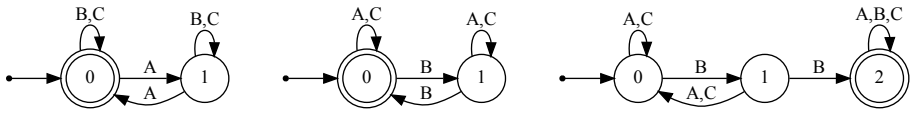


Fig. 1: The 12 states automaton of L. It does not reflect the compositional structure of L as described.



(a) L_1 (b) L_2 (c) L_3

Fig. 2: The set of automata composing L. By applying transformations to the data we can learn different patterns of L in a compositional way.

References

1. Cleeremans, A., Servan-Schreiber, D., McClelland, J.L.: Finite State Automata and Simple Recurrent Networks. Neural Computation **1**(3), 372–381 (09 1989). https://doi.org/10.1162/neco.1989.1.3.372
2. Huijsen, W.O.: Genetic grammatical inference. In: CLIN IV: Papers from the Fourth CLIN Meeting. pp. 59–72. Citeseer (1993)
3. Lang, K.J., Pearlmutter, B.A., Price, R.A.: Results of the abbadingo one dfa learning competition and a new evidence-driven state merging algorithm. In: International Colloquium on Grammatical Inference. pp. 1–12. Springer (1998)
4. Oncina, J., Garcia, P.: Identifying regular languages in polynomial time. In: Advances in structural and syntactic pattern recognition, pp. 99–108. World Scientific (1992)

Survey on Javascript Engines Fuzzers

Lucas Bichet[1], Guillaume Guerard[1], and Soufian Ben Amor[2]

[1] Léonard de Vinci Pôle Universitaire, 92916 Paris La Défense, France
[2] LI-PARAD Laboratory EA 7432, Versailles University, 78035 Versailles, France
guillaume.guerard@devinci.fr

Abstract. The security of Web users is becoming a major issue today. Thus, Web browsers and more particularly the JavaScript engines that compose them are studied carefully to detect and promptly correct the vulnerabilities that they embed. This article presents the current state of research on vulnerabilities affecting JavaScript engines, starting from the current research context to the research tools used.

Keywords: Fuzzing · JavaScript Engine · Code Coverage.

1 Introduction

Since the end of the 1990s, the Internet has been spreading throughout the world at tremendous speed. It is the advent of the mass access, all the homes are then progressively equipped with an Internet connection. Software facilitating the navigation on the Web appears: the web browsers. Although these were very specialized at the beginning, they swiftly became extremely complex software, compatible with many protocols, specifications and interpreted languages.

These software are nowadays extremely used by both personal computers and phones, which makes them critical from the point of view of the security of users on the Internet. In particular, the JavaScript engines embedded in all web browsers nowadays are regularly subject to vulnerabilities discovered by researchers or exploited in the wild.

The notion of vulnerability (in the IT sense of the term) is well-known by the users. However, it is not as evident as it seems at first glance, and it is not always easy to explain what constitutes a vulnerability or not. A vulnerability can be described as an expected or unexpected feature that introduces a risk to the user's security, or to the target's continuity of operation.

Vulnerabilities have in common the fact that they have an impact on the security of users by definition. The method used to exploit the vulnerability and produce something interesting and concrete from the attacker's point of view is called "exploit" or "exploit code".

We will present the basics of the functioning of JavaScript engines to underline the research axes in term of security, then we will come back in more detail on the methods and tools accompanying the research of vulnerabilities on this subject.

Among the many software testing techniques available today, fuzzing has remained highly popular due to its conceptual simplicity, its low barrier to deployment and its vast amount of empirical evidence in discovering real-world software vulnerabilities [24]. Fuzzing refers to a process of repeatedly executing a program with generated inputs that may be syntactically or semantically malformed.

This paper focuses on the vulnerabilities through the JavaScript engine, how to detect the vulnerabilities. Following the literature review, we propose various axes of research to improve the vulnerabilities finding through fuzzer's enhancement.

The paper is built as follows: the section 2 exposes the notion of vulnerabilities and their consequences for a user. The section 3 presents the JavaScript language and engine. The section 4 shows the current research about how to discover vulnerabilities. The section 5 presents research axes to improve the researches. The section 6 concludes this paper.

2 Vulnerabilities

It is meaningful to distinguish bugs and vulnerabilities: a bug represents a programming error in computer code that makes it works in an unexpected way. Bugs can sometimes (but not necessarily) introduce vulnerabilities, while vulnerabilities can sometimes be the result of bugs but not necessarily either.

For example, a bug allowing a buffer stack overflow (writing outside the limits of a previously allocated buffer) probably introduces a vulnerability, while a bug causing an inability to read a file presumably does not.

In fact, the notion of vulnerability is subjective: it strongly depends on the context of the target as well as on the context of the target's users. For example, in the context of industrial activity monitoring software, the ability to remotely interrupt the software bears devastating consequences and therefore introduces an obvious security risk.

It is therefore legitimate to consider such an ability as a vulnerability. In the context of a Web browser, the ability to interrupt the software remains an annoying bug, but the security risks involved are limited: the notion of vulnerability is therefore probably unappropriate here.

2.1 Types of vulnerabilities

Since the notion of a vulnerability has been clarified through examples, we note typical characteristics that allow us to classify them by type such as:

1. Vulnerabilities known as injections [18, 34]: these are vulnerabilities induced by a misuse of inputs provided by a user. Typically, many Web vulnerabilities are injections that frequently come from the use of untrusted information provided by a user in an inappropriate context: SQL injections, JavaScript or XSS injections, etc.

2. Memory corruption vulnerabilities [5, 33]: these are low-level vulnerabilities occurring at the process memory management level. These vulnerabilities allow the arbitrary modification of the process memory by the attacker. For example, heap overflows are the best-known memory corruption, but many other vulnerabilities exist (use-after-free, type confusion or stack overflow [19]). These are the most common vulnerabilities found in JavaScript engines.
3. Logical vulnerabilities: these are vulnerabilities caused by a logical problem in the design of the application. These are habitually vulnerabilities on a larger scale (the entire information system). For example, an identification made through supposedly private data at one end of the information system, while this information can be gleaned publicly (and intentionally) elsewhere in the system.
4. Hardware vulnerabilities: these are vulnerabilities introduced by the hardware supporting a system or an application. This can be related to the design of the processor (Spectre, Meltdown vulnerabilities for example [30]), to the presence of an accessible console port offering elevated privileges on connected objects, etc.
5. Other types: vulnerabilities related to network protocols, access rights, etc.

2.2 Consequences of vulnerabilities

Vulnerabilities have in common the fact that they have an impact on the security of users by definition. The method used to exploit the vulnerability and produce something interesting and concrete from the attacker's point of view is called *exploit* or *exploit code*. In that manner, many ways to impact security can be grouped by:

1. Arbitrary code execution: the attacker obtains the ability to execute the code he requires on the targeted machine through the vulnerable application. Arbitrary code execution is one of the goals of an attacker targeting a JavaScript engine.
2. Exfiltration of private data: the attacker obtains the ability to gather data that he should not obtain access to (Cookies, passwords). The exfiltration of private data (like Cookies) can also be part of the goals sought by an attacker targeting a JavaScript engine.
3. Elevation of privileges: the attacker obtains the ability to perform privileged actions that he was unallowed to perform.
4. Denial of service: the attacker obtains the ability to alter the operation of the targeted vulnerable service. He can therefore interrupt it, slow it down, etc.

3 JavaScript

Let's focus on the JavaScript language and engine. JavaScript is an extremely widespread non-typed scripting language, initially used to make web pages dynamic. It has a significant need for performance, and this need tends web applications to rely on this language.

Because of its simple syntax, its asynchronous code mechanisms and its performance, JavaScript is equally spreading outside the Web and inviting itself in many Desktop applications via *Node.js* (derived from JavaScript) and *Electron.js* (a technology allowing porting a Web application in JavaScript in a Desktop application).

3.1 Javascript engines

A JavaScript engine is a software that interprets JavaScript code and allows its execution. JavaScript engines are part of the essential components of browsers with the rendering engines, the user interface, etc. There are several of them, used by different browsers:

1. JavaScriptCore[3] is used in Safari and developed by Apple. The JavaScript engine of Safari was also formerly called Nitro and SquirrelFish.
2. V8 is used in Google Chrome, Chromium (which itself represents the basis for many browsers such as Microsoft Edge, recent versions of Opera, Brave, Google Chrome) and Node.js in particular. It is developed by Google. Ross McIlroy, a well-known Google researcher, perfectly describes the V8 in the dedicated blog[4].
3. Chakra[5] is used in Internet Explorer 9 and developed by Microsoft.
4. SpiderMonkey[6] is used in Firefox and is developed by Mozilla.

The vast majority of JavaScript engines work in the same way and differ merely in their implementation (see Figure 1). These software are developed in C++ for the most part and are composed of two essential parts, the interpreter and the optimizer, and specific data structures (commonly called hidden classes, maps, shapes, etc.). Vulnerabilities can therefore be present at various levels of the JavaScript engine.

The information above is not intended to explain the functioning of a JavaScript engine but simply to highlight its key components, necessary for a proper understanding of the subject.

Javascript interpreter The JavaScript interpreter converts the JavaScript source code into an abstract syntax tree (AST) and generating the associated bytecode. This bytecode is then executed in a JavaScript virtual machine transcribing it on-the-fly into native machine code as shown in the Figure 2.

The interpreter is in charge of collecting information about the execution of the program such as the number of times a function is called, what type of arguments are passed to a function, etc. These data are stored in *caches* used by the JIT compiler during the optimization of the code.

[3] https://trac.webkit.org/wiki/JavaScriptCore
[4] https://v8.dev/
[5] https://github.com/chakra-core/ChakraCore
[6] https://www.mozilla.org/fr/js/spidermonkey/

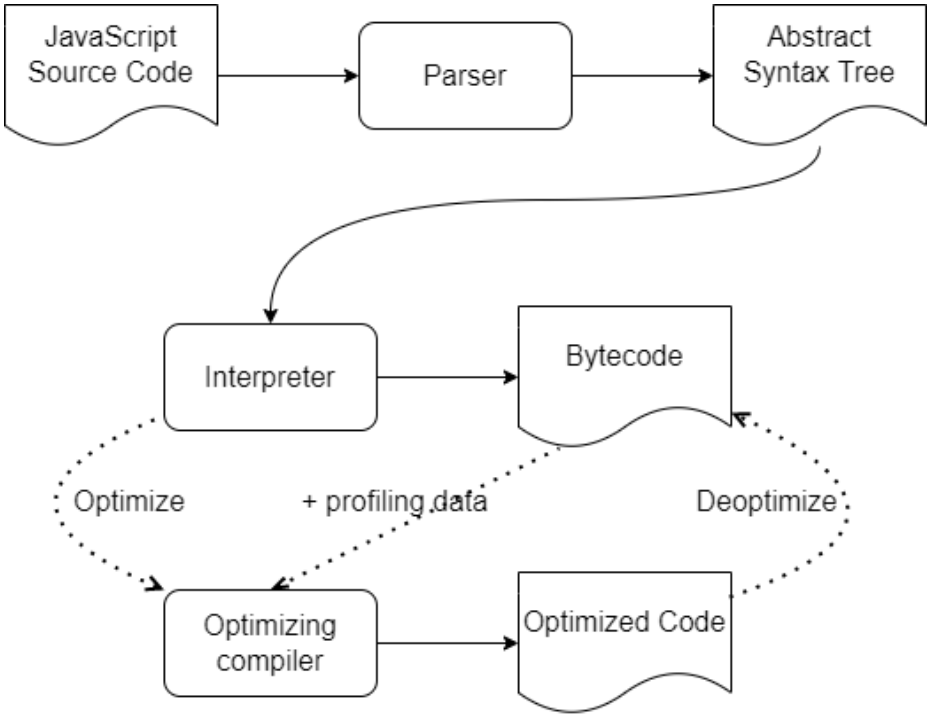


Fig. 1. General organization of a Javascript engine.

Javascript Optimizer & JIT Compiler The purpose of the JavaScript optimizer is to remove redundant parts of the code, and to translate the JavaScript bytecode of the functions very often executed into native machine code, which can be executed directly by the processor as presented in the Figure 3.

It is based on a compiler *Just-in-time (JIT)* or on-the-fly compiler that breaks down the optimization into several passes, which are specific to each JavaScript engine [35]. The information collected by the interpreter makes it possible to establish whether a function needs to be optimized or not: in the positive case, the JIT compiler converts the function into the form of a graph (as *sea of nodes* for V8 [29]) and proceed to the simplification of the graph of the function in several phases. For example, if it was noted that function A only received objects of a class B as arguments, function A will be specifically optimized for objects of class B. However, it is necessary to check that the arguments received by function A are always of type B to be able to deoptimize the function when its input arguments change and switch back to the non-specialized version of the function executed by the interpreter [31].

Data structures The principal difficulty of these engines is to succeed in recognizing the type of the data used, since JavaScript is untyped. For this, a data

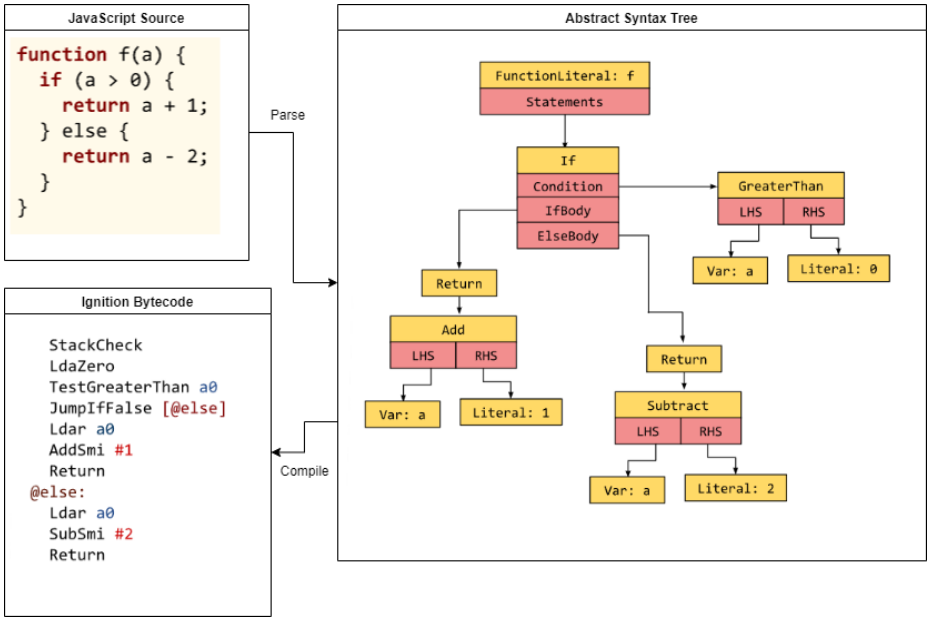


Fig. 2. Generation of Javascript bytecode from source code.

structure is associated with the set of data used and contains all the information on the type of the data in question: its properties, its size, its accessors, etc. This data structure is found in all engines under different names: *Maps* in v8, *Shapes* in JavascriptCore, *HiddenClasses* more generally.

Many other optimizations are embedded in the code of some JavaScript engines, such as compression and pointer tagging for example.

3.2 Types of vulnerability

The most represented vulnerabilities found in Web browsers are memory corruptions like heap overflow [33] and integer overflow [15]. Some vulnerabilities are relatively specific to JavaScript based software, given their particular mode of operation. The confusion vulnerabilities [2] represent a very good example, since they only happen when a software is mistaken about the type of the data it manipulates, a situation that is not very frequent in usual codes but that happen in JavaScript engines given their specific framework.

The interpreter and the JIT mentioned above as well as their numerous subparts are likely to contain vulnerabilities. However, it appears that most of the vulnerabilities come from the JIT compiler, which is a much more complex piece of software than the interpreter. To be specific, type confusion frequently occurs as a result of errors in the optimization or deoptimization of executed functions.

However, the exploitation of JavaScript engine's vulnerabilities and related techniques will be uncovered in this document, as it is a separate and complex

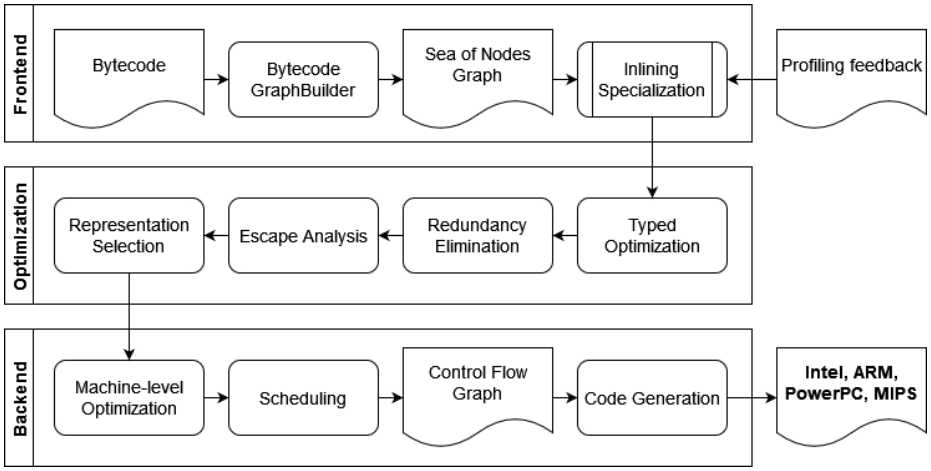


Fig. 3. Optimization of Javascript code (here with v8) ©JJY-security.

topic. There are known methods [13, 14, 32] allowing to convert a bug into an arbitrary code execution, most of them work as follows:

1. Triggering the vulnerability and manipulating the memory to corrupt a JavaScript object.
2. Obtaining primitives *addrof* and *fakeobj* allowing respectively to obtain the address in memory of a given JavaScript object and to construct an arbitrary fake JavaScript object in memory.
3. Obtaining arbitrary read and write primitives via the creation of a fake JavaScript array where its data pointer is modified thanks to the preceding primitives.
4. Execution of arbitrary code by the preceding primitives using various techniques to obtain RWX pages in memory (function compiled by the JIT or WebAssembly function).

JavaScript is a widespread language. Its specification is constantly evolving and JavaScript engines are constantly being equipped with recent and experimental features that are interesting areas of vulnerability research. One example is WebAssembly (WASM), which is a pseudo assembly language that offers an intermediate representation between JavaScript code and machine code while remaining independent of the underlying hardware. This feature is examined from a security point of view because it brings many new features to the code base and thus new horizons in terms of vulnerability research [16].

3.3 Vulnerability scanning techniques

There are essentially two principal ways to seek for vulnerabilities in web browsers and more generally in software.

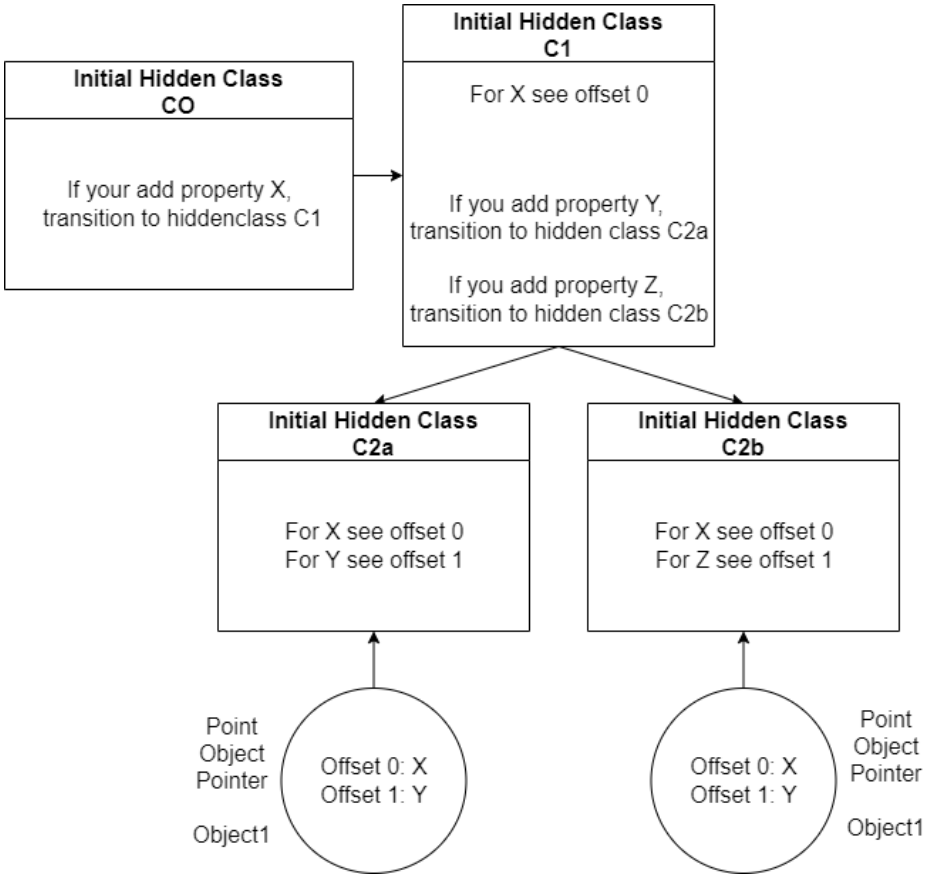


Fig. 4. Example of an HiddenClass containing information related to the type of the manipulated data.

Firstly, the manual code review consists in methodically reading the source code or the native code if the project is not open-source and trying to perceive errors. This technique is effective, but it is extremely time-consuming and tedious. However, it additionally allows for identifying undetectable details with the second technique, because a trained human eye will be skilled to assume the places where mistakes could have been made.

Secondly, the *fuzzing* consists of automatically generating inputs and submitting them to the target software. It is enough to monitor how the software behaves (crash, slowness, unusual behavior) to isolate inputs that probably lead to vulnerabilities. This technique is often very efficient but less accurate. It is equally necessary to configure a tool or even to develop one when striking a very specific target. The inputs are chosen or altered thanks to various techniques like genetic algorithms.

Fuzzing was introduced by Miller et al. [26] for evaluating the robustness of UNIX utilities against unexpected inputs. The difference between fuzzing and other black-box test generation methods is that fuzzing relies on very weak oracles-checking only for crashes or hangs, which lets it explore the input space automatically.

The optimization of the functioning of a fuzzer is a very vast domain: we can try improving the mutation algorithms generating the inputs, find new metrics to evaluate the impact of the tested input, improve the speed of the fuzzer, etc. We will therefore present in the following section the techniques and tools of fuzzing mostly used for the study of JavaScript engines.

4 Fuzzing of JavaScript engines

There are many well-known fuzzers whose efficiency has been proven. Some are intended to be non-specialized such as AFL++ [9], which is also very modular and therefore very reused. But in some cases, the target program taken as input data is structurally complex that a general fuzzer becomes insufficient and therefore useless. Indeed, when the generation of a valid input for the program is non-trivial, it becomes necessary to develop specially adapted fuzzing tools.

In the case of the study of a JavaScript engine, the generation of valid JavaScript code is essential if one wants to be able to fuzz in-depth the program, and in particular its JIT compiler. Indeed, if the JavaScript codes passed in the input are invalid, they will not even pass the stage of *parsing* and *interpretation*. Thus, it will never be executed and even less optimized by the JIT compiler. We will therefore only refer to the submerged part of the iceberg that is the parser integrated into the JavaScript engine.

We observe that all the solutions of fuzzing employed today to answer these constraints are more or less based on the use of the JavaScript grammar for the generation of code and the use of the code coverage as a metric to guide the fuzzing through mutation of the generated code or grammar [6, 24]. Although all JavaScript engine fuzzers are based in some way on grammar, their usage can vary depending on the situation.

Two classes of fuzzers have been developed, *generative* and *mutational* fuzzers. Generative approaches build a new test case from the ground following predefined rules like a context-free grammar of the JavaScript programming language or reassembling synthesizable code bricks dissected from the input corpus; mutational approaches synthesize a test case from existing seed inputs and adapt them for upcoming tests.

4.1 Fuzzing & feedback

In addition to the result of execution, it is conventional for fuzzers to utilize methods to obtain additional feedback after each execution to better guide the fuzzing.

We can separate fuzzers into several distinct categories based on the type of feedback they use:

1. The *white box fuzzers* recover during the execution the exact conditions to which the input is subjected. They modify the input in a suitable way to pass new conditions in the subsequent executions. The white box fuzzing is based on binary instrumentation techniques such as symbolic execution [11].
2. The *grey box fuzzers* only retrieve some information such as the code coverage: for each entry, they retrieve the code paths in the AST reached by this execution to be able to guide the next executions and encourage the exploration of new paths [37].
3. The *black box fuzzers* do not use any additional feedback. They use the target program as a black box and do not analyze how inputs to the program are processed. As a result, they are frequently faster than the first two but less efficient.

The vast majority of current fuzzers are grey-box fuzzers using code coverage as the primary metric. JavaScript fuzzers are no exception to this trend, and use code coverage as a clue to the interest of an entry for fuzzing.

4.2 News on Javascript's Fuzzers

We have chosen to focus on several fuzzers from different Javascript engines, in order to list and compare the notable features they deploy. These fuzzers have been chosen because they have produced good results, are recent and each introduces different novelties. There are many other interesting fuzzers [1, 7, 28, 36, 37], some of them will illustrate the next section about axes of research.

CodeAlchemist Presented in 2019, CodeAlchemist [17] exploits the notion of semantic validity in the generation of JavaScript code. Indeed, by relying on grammar or by applying mutations to pre-selected codes (called seeds), they guarantee the produced code will be syntactically valid i.e. it will be considered as valid code, but nothing guarantees its validity in a precise context. The sequence of lines of code may not produce any sense, and this is precisely what CodeAlchemist intends to prevent. It extracts code bricks from the seeds provided as input, and associates constraints to them as rules to follow when connecting blocks together. The fuzzer is thus separated into three parts: the seeds parser, the constraints analyzer, and the fuzzer itself.

Deity Introduced in 2019, Deity [23] employs the notion of fuzzing guided by abstract syntax tree mutation (AST). In fact, Deity manages a abundant number of seeds for its mutation algorithm. These seeds come from JavaScript code recovered on the Internet but also from newly discovered vulnerabilities since we can observe they are often triggered by the same exploits. The idea is to convert these JavaScript codes into AST as an interpreter would do and to operate mutations directly on this AST: deletion of nodes, merging of several paths coming from various codes. Once the mutation has been applied, the algorithm redoes the reverse conversion to recover valid and mutated JavaScript source codes, which are submitted to the targeted engine.

EvoGFuzz Presented in 2019, EvoGFuzz [8] has the particularity to apply mutations not on codes generated by a grammar, but directly on the grammar itself. It requires seeds and a file describing the JavaScript grammar. At the start, it will generate probabilistic grammar by isolating the most used grammar rules in the seeds. Then, a population of codes to submit to the JavaScript engine is generated, and a genetic algorithm is used to select the most interesting entries. The structure of these entries will then influence the defined grammar, and thus the next generation of entries.

FuzzILi Presented in 2018, FuzzILi [12] is a fuzzer also employing a mutative approach. It has the particularity to use an intermediate representation language specially created for the occasion: FuzzIL. This language takes the form of a both easily mutable and easily convertible bytecode into JavaScript code. Then, they exert several kinds of mutations to it: *input mutator* changes the variables used by the instructions, *combine mutator* combines two already existing intermediate representation languages. To conclude, the code coverage provides information about the interest of the newly created mutation.

Montage Presented in 2020, Montage [21] starts with two interesting observations:

- Vulnerabilities often come from files previously patched against other vulnerabilities
- Code fragments that trigger vulnerabilities frequently reuse code snippets from existing test code.

Based on these observations, Montage uses a Neural Network Language Model (NNLM) to learn the links between the code fragments found in the test sets. The fuzzing is done in three phases:

- Firstly, it consists of extracting code fragments from the existing test sets.
- Then, the model trains the NNLM on the basis of the extracted fragments.
- Finally, the model generates newly potentially vulnerable codes thanks to the NNLM previously trained.

5 Research axes

Fuzzing is the most popular approach for discovering vulnerabilities in JavaScript engines. Many fuzzers exist and have already been proven, but most of the time they are only made public when they have already been overexploited on known open-source JavaScript engines. It is therefore possible to draw inspiration from them, to reuse them on other open-source engines, but the exploration of creative ideas and tracks remains essential to discover more vulnerabilities. We identify five potential axes of improvement.

5.1 Hybrid methods

Hybridization consists in designing a new tool by combining several projects and in particular the novelties introduced by the different fuzzers presented. The idea is to benefit from the advances of fuzzers, to complete others, and thus obtain better efficiency in the fuzzing.

Outside JavaScript fuzzing, Yun et Al. propose QSYM [39], a hybrid fuzzer for real-world programs' binaries, which uses Dynamic Binary Translation (DBT) to natively execute the input binary as well as to select basic blocks for symbolic execution. The DBT produces basic blocks for native execution and prunes them for symbolic execution, allowing to switch between two execution models. Then, QSYM selectively emulates only the instructions necessary to generate symbolic constraints. The fuzzer is used on the LAVA-M dataset and outperformed bug-finding tools like Driller and VUzzer.

Zhao et Al. observe hybrid fuzzing which combines fuzzing and concolic execution has become an innovative technique for software vulnerability detection (especially on binaries). They propose DigFuzz [40]. They design a novel Monte Carlo-based probabilistic path prioritization model to quantify each path's difficulty and prioritize them for concolic execution. They test their fuzzer on the LAVA dataset and outperformed Driller and AFL.

Kim et Al. work on Linux kernel [20]. They propose a hybrid fuzzing with three features: 1) converting indirect control transfers to direct transfers, 2) inferring system call sequence to build a consistent system state, and 3) identifying nested arguments types of system calls. The proposed HFL fuzzer outperformed Moonshine and Syzkaller, the main fuzzers on Linux kernel.

To conclude about hybrid methods, the goal of those methods is to infer a program model or input grammar from either observing the behavior of the program on multiple inputs, using formal approaches, machine learning based on a previously available corpus, or observing and summarizing the program execution.

5.2 Algorithm improvements

Most of the fuzzers have drawbacks. Lexical approaches such as traditional fuzzing fail because of the sheer improbability to generate valid inputs and keywords, whereas the symbolic constraint solving of semantic approaches fail due to the combinatorial explosion of paths. Therefore, researchers aim to improve algorithms to avoid invalid inputs or paths.

Improving the algorithms consists of changing the metrics and the different algorithms used in the different parts of the fuzzer:

- Genetic algorithms for choosing which entries to keep according to their code coverage
- Input mutation algorithms
- Seed selection algorithms
- Etc.

Considering JavaScript engine, Park et Al. [28] propose a new fuzzer DIE, including a new technique called an aspect-preserving mutation, that stochastically preserves beneficial properties and conditions of the original seed input in generating a new test case.

Mathis et Al. present parser-directed fuzzing pFuzzer [25] as it specifically targets syntactic processing of inputs via input parsers by a dynamic tainting of input characters. It is able to generate syntactically valid inputs for a large class of programs, avoiding large input errors. Padhye et Al. propose Zest [27], which converts random-input generators into deterministic parametric generators. est leverages program feedback in the form of code coverage and input validity to perform feedback-directed parameter search.

Liang et Al. expose DeepFuzzer [22], an enhanced grey box fuzzer with qualified seed generation, balanced seed selection, and hybrid seed mutation. They use a symbolic execution approach to generate qualified initial seeds which then guide the fuzzer through complex checks. They apply random and restricted mutation strategies which are combined to maintain a dynamic balance between global exploration and deep search.

5.3 Improving directed fuzzing

JavaScript is a language in constant evolution (same for any back-end JavaScript runtime environment like *Node.js*), it is necessary to integrate the recent constructions and functions proposed by the standard such as "Array.prototype. reduce()" and the generating expressions in JavaScript 1.8. This can be performed by integrating adapted seeds, but also by adapting the mutation algorithms to encourage the generation of these constructs. Moreover, the control structures ("for", "if" ...) are globally little used by fuzzers, so it could also be interesting to strengthen the mutation algorithms so that they use more of these structures.

New code sources additionally include the recent features related to the code base of each JavaScript engine. By following the most recent additions to the code base, we see that new features in terms of optimization are gradually being introduced, which probably heralds the discovery of new vulnerabilities in these new areas. For example, V8 is introducing modern compilers to its optimization chain: Turboprop and Sparkplug[7], which are described as "midtier" compilers, capable of generating less optimized code than the current JIT compiler, but much faster.

The process of adding new features in fuzzer is mostly done by generating a new grammar for the new version. When considering seeds that suit the tested program, we called the process directed fuzzing. Established techniques such as pattern recognition, inference, and feedback have been developed. Researchers tend to enhance directed fuzzing by including deep-learning methods to discover alternative directions to fuzz.

Böttinger et Al. [3] add a Q-Learning function (from deep learning theory) to reward some mutations. This process guides the fuzzer into inputs that better fit

[7] https://v8.dev/

the version or the program tested. Zong et Al. present FuzzGuard [41], a deep-learning-based approach to predict the reachability of inputs since nine out of ten inputs don't strike a bug in undirected fuzzing. FuzzGuard includes a 3-layer Convolutional Neural Network model to assume the impact of inputs before the test process, avoiding making inputs that will not return results.

5.4 Code coverage of optimized functions

Practically, tracking full and accurate path coverage is infeasible in practice due to the high instrumentation overhead. Thus, the fuzzers include algorithms to cover the code.

Wang et Al. propose SAFL [38], including a coverage-directed mutation. This fair and fast algorithm helps the fuzzing process to exercise rare and deep paths with more significant probability. Gan et Al. present CollAFL [10], a coverage-sensitive fuzzing solution. It mitigates path collisions by providing more accurate coverage information, while still preserving low instrumentation overhead. It also utilizes the coverage information to execute three innovative fuzzing strategies, promoting the speed of discovering alternative paths and vulnerabilities.

Code coverage is currently achieved by including the necessary code to the JavaScript engine source code before it is compiled. The compiled code generated during the initial compilation is correctly covered and instrumented, but this is not the case for the compiled code generated by the JIT compiler during the execution of the engine. It would be extremely interesting to be able to obtain information about the code coverage in these on-the-fly compiled functions, we could specifically fuzz them by varying the inputs to them according to the code coverage feedback.

5.5 Generic fuzzing

Most of the current fuzzers require the source code of the engine they are interested in, mainly to include to it the parts of code necessary to obtain information like code coverage. This requirement remains not really a problem since all mainstream browsers use open-source JavaScript engines. However, it might be interesting to consider closed-source JavaScript engines.

It would probably be necessary to find a way to cover the code that does not require the source code. AFL++ is a generic fuzzer project proposing to implement the emulation of an executable via QEMU to provide this kind of information [9]. Chen et Al. propose PolyGlot [4] a fuzzing framework that can generate semantically valid test cases to extensively test processors of different programming languages. They design a uniform intermediate representation to neutralize the difference in the syntax and semantics of programming languages in the Backus-Naur form.

6 Conclusion

Security vulnerabilities in software may lead to serious consequences, and vulnerability exploitation has become a hot area of research in networks and information security. Along with the expansion of complexity of software and JavaScript engines, fuzzing has incomparable advantages which other vulnerability exploiting technology can't provide such as static analysis.

The literature on fuzzing, and more precisely on JavaScript's fuzzers, is blooming since 2018 with various new techniques. The genetic evolution theory and the development of new algorithms for seeds selection and trees pathfinding and merging have greatly improved the ability and accuracy of fuzzers. We provide in this paper a review of recent trends in fuzzing on JavaScript engines like model inference, hybrid fuzzing, and new genetic algorithms. Furthermore, we provide several axes to improve the fuzzing algorithms based on some recent and promising works. We also provide unexplored and specific axes to the JavaScript engines.

References

1. Aschermann, C., Frassetto, T., Holz, T., Jauernig, P., Sadeghi, A.R., Teuchert, D.: Nautilus: Fishing for deep bugs with grammars. In: NDSS (2019)
2. Bishop, M.: Vulnerabilities analysis. In: Proceedings of the Recent Advances in intrusion Detection. pp. 125–136 (1999)
3. Böttinger, K., Godefroid, P., Singh, R.: Deep reinforcement fuzzing. In: 2018 IEEE Security and Privacy Workshops (SPW). pp. 116–122. IEEE (2018)
4. Chen, Y., Zhong, R., Hu, H., Zhang, H., Yang, Y., Wu, D., Lee, W.: One engine to fuzz'em all: Generic language processor testing with semantic validation. In: Proceedings of the 42nd IEEE Symposium on Security and Privacy (IEEE S&P 2021) (2021)
5. Cova, M., Kruegel, C., Vigna, G.: Detection and analysis of drive-by-download attacks and malicious javascript code. In: Proceedings of the 19th international conference on World wide web. pp. 281–290 (2010)
6. Dewey, K., Roesch, J., Hardekopf, B.: Language fuzzing using constraint logic programming. In: Proceedings of the 29th ACM/IEEE international conference on Automated software engineering. pp. 725–730 (2014)
7. Dominiak, M., Rauner, W.: Efficient approach to fuzzing interpreters. BlackHat Asia (2019)
8. Eberlein, M., Noller, Y., Vogel, T., Grunske, L.: Evolutionary grammar-based fuzzing. In: International Symposium on Search Based Software Engineering. pp. 105–120. Springer (2020)
9. Fioraldi, A., Maier, D., Eißfeldt, H., Heuse, M.: Afl++: Combining incremental steps of fuzzing research. In: 14th {USENIX} Workshop on Offensive Technologies ({WOOT} 20) (2020)
10. Gan, S., Zhang, C., Qin, X., Tu, X., Li, K., Pei, Z., Chen, Z.: Collafl: Path sensitive fuzzing. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 679–696. IEEE (2018)
11. Godefroid, P., Kiezun, A., Levin, M.Y.: Grammar-based whitebox fuzzing. In: Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 206–215 (2008)

12. Groß, S.: FuzzIL: Coverage guided fuzzing for JavaScript engines. Ph.D. thesis, Master's thesis, Karlsruhe Institute of Technology, 2018. https://saelo ... (2018)
13. Groß, S.S.: Attacking javascript engines : A case study of javascriptcore and cve-2016-4622 (2016), http://www.phrack.org/papers/
14. Groß, S.S.: Jit exploitation (2019), http://www.phrack.org/papers/
15. Hackett, B., Guo, S.y.: Fast and precise hybrid type inference for javascript. ACM SIGPLAN Notices **47**(6), 239–250 (2012)
16. Hamidy, G., et al.: Differential fuzzing the webassembly (2020)
17. Han, H., Oh, D., Cha, S.K.: Codealchemist: Semantics-aware code generation to find vulnerabilities in javascript engines. In: NDSS (2019)
18. Johari, R., Sharma, P.: A survey on web application vulnerabilities (sqlia, xss) exploitation and security engine for sql injection. In: 2012 International Conference on Communication Systems and Network Technologies. pp. 453–458. IEEE (2012)
19. Kang, Z.: A review on javascript engine vulnerability mining. In: Journal of Physics: Conference Series. vol. 1744, p. 042197. IOP Publishing (2021)
20. Kim, K., Jeong, D.R., Kim, C.H., Jang, Y., Shin, I., Lee, B.: Hfl: Hybrid fuzzing on the linux kernel. In: Proceedings of the 2020 Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA (2020)
21. Lee, S., Han, H., Cha, S.K., Son, S.: Montage: A neural network language model-guided javascript engine fuzzer. arXiv preprint arXiv:2001.04107 (2020)
22. Liang, J., Jiang, Y., Wang, M., Jiao, X., Chen, Y., Song, H., Choo, K.K.R.: Deepfuzzer: Accelerated deep greybox fuzzing. IEEE Transactions on Dependable and Secure Computing (2019)
23. Lin, H., Zhu, J., Peng, J., Zhu, D.: Deity: Finding deep rooted bugs in javascript engines. In: 2019 IEEE 19th International Conference on Communication Technology (ICCT). pp. 1585–1594. IEEE (2019)
24. Manès, V.J.M., Han, H., Han, C., Cha, S.K., Egele, M., Schwartz, E.J., Woo, M.: The art, science, and engineering of fuzzing: A survey. IEEE Transactions on Software Engineering (2019)
25. Mathis, B., Gopinath, R., Mera, M., Kampmann, A., Höschele, M., Zeller, A.: Parser-directed fuzzing. In: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 548–560 (2019)
26. Miller, B.P., Fredriksen, L., So, B.: An empirical study of the reliability of unix utilities. Communications of the ACM **33**(12), 32–44 (1990)
27. Padhye, R., Lemieux, C., Sen, K., Papadakis, M., Le Traon, Y.: Semantic fuzzing with zest. In: Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis. pp. 329–340 (2019)
28. Park, S., Xu, W., Yun, I., Jang, D., Kim, T.: Fuzzing javascript engines with aspect-preserving mutation. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 1629–1642. IEEE (2020)
29. Report, F.I.: Sea of nodes in v8 (2015), https://darksi.de/d.sea-of-nodes/
30. Schwarz, M., Lackner, F., Gruss, D.: Javascript template attacks: Automatically inferring host information for targeted exploits. In: NDSS (2019)
31. Sevcik, J.: Deoptimization in v8 (2016)
32. @sirdarkcat, t.: Eat sleep pwn repeat browser training (2019)
33. Sotirov, A.: Heap feng shui in javascript. Black Hat Europe **2007**, 11–20 (2007)
34. Thiyab, R.M., Ali, M., Basil, F., et al.: The impact of sql injection attacks on the security of databases. In: Proceedings of the 6th International Conference of Computing & Informatics. pp. 323–331 (2017)
35. Titzer, B.L.: Turbofan jit design

36. Wang, J., Chen, B., Wei, L., Liu, Y.: Skyfire: Data-driven seed generation for fuzzing. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 579–594. IEEE (2017)
37. Wang, J., Chen, B., Wei, L., Liu, Y.: Superion: Grammar-aware greybox fuzzing. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). pp. 724–735. IEEE (2019)
38. Wang, M., Liang, J., Chen, Y., Jiang, Y., Jiao, X., Liu, H., Zhao, X., Sun, J.: Safl: increasing and accelerating testing coverage with symbolic execution and guided fuzzing. In: Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings. pp. 61–64 (2018)
39. Yun, I., Lee, S., Xu, M., Jang, Y., Kim, T.: {QSYM}: A practical concolic execution engine tailored for hybrid fuzzing. In: 27th {USENIX} Security Symposium ({USENIX} Security 18). pp. 745–761 (2018)
40. Zhao, L., Duan, Y., Yin, H., Xuan, J.: Send hardest problems my way: Probabilistic path prioritization for hybrid fuzzing. In: NDSS (2019)
41. Zong, P., Lv, T., Wang, D., Deng, Z., Liang, R., Chen, K.: Fuzzguard: Filtering out unreachable inputs in directed grey-box fuzzing through deep learning. In: 29th {USENIX} Security Symposium ({USENIX} Security 20). pp. 2255–2269 (2020)

Multiplicative (non-zero) Homomorphic CRT Secret Sharing with Perfect Information Security and Beyond

Shlomi Dolev, Yaniv Kleinman*

Ben-Gurion University of the Negev, Be'er Sheva, Israel
{dolev@cs, yanivkl@post}.bgu.ac.il

August 21, 2022

Abstract

This work suggests a novel CRT-based secret sharing scheme with perfect information-theoretic security and multiplicative homomorphism. The scheme is designed to support non-zero secrets of multiplicative groups.

We will review some related works in CRT-based secret sharing schemes and examine the reasons for the innovativeness of our scheme. Our scheme will be detailed and analyzed in this work in order to set solid foundations for future work expanding this scheme to support additions.

1 Introduction

There is a rising need for clouding storage and clouding computing. Users of cloud services want to be sure that their sensitive data is not exposed to the service provider. Cloud computing lets companies focus on their primary objective, while cloud providers handle all the storage and computing infrastructure.

Users of cloud providers need to hide sensitive data. Data can be encrypted to hide it content. However, there are some flaws in this approach:

- Encryption is done using a key that needs to be stored.
- Encryptions are mainly based on the hardness of computing problems, meaning brute force on keys can always solve it rather than the ultimate

*Corresponding author

perfect information security. This fact is becoming terrifying when considering the emergence of quantum computing.

- Encryption can be very time-consuming, for example, calculating powers on numbers done in RSA.

Besides encryption, other methods like Secret Sharing (SS) and Secure Multiparty Computations (SMC or SMPC) exist. These methods are based on mathematical proofs that ensure the secret data is being secured as long as the adversary does not have a sufficient number of shares defined by a threshold to recover the secret.

The most known secret sharing schemes are Shamir's secret sharing scheme [6], and the schemes based on the Chinese Remainder Theorem (CRT) like Asmuth-Bloom's scheme [1]. Both schemes are designed to require a threshold of t out of n shares to reconstruct the secret, where $t \leq n$ shares are enough to reconstruct the secret fully. The terminology for using such a reconstruction threshold is called a *threshold scheme*.

The scheme must support mathematical operations for calculations performed on encrypted or shared data on the cloud company's servers. This support is mainly achieved using homomorphic operations. Homomorphism is a map between two algebraic structures of the same type that preserve the operation of the structures [3]. A homomorphic operation can be addressed by the equation: $f(x \oplus y) = f(x) \oplus f(y)$, where $'\oplus'$ is a binary mathematical operation in this case. Homomorphism is a property of the function f. Homomorphism can be limited to several operations; in this case, the scheme is only partly homomorphic.

2 Definitions

- n - The number of participants.
- t - The reconstruction threshold. State the minimum number of participants needed to reconstruct the secret.
- s - The secrecy bound. State the maximum number of participants who cannot learn about the secret.
- S - The secret data.
- S_i - The secret share of participant i.
- $[\cdot]_p$ - The arithmetic inside is performed in $\mathbb{Z}_p$
- U_M - The multiplicative group of integers modulo M.
- m_i - The i'th number to perform the modulus calculations.
- m_{i+j} - The $(i+j)(\text{mod } n)$ number to perform the modulus calculations with.
- $\mathcal{D}$ - The secret's distribution.
- $\mathcal{S}$ - The secret domain.
- $\mathcal{S}_i$ -The participant ith shared part's domain.
- $\mathcal{A}$ - The access structure. All qualified groups of participants who are capable of reconstructing the secret.

The relations between the scheme factors are:

1. $1 \leq s < t \leq n$
2. The gap from s to t does not have to be one.
3. It is possible that a group of participants smaller than t could reconstruct the data.
4. It is possible that a group of participants larger than s would not learn anything about the data.

Definition 1. *Perfect Secret Sharing Scheme* should satisfy the following two conditions:

- *Correctness*: Any qualified group of participants in $\mathcal{A}$ can reconstruct the secret.
- *Perfect Privacy*: No unqualified group of participants in $\bar{\mathcal{A}}$ can get any information about the secret.

Definition 2. *Perfect ramp secret sharing scheme* should satisfy the following two conditions:

- *Correctness*: Any qualified group of participants in $\mathcal{A}$ can reconstruct the secret.
- *Perfect Ramp Privacy*: For every group of participants $G, |G| \leq s$, given a secret S with distribution $\mathcal{D}$. The secret distribution stays the same, meaning the probability of the secret being equal to a specific element $S' \in \mathcal{S}$ stays the same even when knowing the shared secret data held by the participants of G. More formally: Given any $S' \in \mathcal{S}$:

  - $Pr[S = S'] = p$.
  - For every $S' \in \mathcal{S}$ and $S'_{i_j} \in \mathcal{S}_{i_j}, 1 \leq j \leq s$, with $1 \leq i_1 < i_2 < \cdots < i_s \leq n$, we have:

$$Pr[S = S' | S_{i_1} = S'_{i_1}, \ldots, S_{i_s} = S'_{i_s}] = p$$

3 Related Work

Some of the known SMPC homomorphic methods relevant to this research are:

Simple additive SSS – In this scheme, the sum of shared values reconstructs the secret. One such scheme has a threshold of $t = n - 1$, meaning it is an n out of n scheme. Let S be the secret, n be the number of participants, and $\mathbb{Z}_p$ be the field on which the calculations are being made [5].

- Distribution phase:
  The dealer choose at random the numbers $S_1, S_2, ..., S_{n-1} \in \mathbb{Z}_p$. The dealer then computes $S_n = S - \sum_{1 \leq i < n} S_i$. Finally, the dealer sends S_i to participant i for $1 \leq i \leq n$.
- Reconstruction phase:
  Let G be a group of participants gathered to reconstruct the secret. The participants compute the secret S by summing all their secret shares.
  $RF(\bigcup_{p_i \in G} S_i) : \begin{cases} if\ |G| = n \Rightarrow S = \sum_{1 \leq i \leq n} S_i (\text{mod } p) \\ if\ |G| < n \Rightarrow \perp \end{cases}$
- Additive homomorphism:
  Let S_1 and S_2 be secrets and S_{1_i} and S_{2_i} the secrets' shares of the i'th participant. $S_1 + S_2 = \sum_{1 \leq i \leq n} S_{1_i} + \sum_{1 \leq i \leq n} S_{2_i} = \sum_{1 \leq i \leq n} S_{1_i} + S_{2_i}$. Each participant can perform $S_{1_i} + S_{2_i}$ and send the sum for the reconstruction phase. Therefore, the scheme is an additive homomorphic scheme.

Simple multiplicative homomorphic SSS – This scheme is very similar to the additive one, but in this case, the product of shared values gives the secret. Moreover, there are some restrictions; Zero or numbers that are not co-prime to p are not allowed to be used. One such scheme has a threshold of $t = n - 1$, meaning it is an n out of n scheme. Let S be the secret, n be the number of participants, and U_p be the group on which the calculations are being made. [2]:

- Distribution phase:
  The dealer picks $n-1$ uniformly random nonzero elements $S_i, 1 \leq i < n$, from U_p. The dealer then calculates $S_n = S \cdot (\prod_{1 \leq i < n} S_i)^{-1}$. Finally, the dealer sends S_i to participant i for $1 \leq i \leq n$.

- Reconstruction phase:
  Let G be a group of participants gathered to reconstruct the secret. The participants compute the secret S by multiplying all their secret shares.
  $RF(\bigcup_{p_i \in G} S_i) : \begin{cases} if\ |G| = n \Rightarrow S = \prod_{1 \leq i \leq n} S_i (\text{mod } p) \\ if\ |G| < n \Rightarrow \perp \end{cases}$

- Multiplicative homomorphism:
  Let S_1 and S_2 be secrets and S_{1_i} and S_{2_i} the secrets' shares of the i'th participant. $S_1 \cdot S_2 = \prod_{1 \leq i \leq n} S_{1_i} \cdot \prod_{1 \leq i \leq n} S_{2_i} = \prod_{1 \leq i \leq n} S_{1_i} \cdot S_{2_i}$. Each participant can perform $S_{1_i} \cdot S_{2_i}$ and send the product to the reconstruction phase. Therefore, the scheme is a multiplicative homomorphic scheme. All the multiplications are correct and do not form a zero or a number with a common divider with p, because U_p is a multiplicative group.

Ramp additive homomorphic SSS – [4]. This scheme is based on the ideas of Asmuth-Bloom SSS [1] as mentioned earlier in the background. The scheme is an n out of n scheme with a security factor of s. This security factor means there is no information leak unless there are at least $s+1$ secret sharing parts. Using such a security factor s is called a *ramp scheme*. Let S be the secret, n the number of participants, and $\mathbb{Z}_{\text{prod}}$ the group on which the calculations are being made.

- Distribution phase:
  The dealer chooses a set of integers $(\text{prod}, m_1, m_2, ..., m_n)$ such that:
  1. $m_1 < m_2 < ... < m_n$ and $S < \text{prod} = M_n = \prod_{i=1}^{n} m_i$
  2. $gcd(m_i, m_j) = 1 (\forall i \neq j)$

  The dealer randomly chooses s integers $(r_1, ..., r_s)$ in $\mathbb{Z}_{\text{prod}}$, and computes $S_{\text{mix}} = [S + \sum_{i=1}^{n} r_i]_{\text{prod}}$.
  The dealer computes and distributes the shared set of each participant i:

$$S_i = (S_{\text{mix}} (\text{mod } m_i), r_1 (\text{mod } m_{i+1}), \ldots, r_s (\text{mod } m_{i+s}))$$

- Reconstruction phase:
  Let G be a group of participants gathered to reconstruct the secret. The participants compute the secret S by solving the CRT equations.

$$RF \begin{pmatrix} \bigcup\limits_{p_i \in G} S_{i,0} \\ \bigcup\limits_{p_i \in G} S_{i,1} \\ \ldots \\ \bigcup\limits_{p_i \in G} S_{i,s} \end{pmatrix} : \begin{cases} if\ |G| = n : \begin{pmatrix} S_{\text{mix}} = CRT[S_{1,0}, ..., S_{n,0}]_{\text{prod}} \\ r_1 = CRT[S_{1,1}, ..., S_{n,1}]_{\text{prod}} \\ \ldots \\ r_s = CRT[S_{1,s}, ..., S_{n,s}]_{\text{prod}} \end{pmatrix} \\ \Rightarrow S = [S_{\text{mix}} - \sum_{i=1}^{s} r_i]_{\text{prod}} \\ if\ s+1 \leq |G| < n \Rightarrow partial\ information \\ if\ |G| < s+1 \Rightarrow \perp \end{cases}$$

- Additive homomorphism:
  Let S_1 and S_2 be secrets. $S_{1_{\text{mix}}}, r_{1_1}, ..., r_{1_s}$ are the blinded secret and all the blinding randoms of S_1. $S_{2_{\text{mix}}}, r_{2_1}, ..., r_{2_s}$ are the blinded secret and all the blinding randoms of S_2. Each participant i has the secret shares of each blinded secret and blinding randoms:
  $S_{1_{mix_i}}, r_{1_{1_i}}, ..., r_{1_{s_i}}, S_{2_{mix_i}}, r_{2_{1_i}}, ..., r_{2_{s_i}}$.
  Now we will show that this scheme is additive homomorphic:
  $S_1 + S_2 = S_{1_{\text{mix}}} - \sum_{1 \leq j \leq s} r_{1_j} + S_{2_{\text{mix}}} - \sum_{1 \leq j \leq s} r_{2_j} = S_{1_{\text{mix}}} + S_{2_{\text{mix}}} - \sum_{1 \leq j \leq s} r_{1_j} + r_{2_j}$. Each participant can perform $S_{1_{mix_i}} + S_{2_{mix_i}}$ and $r_{1_{j_i}} + r_{2_{j_i}}$ for each $1 \leq j \leq s$ and send the sum of all the needed parts to the reconstruction phase, where using a CRT solver algorithm. Therefore, the scheme is an additive homomorphic scheme.

4 Motivation for the New Scheme

In this work, we want to introduce a novel SSS with the feature of multiplicative homomorphism. There are already schemes that allow homomorphic multiplication, one of them found in the related work Section(3). However, under some disclaimers, our scheme can be extended to support more features, such as threshold reconstruction and additive homomorphism.

5 Method Explanation

The scheme is an n out of n scheme with a security factor of s, meaning that without having at least $s+1$ secret sharing parts, there is no information leak. Using such a security factor s is called a *ramp scheme*. Let $S \in U_{\text{prod}}$ be the secret, n be the number of participants, and U_{prod} be the group in which the calculations are being made.

- Distribution phase:
  The dealer chooses a set of pairwise co-primes $m_1, m_2, ..., m_n$ and calculate $\text{prod} = \prod_{1 \le i \le n} m_i$ such that:

  1. $m_1 < m_2 < ... < m_n$ and $S < \text{prod} = M_n$.
  2. $\gcd(m_i, m_j) = 1, \forall i \neq j$.

  The dealer randomly chooses s integers $r_1, ..., r_s$ in U_{prod}, and computes $S_{\text{mix}} = [S \cdot \prod_{1 \le i \le s} r_i]_{\text{prod}}$. The dealer computes and distributes the shared set of each participant $1 \le i \le n$:

$$S_i = (S_{\text{mix}} (\text{mod } m_i), r_1 (\text{mod } m_{i+1}), \ldots, r_s (\text{mod } m_{i+s})).$$

- Reconstruction phase:
  Let G be a group of participants gathered to reconstruct the secret. The participants compute the secret S by solving the CRT equations, following the steps of the reconstruction function:

$$RF\begin{pmatrix} \bigcup\limits_{p_i \in G} S_{i,0} \\ \bigcup\limits_{p_i \in G} S_{i,1} \\ \cdots \\ \bigcup\limits_{p_i \in G} S_{i,s} \end{pmatrix} : \begin{cases} if\ |G| = n : \begin{pmatrix} S_{\text{mix}} = CRT[S_{1,0}, ..., S_{n,0}]_{\text{prod}} \\ r_1 = CRT[S_{1,1}, ..., S_{n,1}]_{\text{prod}} \\ \cdots \\ r_s = CRT[S_{1,s}, ..., S_{n,s}]_{\text{prod}} \end{pmatrix} \\ \Rightarrow S = [S_{\text{mix}} \cdot \prod_{i=1}^{s} r_i^{-1}]_{\text{prod}} \\ if\ s+1 \le |G| < n \Rightarrow partial\ information \\ if\ |G| < s+1 \Rightarrow \perp \end{cases}$$

5.1 Auxiliary Claims

Corollary 1. *Given the multiplicative group $A = U_{m_1 \cdot m_2 \cdot \cdots \cdot m_n} = U_{M_n}$ there is an isomorphism to the direct product of M_n pairwise co-primes dividers, meaning $B = U_{m_1} \times U_{m_2} \times ... \times U_{m_n}$.*

Corollary 2. *Define $M_n = m_1 \cdot m_2 \cdot \cdots \cdot m_n$ where $m_1, m_2, \ldots, m_n$ are pairwise co-primes. Given element $\alpha \in A = U_{M_n}$ of a finite group:*
$\forall \gamma \in U_{M_n}, \exists \beta \in U_{m_n}$ s.t. $\alpha \cdot \beta = \gamma$. Meaning that all elements can be the product of α and another element in the group.

Corollary 3. *Let G be a finite group. Given two elements $g_1, g_2 \in G$ chosen randomly, uniformly and independently. The multiplication $g_1 \cdot g_2 = g\prime \in G$ is a randomly, uniformly element of G.*

Corollary 4. *Let G be a finite group. Given two elements $g_1, g_2 \in G$ where g_1 is taken from a uniform distribution and g_2 is from some distribution $\mathcal{D}$. g_1 and g_2 are chosen independently. The multiplication $g_1 \cdot g_2 = g\prime \in G$ is a randomly uniform element of G.*

Proofs for all the corollaries can be found in Appendix (A)

5.2 Correctness

In order to use a scheme, one must know that the scheme is always correct.

Theorem 1. *The multiplicative scheme can always be reconstructed given a group of participants $G \in \mathcal{A}$.*

Proof. Given a group of participants $G \in \mathcal{A}$. The scheme is of threshold $t = n - 1$, meaning $|G| = n$. The correctness of this scheme relays on the fact that all elements in U_{M_n} have an inverse. Each $r_i, 1 \leq i \leq s$ can be reconstructed from the n shares of its modulus using CRT as $r_i \in U_{M_n}$ and CRT can reconstruct numbers to the product of modulus pairwise co-primes equations meaning one solution in $\mathbb{Z}_{M_n}$ because all the modulus are the pairwise co-primes factors of M_n. After reconstructing $r_i, \forall 1 \leq i \leq s$, we can calculate r_i^{-1}. That is the reason why we *must* take the randoms from U_{M_n} and not from $\mathbb{Z}_{M_n}$. To get the secret we calculate:

$$S_{\text{mix}} \cdot \prod_{i=1}^{s} r_i^{-1} = S \cdot \prod_{i=1}^{s} r_i \cdot \prod_{i=1}^{s} r_i^{-1} = S$$

That can be done because multiplication is associative in U_{M_n}. □

5.3 Multiplicative Homomorphism

Theorem 2. *The multiplicative scheme is multiplicatively homomorphic.*

Proof. To show that the scheme is multiplicative homomorphic, we will show that when taking k secrets, distributing them, multiplying them as shares on the participants' side and finally reconstructing the result, the result will be correct.

□

An expanded explanation of this proof can be found in Appendix (B)

5.4 Security Analysis

Since we want to use the scheme to store and make operations on secret data, we want to ensure that the scheme holds the security properties we need.

Theorem 3. *The multiplicative scheme is a Perfect ramp secret sharing scheme and is a perfect secret sharing scheme in case $s = n - 1$.*

For the simplicity of notation and sizes, we will use $m_i, \forall 1 \leq i \leq n$ as primes and not the general case of pairwise co-primes. To get some *intuition* we will start by analyzing the basic case of $s = 1$. The domain of S_{mix} and r_1 is U_{M_n} and the size of the domain is $|U_{M_n}| = \varphi(M_n) = \prod_{1 \leq i \leq n} \varphi(m_i)$.
For any elements $r'_1, S'_{\text{mix}}, S' \in U_{M_n}$, we have:
$Pr[r_1 = r'_1] = \frac{1}{\varphi(M_n)}$ as r_1 is randomly uniformly chosen.
$Pr[S = S'] = p$ as S is chosen from some distribution $\mathcal{D}$.
$Pr[S_{\text{mix}} = S'_{\text{mix}}] = \frac{1}{\varphi(M_n)}$ as r_1 is randomly uniformly chosen, and the secret S is of some distribution, based on Corollary 4.

The probabilities of r_1 and S_{mix} to be equal to r'_1 and S'_{mix}, respectively, do change knowing some i'th participant data since $s = 1$:

- $Pr[r_1 = r'_1 | r_1 (\text{mod } m_{i+1}) = r'_1 (\text{mod } m_{i+1})] = \frac{1}{\frac{\varphi(M_n)}{\varphi(m_{i+1})}} = \frac{\varphi(m_{i+1})}{\varphi(M_n)}$[1],

[1] same as a secret shared in Mignotte's scheme, which is the equivalence class of $r_1 (\text{mod } m_{i+1})$

- $Pr[S_{\text{mix}} = S'_{\text{mix}} | S_{\text{mix}} (\text{mod } m_i) = S'_{\text{mix}} (\text{mod } m_i)] = \frac{1}{\frac{\varphi(M_n)}{\varphi(m_i)}} = \frac{\varphi(m_i)}{\varphi(M_n)}$.

Yet, we claim that the conditional probability of the secret S to be equal to S' does not change:

$$Pr[S = S' | \begin{array}{l} r_1 (\text{mod } m_{i+1}) = r'_1 (\text{mod } m_{i+1}), \\ S_{\text{mix}} (\text{mod } m_i) = S'_{\text{mix}} (\text{mod } m_i)] \end{array} = p$$

We can write the conditional probability as follows:

$$Pr[S = S' | r_1 (\text{mod } m_{i+1}) = r'_1 (\text{mod } m_{i+1}), S_{\text{mix}} (\text{mod } m_i) = S'_{\text{mix}} (\text{mod } m_i)]$$

$$= \frac{Pr[S = S' \wedge r_1 (\text{mod } m_{i+1}) = r'_1 (\text{mod } m_{i+1}) \wedge S_{\text{mix}} (\text{mod } m_i) = S'_{\text{mix}} (\text{mod } m_i)]}{Pr[r_1 (\text{mod } m_{i+1}) = r'_1 (\text{mod } m_{i+1}) \wedge S_{\text{mix}} (\text{mod } m_i) = S'_{\text{mix}} (\text{mod } m_i)]}.$$

Now

$$\begin{array}{l} Pr[S_{\text{mix}} (\text{mod } m_i) = S'_{\text{mix}} (\text{mod } m_i)] = \\ Pr[r_1 \cdot {r'_1}^{-1} (\text{mod } m_i) = S' \cdot S^{-1} (\text{mod } m_i)] = \frac{1}{\varphi(m_i)} \end{array} \quad (1)$$

and

$$\begin{array}{l} Pr[r_1 (\text{mod } m_{i+1}) = r'_1 (\text{mod } m_{i+1})] = \\ Pr[r_1 \cdot {r'_1}^{-1} (\text{mod } m_{i+1}) = 1 (\text{mod } m_{i+1})] = \frac{1}{\varphi(m_{i+1})}. \end{array} \quad (2)$$

Since the events in (1) and (2) are clearly independent, we obtain:

$$Pr[\begin{array}{ll} r_1 (\text{mod } m_{i+1}) = r'_1 (\text{mod } m_{i+1}) & \wedge \\ S_{\text{mix}} (\text{mod } m_i) = S'_{\text{mix}} (\text{mod } m_i) &] = \frac{1}{\varphi(m_{i+1})} \cdot \frac{1}{\varphi(m_i)} \end{array}$$

Since S, S' and r_1, r'_1 are independent:

$$\begin{array}{l} Pr[S = S' \wedge r_1 (\text{mod } m_{i+1}) = r'_1 (\text{mod } m_{i+1}) \wedge \\ \quad S_{\text{mix}} (\text{mod } m_i) = S'_{\text{mix}} (\text{mod } m_i)] = \\ Pr[S = S' \wedge r_1 \cdot {r'_1}^{-1} (\text{mod } m_{i+1}) = 1 (\text{mod } m_{i+1}) \wedge \\ \quad r_1 \cdot {r'_1}^{-1} (\text{mod } m_i) = S' \cdot S^{-1} (\text{mod } m_i)] = p \cdot \frac{1}{\varphi(m_{i+1})} \cdot \frac{1}{\varphi(m_i)} \end{array}$$

and,

Finally:

$$\frac{Pr[S = S' \wedge r_1 (\text{mod } m_{i+1}) = r'_1 (\text{mod } m_{i+1}) \wedge S_{\text{mix}} (\text{mod } m_i) = S'_{\text{mix}} (\text{mod } m_i)]}{Pr[r_1 (\text{mod } m_{i+1}) = r'_1 (\text{mod } m_{i+1}) \wedge S_{\text{mix}} (\text{mod } m_i) = S'_{\text{mix}} (\text{mod } m_i)]} = p,$$

as required.

The security analysis proof for the general case can be found in Appendix(C)

6 Conclusion Remarks

Addition expansion of our multiplicative method. Returning to our main goal - performing fully homomorphic operations on secret data. This goal is equivalent to being able to calculate any polynomial function on secret data. All polynomials can be represented as the addition of products called monomials, with no parentheses.
In the proposed scheme, the dealer must know the function that we would like to calculate in advance. Given a function represented as the addition of monomials, the dealer chooses the randoms so that the product of all monomials' randoms is equal. For example, for a function represented as the addition of monomials:

$$f(x_1, x_2, x_3, x_4) = x_1 \cdot x_2 + x_2 \cdot x_3 + x_1 \cdot x_3 \cdot x_4$$

. An example of randoms for the given equation:

$$r_1 = 6, r_2 = 11, r_3 = r_1 = 6, r_4 = 11 \cdot [6^{-1}]_{M_n}$$

As this example shows, there are many dependencies between the randoms. In order to break some of these dependencies, we offer two methods:

1. Each occurrence of a variable can be hidden with a different random number. Using this method, the number of variables to be stored equals all occurrences of the variables in the equation.

2. We can add one variable to each monomial with the value 1. Then we can give each variable an actual random number to hide the variable value and, in the end, fix the product of the randoms with this independent random number multiplying the 1 variable.

Based on the methods described above, one can create a random selection for the function that needs to be calculated - depending on the trade-offs between flexibility and performance. Moreover, a deeper analysis needs to be done about the security of each method.

Addition under the limitation of calculations with a threshold. As defined in our proposed scheme, $m_1 < m_2 < \cdots < m_n$ are the co-primes that the modulus operations are performed on the participants' side. So, when the calculations are done under the limit of m_1, all results and intermediate results are elements of U_{M_n}. That means all the calculations are perfectly secured by the security analysis performed in Section 5.4.

Unbounded calculations. In this case, the security analysis needs to deal with the different scenarios. If for example a calculation or an intermediate calculation S_{res} is not from U_{M_n}, then, there is a modulo m_i such that $m_i | S_{\text{res}_{\text{mix}}}$, meaning that the participant with the m_i modulo used for S_{mix} can know the equivalence class of the real secret[2]: $S = S + k \cdot m_i, \exists k$.

Arithmetic circuit extension. We can calculate all the randoms needed for any given function in advance by parsing the equation. We can ensure that each addend's randoms are equal for each addition operation in the function.

[2] All the randoms are from U_{M_n} therefore they do not change the affect of zero divisors.

7 References

[1] C. Asmuth, J. Bloom, A modular approach to key safeguarding, IEEE Transactions on Information Theory (Volume: 29, Issue: 2, Mar 1983) 208 - 210.

[2] Dor Bitan, Shlomi Dolev, Optimal-round preprocessing-MPC of polynomials over non-zero inputs via distributed random matrix. Wireless Netw (2022).

[3] Stanley Burris, H.P. Sankappanavar, A Course in Universal Algebra (2012).

[4] Oğuzhan Ersoy, Thomas Brochmann Pedersen, Emin Anarim, Homomorphic extensions of CRT-based secret sharing, Discrete Applied Mathematics (Volume 285, 15 October 2020) 317-329.

[5] Yildirim, İsmail Fatih, SecurePL: A compiler and toolbox for practical and easy secure multiparty computation, Master Thesis Sabanci University (2008).

[6] Adi Shamir, How to share a secret, Communications of the ACM (Volume 22, Issue 11, 01 November 1979), 612–613.

[7] Leslie G. Valiant, Why is Boolean complexity theory difficult?, Proceedings of the London Mathematical Society Symposium on Boolean function complexity (1992), 84–94.

A Auxiliary Claims Proofs

Corollary 1. Given the multiplicative group $A = U_{m_1 \cdot m_2 \cdot \cdots \cdot m_n} = U_{M_n}$ there is an isomorphism to the direct product of M_n pairwise co-primes dividers, meaning $B = U_{m_1} \times U_{m_2} \times ... \times U_{m_n}$.

Proof. Let us define $f : A \to B$ in the following way:

$$f(\alpha) = \langle \alpha(\text{mod } m_1), \alpha(\text{mod } m_2), ..., \alpha(\text{mod } m_n) \rangle \tag{3}$$

We need to show that $f(\alpha_1 \cdot \alpha_2) = f(\alpha_1) \cdot f(\alpha_2)$.

$$f(\alpha_1 \cdot \alpha_2) = \langle (\alpha_1 \cdot \alpha_2)(\text{mod } m_1), (\alpha_1 \cdot \alpha_2)(\text{mod } m_2), ..., (\alpha_1 \cdot \alpha_2)(\text{mod } m_n) \rangle \tag{4}$$

$$\begin{aligned} f(\alpha_1) \cdot f(\alpha_2) = \langle \alpha_1(\text{mod } m_1), \alpha_1(\text{mod } m_2), ..., \alpha_1(\text{mod } m_n) \rangle \cdot \\ \langle \alpha_2(\text{mod } m_1), \alpha_2(\text{mod } m_2), ..., \alpha_2(\text{mod } m_n) \rangle = \\ \langle (\alpha_1 \cdot \alpha_2)(\text{mod } m_1), (\alpha_1 \cdot \alpha_2)(\text{mod } m_2), ..., (\alpha_1 \cdot \alpha_2)(\text{mod } m_n) \rangle \end{aligned} \tag{5}$$

The last equality in Equation(5) is achieved by the definition of multiplication in a direct product. □

Corollary 2. Define $M_n = m_1 \cdot m_2 \cdot \cdots \cdot m_n$ where $m_1, m_2, \ldots, m_n$ are pairwise co-primes. Given element $\alpha \in A = U_{M_n}$ of a finite group:
$\forall \gamma \in U_{M_n}, \exists \beta \in U_{m_n}$ s.t. $\alpha \cdot \beta = \gamma$. Meaning that all elements can be the product of α and another element in the group.

Proof. Given $\alpha \in U_{M_n}$, let's assume in contradiction that there is an element $\gamma_1 \in U_{M_n}$ such that: $\alpha \cdot \beta \neq \gamma_1, \forall \beta \in U_{m_n}$. We know that U_{M_n} is a multiplicative group, so every multiplication of elements in the group forms an element in the group. Since the source and the domain of the multiplication are of the same finite size, there is at least one element $\gamma_2 \in U_{M_n}$ such that:

$$\alpha \cdot \beta_1 = \alpha \cdot \beta_2 = \gamma_2, \beta_1 \neq \beta_2 \tag{6}$$

The element α has an inverse α^{-1} because U_{M_n} is a group. Applying α^{-1} on Equation (6) we get: $\alpha^{-1} \cdot \alpha \cdot \beta_1 = \alpha^{-1} \cdot \alpha \cdot \beta_2 = \alpha^{-1} \cdot \gamma \Rightarrow$
$\beta_1 = \beta_2 = \alpha^{-1} \cdot \gamma$ in contrast to $\beta_1 \neq \beta_2$ which means that our assumption was incorrect: $\nexists \gamma_1 \in U_{M_n}, \forall \beta \in U_{m_n}$ s.t. $\alpha \cdot \beta \neq \gamma_1 \Rightarrow \forall \gamma \in U_{M_n}, \exists \beta \in U_{m_n}$ s.t. $\alpha \cdot \beta = \gamma$. □

Corollary 3 Let G be a finite group. Given two elements, $g_1, g_2 \in G$ chosen randomly, uniformly and independently. The multiplication, $g_1 \cdot g_2 = g\prime \in G$ is a randomly, uniformly element of G.

Proof. We need to show that each $g\prime$ can be chosen with the same probability, which means $\frac{1}{|G|}$.

$$Pr[g_1 \cdot g_2 = g'] = Pr[\bigcup_{g_0 \in G} \{g_1 = g_0, g_2 = g_0^{-1} \cdot g'\}] =^1$$

$$\sum_{g_0 \in G} Pr[g_1 = g_0, g_2 = g_0^{-1} \cdot g'] =^2$$

$$\sum_{g_0 \in G} Pr[g_1 = g_0] Pr[g_2 = g_0^{-1} g\prime] =^3 \sum_{g_0 \in G} \frac{1}{|G|} \cdot \frac{1}{|G|} = |G| \cdot \frac{1}{|G|^2} = \frac{1}{|G|}$$

The 1 equality is achieved by the fact that each event of $g_0 \in G$ is different.
The 2 equality is achieved because of the independence of g_1 and g_2.
The 3 equality is achieved because g_1 and g_2 are chosen randomly uniformly. □

Corollary 4 Let G be a finite group. Given two elements $g_1, g_2 \in G$, where g_1 is taken from a uniform distribution and g_2 is from some distribution $\mathcal{D}$. g_1 and g_2 are chosen independently. The multiplication $g_1 \cdot g_2 = g\prime \in G$ is a randomly uniform element of G.

Proof. Same proof as Corollary(3), until equality 3.

$$Pr[g_1 \cdot g_2 = g'] = \cdots = \sum_{g_0 \in G} Pr[g_1 = g_0] Pr[g_2 = g_0^{-1} g\prime] =^3 \frac{1}{|G|}$$

The 3 equality is achieved as g_0 goes over all elements in G. The same holds with g_0^{-1} as each element has a different inverse. So finally, when going over all elements in G, we receive that each element g' has the same probability. □

B Multiplicative Homomorphism Expanded Proof

Let n be the number of participants, k the number of secrets and s the secrecy bound.

- Let $S_1, S_2, \ldots, S_k \in U_{M_n}$ be secrets that the user wants to know later about their product.
- Apply the distribution phase of the scheme on each secret. Note that $S_{i_j}, 1 \leq i \leq k, 1 \leq j \leq n$ is the part of the $S_i, 1 \leq i \leq k$ secret held by the j'th participant. Calculating:
  $S_{1_1} = (S_{1_{\text{mix}}} (\text{mod } m_1), r_{1_1} (\text{mod } m_2), \ldots, r_{1_s} (\text{mod } m_{s+1}))$
  $\ldots$
  $S_{k_1} = (S_{k_{\text{mix}}} (\text{mod } m_1), r_{k_1} (\text{mod } m_2), \ldots, r_{k_s} (\text{mod } m_{s+1}))$
  $S_{1_2} = (S_{1_{\text{mix}}} (\text{mod } m_2), r_{1_1} (\text{mod } m_3), \ldots, r_{1_s} (\text{mod } m_{s+2}))$
  $\ldots$
  $S_{k_2} = (S_{k_{\text{mix}}} (\text{mod } m_2), r_{k_1} (\text{mod } m_3), \ldots, r_{k_s} (\text{mod } m_{s+2}))$
  $\ldots$
  $S_{1_n} = (S_{1_{\text{mix}}} (\text{mod } m_n), r_{1_1} (\text{mod } m_1), \ldots, r_{1_s} (\text{mod } m_s))$
  $\ldots$
  $S_{k_n} = (S_{k_{\text{mix}}} (\text{mod } m_n), r_{k_1} (\text{mod } m_1), \ldots, r_{k_s} (\text{mod } m_s))$
- Multiply all shares of each participant - this operation can be done by the participants themselves as needed:
  $S_{\text{res}_1} = (\prod_{1 \leq i \leq k} S_{i_{\text{mix}}} (\text{mod } m_1), \ldots, \prod_{1 \leq i \leq k} r_{i_s} (\text{mod } m_{s+1}))$
  $S_{\text{res}_2} = (\prod_{1 \leq i \leq k} S_{i_{\text{mix}}} (\text{mod } m_2), \ldots, \prod_{1 \leq i \leq k} r_{i_s} (\text{mod } m_{s+2}))$
  $\ldots$
  $S_{\text{res}_n} = (\prod_{1 \leq i \leq k} S_{i_{\text{mix}}} (\text{mod } m_n), \ldots, \prod_{1 \leq i \leq k} r_{i_s} (\text{mod } m_s))$
- Reconstructing all the results of products shares in order to find the result of the product of all secrets $S_{\text{res}} = \prod_{1 \leq i \leq k} S_i$:

  $S_{\text{res}_{\text{mix}}} = CRT[S_{\text{res}_{1_0}} (\text{mod } m_1), \ldots, S_{\text{res}_{n_0}} (\text{mod } m_n)]_{M_n}$
  $r_{\text{res}_1} = CRT[r_{\text{res}_{1_1}} (\text{mod } m_2), \ldots, r_{\text{res}_{n_1}} (\text{mod } m_1)]_{M_n}$
  $\ldots$
  $r_{\text{res}_s} = CRT[r_{\text{res}_{1_s}} (\text{mod } m_{s+1}), \ldots, r_{\text{res}_{n_s}} (\text{mod } m_s)]_{M_n}$

Each CRT equation has all the pairwise co-prime modulus $m_i, 1 \leq i \leq n$ results, meaning that the numbers are reconstructed perfectly in modulo $\mathbb{Z}_{M_n}$. Also, the calculations are correct from Corollary 1.

C Security Analysis Proof

Theorem 3 The multiplicative scheme is a Perfect ramp secret sharing scheme and is a perfect secret sharing scheme in case $s = n - 1$.

Proof. Let G be a group of curious participants gathered to reconstruct the secret or leak some information about it, $|G| \leq s$.

Given $S_{\text{mix}}, r_1, \ldots, r_s \in U_{M_n}$, the domain of $S_{\text{mix}}, r_1, \ldots, r_s$ is U_{M_n} and the size of the domain is $|U_{M_n}| = \varphi(M_n) = \prod_{1 \leq i \leq n}(m_i - 1)$ (The Euler function in case M_n is the product of primes from degree 1).
For each element $r'_1, \ldots, r'_s, S'_{\text{mix}}, S' \in U_{M_n}$ the probabilities of $r_1, \ldots, r_s, S_{\text{mix}}$ and S to be equal accordingly are:
$Pr[r_1 = r'_1] = \frac{1}{\varphi(M_n)}$ as r_1 is randomly uniformly chosen.
...
$Pr[r_s = r'_s] = \frac{1}{\varphi(M_n)}$ as r_s is randomly uniformly chosen.
$Pr[S = S'] = p$ as S is chosen from some distribution $\mathcal{D}$.
$Pr[S_{\text{mix}} = S'_{\text{mix}}] = \frac{1}{\varphi(M_n)}$ as $r_1, \ldots, r_s$ are randomly uniformly chosen, and the secret S is of some distribution, based on Corollary 3 with induction That can be applied and Corollary 4.

The probabilities of $r_1, \ldots, r_s$ and S_{mix} to be equal to $r'_1, \ldots, r'_s$ and S'_{mix} respectively, do change knowing the information held by the group
$G = \{i_1, \ldots, i_s\}, 1 \leq i_1 < \cdots < i_s \leq n$:

$$Pr[r_1 = r'_1 | r_1(\text{mod } m_{i_1+1}) = r'_1(\text{mod } m_{i_1+1}), \ldots, r_1(\text{mod } m_{i_s+1}) = r'_1(\text{mod } m_{i_s+1})] =$$

$$\frac{1}{\frac{\varphi(M_n)}{\prod_{i \in G}\varphi(m_{i+1})}} = \frac{\prod_{i \in G}\varphi(m_{i+1})}{\varphi(M_n)}$$

...

$$Pr[r_s = r'_s | r_s(\text{mod } m_{i_1+s}) = r'_s(\text{mod } m_{i_1+s}), \ldots, r_s(\text{mod } m_{i_s+s}) = r'_s(\text{mod } m_{i_s+s})] =$$

$$\frac{1}{\frac{\varphi(M_n)}{\prod_{i \in G} \varphi(m_{i+s})}} = \frac{\prod_{i \in G} \varphi(m_{i+s})}{\varphi(M_n)}$$

$$Pr[S_{\text{mix}} = S'_{\text{mix}} | S_{\text{mix}}(\text{mod } m_{i_1}) = S'_{\text{mix}}(\text{mod } m_{i_1}), \ldots, S_{\text{mix}}(\text{mod } m_{i_s}) = S'_{\text{mix}}(\text{mod } m_{i_s})] =$$

$$\frac{1}{\frac{\varphi(M_n)}{\prod_{i \in G} \varphi(m_i)}} = \frac{\prod_{i \in G} \varphi(m_i)}{\varphi(M_n)}$$

However, we claim that the conditional probability of the secret S to be equal to S' does not change:

$$Pr[S = S' | r_1(\text{mod } m_{i_1+1}) = r'_1(\text{mod } m_{i_1+1}), \ldots, r_1(\text{mod } m_{i_s+1}) = r'_1(\text{mod } m_{i_s+1}),$$

$$\ldots, r_s(\text{mod } m_{i_1+s}) = r'_s(\text{mod } m_{i_1+s}), \ldots, r_s(\text{mod } m_{i_s+s}) = r'_s(\text{mod } m_{i_s+s}),$$

$$S_{\text{mix}}(\text{mod } m_{i_1}) = S'_{\text{mix}}(\text{mod } m_{i_1}), \ldots, S_{\text{mix}}(\text{mod } m_{i_s}) = S'_{\text{mix}}(\text{mod } m_{i_s})] = p$$

In fact we know that $s < n$ and therefore, we know that each modulo of $\varphi(m_{i_j}), \forall 1 \leq j \leq s$ does not appear n times in different equations. Hence the total amount of valid options for S to be calculated by the group G is:

$$\prod_{1 \leq j \leq s} \frac{\varphi(M_n)}{\prod_{i \in G} \varphi(m_{i+j})} \cdot \frac{\varphi(M_n)}{\prod_{i \in G} \varphi(m_i)} = \frac{\varphi(M_n)^{s+1}}{(\prod_{1 \leq j \leq s} \prod_{i \in G} \varphi(m_{i+j})) \cdot (\prod_{i \in G} \varphi(m_i))}$$

$$\leq^{3} \frac{\varphi(M_n)^{s+1}}{\varphi(M_n)^s}$$

As in the intuition part 5.4, based on Corollary 1, it is possible to show that the distribution of S does not change, even when knowing the information of the group G, and each element in $S' \in U_{M_n}$ stays with the same probability that S is equal to it. Thus, the scheme is perfect ramp security for any $s < n$.

The choice of $s = n - 1$ yields a perfect secret sharing scheme according to Definition 1 as $\forall G \notin \mathcal{A}$ there is no information leak. □

[3] Since $s < n$, than $\varphi(m_i), \forall 1 \leq i \leq n$ appear at most s times.

Entrepreneurship Pitch Track Chaired by: Yonah Alexandre Bronstein

Entrepreneurship Pitch Track chaired by: Yonah Alexandre Bronstein

The Hi-Tech industry and state-of-the-art research are becoming closer partners, as the implementation of research results becomes quicker and requires less manpower. The goal of the CSCML Pitch Track is to expose researchers to the world of entrepreneurs and vice versa, for the sake of creating mutual value and advancing the economy and society. Ten startups pitched this year in various areas including; Cyber security from the human level down, NFT picking advice for investors, fast & attractive phone app generation, cyber fraud insurance, augmented reality, and computer assisted remote Physical Therapy enabling! It was heartening, this year as well as in past years, to note that, even in a business focused track, there were entries that could justifiably be considered "for the greater good of the people" – that is, even if they had business motives and priorities, they would still end up benefiting all of us. These entrepreneurs deserve all the encouragement that we in the community can give them, in whatever form is suitable. As was the case last year, the Entrepreneurship Pitch Track at CSCML 2021 did an excellent job of fulfilling this objective and consequently was a great success. It received endorsement from leading VCs (Ford research center, Incubit, Disruptive AI, lool Ventures…) and corporations (IBM, Microsoft, Checkpoint …).

Out of the ten start-ups pitched during CSCML 2022, three were selected as finalists: Tamir Elazar & Ofer Hadar's "TheraPlay" was selected by the Entrepreneurship Pitch Track Committee and the audience as the leading entry and won the $500 prize. TheraPlay makes remote Physical Therapy more exciting for children, and safer for older adults concerned about Coronavirus. "Resight," presented by Omri Stein, augments reality while giving virtual objects a persistence that they lacked so far, received second place. And "Signex," presented by Itay Dekel, helps NFT buyers make better selections, and received third place.

Looking forward to an ever better CSCML in the years to come.

Regards,

Yonah Alexandre Bronstein

Entrepreneurship Pitch Track Chair

Green Blocks

Refining Behavior

The Team

Guy Shabtay
CTO & Co -Founder
Software Engineering Student.
Java, Pyton ,C++,C, Algorithms, Optimization.

Kobi Ninio
CFO & Co-Founder
B.A Major in Int. Affairs + M.B.A.
Investing & Managing.

Ziv Buksenboim
CEO & Co-Founder
B.Sc Electronic Engineer + M.B.A.
Investing & Managing.

The Vision

Behavioral Change

Phase 1

Phase 2

Phase 3

The Problem By Numbers

United States, 2020 :

35,766 fatal Motor Vehicle Crashes – 38,824 deaths occurred.

Estimated Annual Cost – 27 BILLION $.

Behavior Factor

12.2% - Uncontrollable

– Bad Weather, Technical Issues, Road Issues.

87.8% - Controllable (!)

– Alcohol Use, Speeding, Distractions etc.

Our Solution

Comply-To-Earn platform to motivate safe driving and road rules compliance.

Introducing – Green Blocks

Frontend - Mobile App –
monitoring speed, breaks, accelerations, turns and mileage.

Backend - Blockchain Rewarding System –
rewarding and ranking by the quality and safety of the driving and the level of compliance.

Green Coin GRC

GRC – Utility :

Comply To Road Rules

Car Insurance

Parking Services

Convenience stores

Donation

The Market

Drivers in the US – 2020:

228,200,000

2022 (proj.) – 238,183,000.

2023 (proj.) – 241,041,000.

Competition

		Institutes	Insurance companies	GREEN BLOCKS
1	Daily Insentive Experience	✗	✗	✓
2	Automated	✗	✓	✓
3	Privacy	✗	✗	✓

Business Model

- Commissions.
- Partnerships:
  - Insurance Companies.
  - Navigation Apps
  - Government, Associations.

ROADMAP


Team Building
Community Building
Lunch GreenBlocks App
Develop Own BlockChain
Expending To Europe
1
2
3
4
5
6
Q1
Q2
Q3
Q4
Q5
Q6
2026
Phase 2
Expending
Assembly NFT MarketPlace
Partnerships:
Car Companies, Government, Associations, Insurance companies
Building Insurance Model
Exploring Europe Market
Lunch GreenBlockChain
(rewards based on PoC)

Thank You !

Any Questions?

WANT BIG IMPACT?

Join GreenBlocks.

SAFEMIND

CYBER. WE TAKE IT PERSONALLY

Contact Info: Shai Kavas. Mobile:+972.54.7887529 www.SafeMind.ai Shai.Kavas@SafeMind.ai Headquarters: Israel

VISION & MISSION

Powered by AI, we address the weakest link in the cyber security chain - the **Human Factor.**

SAFEMIND Provides **pre-breach prediction & and prevention.** Which Individual/s (Employees, or groups) will be most susceptible to specific type of cyber attacks (Ransomware, Phishing, Data Loss etc) and applies adaptive controls accordingly.

SAFEMIND achieves higher security, and higher security awareness without interrupting the user.

THE PROBLEM

HUMAN CYBER RESILIENCE IN THE AGE OF TARGETED ATTACKS

WHY NOW?

1. Human error - biggest threat to businesses, especially in hybrid work model
2. Training & simulations - not sufficient & causing user fatigue
3. Human cyber attack surface expanding - mobile, PC, cloud. Next… metaverse
4. Same person acts differently depending on the state of mind

THE SOLUTION

PERSONAL. ADAPTIVE. CONTEXT BASED SECURITY

Human attack surface management platform: visual attack path, prioritized remediation (risk scores, probability), predictions, actionable data and contextual incident prevention playbooks

EXECUTIVE SUMMARY

INDUSTRY
CYBERSECURITY

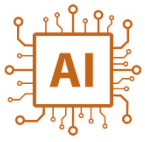
TARGET STAGE
SEED
Post-Pilot | Pre-Product | Pre-Revenue

ROUND SIZE / RUNWAY
$4M / 24 MONTHS

INTELLECTUAL PROPERTY / REGISTERED PATENT
7 YRS RESEARCH / YES

BUSINESS MODEL
B2B / SaaS

DIFFERENTIATION



BEHAVIORAL TAXONOMY

Classify attack category, context & state of mind

PRIVACY: PERSONA MASKING

Using adversarial AI against offensive AI algorithms

CIRCLE OF TRUST

Risk reputation based on the cyber behavior of social circles

DEFENSIBLE PROPRIETARY DATASET

Value from Day 1. Actionable mitigations

WHO WE ARE

SHAI KAVAS
Co-founder & CEO
- Entrepreneur
- 20+Yrs Cyber Executive
- Intel, McAfee, RAD

PROF. ASAF SHABTAI
Chief Research Officer
AI & CYBER SECURITY
- Partners: Verint, DT, NEC, RBC, RAFAEL, IBM
- 140+ publications, 35+ patents

DR. RON BITTON
Co-founder & Acting CTO
- Partners: NEC, Verint, DT, Astronautics
- 5 Patents, 15+ publications

PROF. RAMI PUZIS
Chief Innovation Officer
- Partners: Verint, DELL/EMC,DT, Lockheed Martin, Amdocs, IBM
- 100+ publications, 25+ patents

WHAT DID THEY SAY ABOUT US?

STEVE GROBMAN
McAfee SVP & CTO
"THE BIG VALUE - EXPLAINABLE AI"

MATTHEW ROSENQUIST
ECPLIZ, CISO
"SO MUCH DEPTH & INSIGHTS!"

GET IN **TOUCH**
Shai.Kavas@SafeMind.ai
+972.54.7887529

Tap

WORK WITH US
TO SHAPE THE FUTURE

Tap

BETTER APPS for Everyone

Business utilities

Entertainment

We transform great ideas into elegant software by harnessing state of the art technologies. Our design and cutting-edge tech empower our users with new capabilities that were only available to professionals.

Tap

Big User Base
Huge Impact!

We transform great ideas into elegant software by harnessing state-of-the-art technologies.

+350M	+30M	+195
DOWNLOADS	ACTIVE USERS	COUNTRIES

Leadership

Hristijan
Android lead

Ofir Krisspel
CEO

Quan Nguyen
iOS lead

Dmitry
AI

Ellie Bleiberg
Product

Tap

Work with us
To shape the future

AI tech

Edge Detection, Inpainting
Segmentation, OCR

iOS/Android Stack:

Kotlin/Swift
Compose/SwiftUI
Coroutines/combine
RoomDB/Realm

Tap

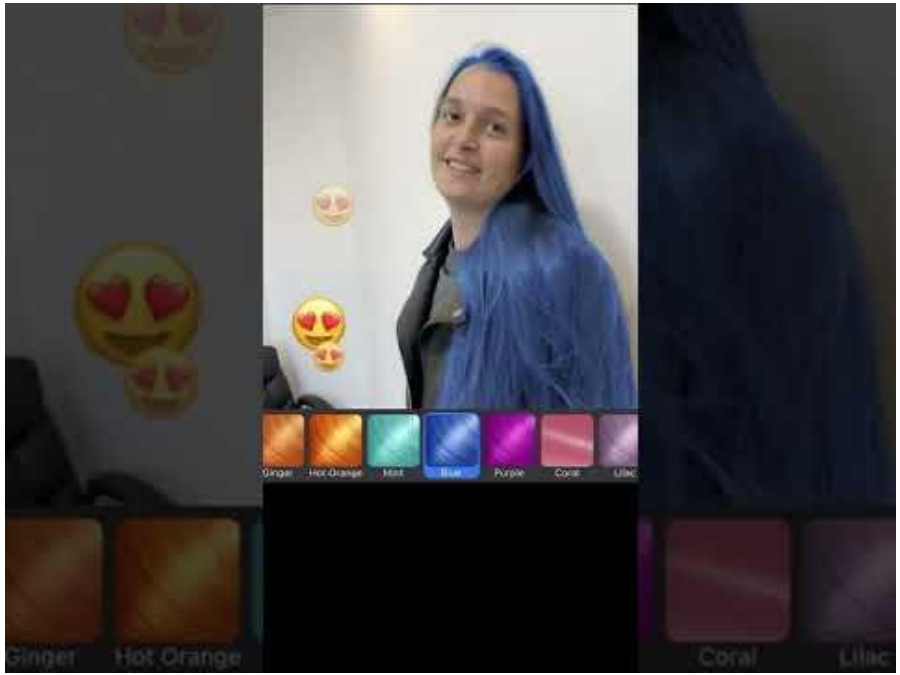
Our Tech

AI tech

Face detection, 3D Face Mesh, Image matting, Inpainting, GANs
Hair Segment, Instance segmentation

iOS/Android Stack:

Kotlin/Swift
Compose/SwiftUI
Coroutines/combine
RoomDB/Realm



Development Flow

Tap

Lets talk

Web

LinkedIn

Slack

Telegram

Brain

A smart comprehensive Project Management system

BUSINESS PRESENTATION

HTTPS://WWW.BRAIN-PM.COM/

IN SHORT

- Brain PM is a smart comprehensive Project Management system that uses Machine Learning and Artificial Intelligence to support decision making, with an easy-to-use user interface.

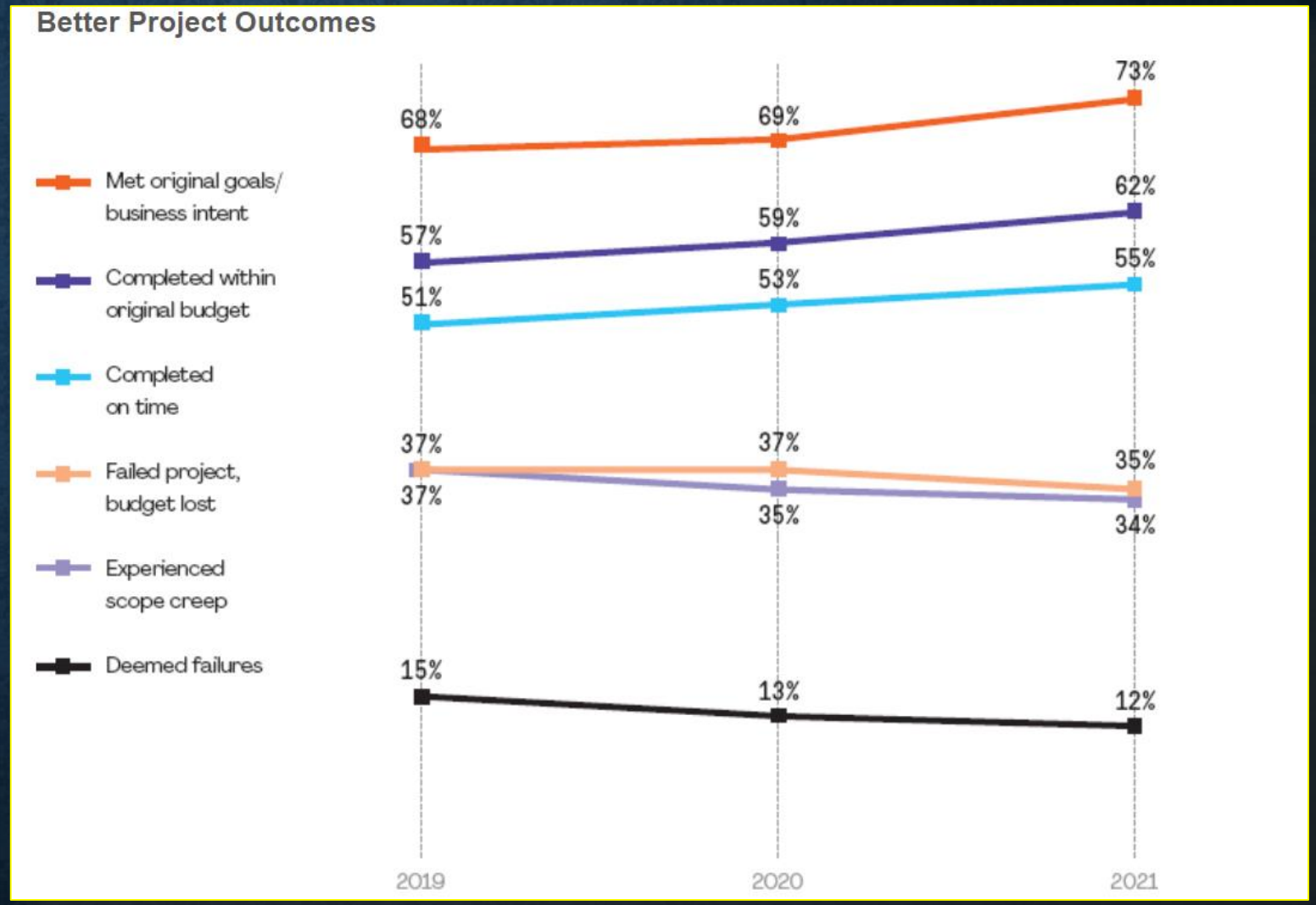
THE 2 MAIN PROBLEMS IN PROJECT MANAGEMENT

- Projects tend to fail fully or partially because of the lack of awareness to mistakes already done in similar projects: https://www.itproportal.com/features/are-we-about-to-make-the-same-software-failure-mistakes-all-over-again/
- Project management lacks simplicity and comprehensiveness: https://kepner-tregoe.com/blogs/what-is-missing-from-project-management/

RESULT OF THESE PROBLEMS

Approximately half of the projects are not completed on time, approximately third of them fail:

https://www.pmi.org/learning/library/beyond-agility-gymnastic-enterprises-12973

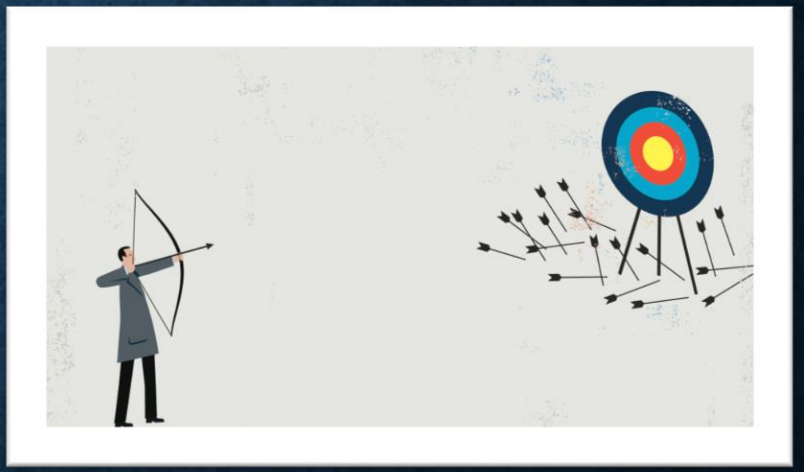


ROOTS OF PROJECT FAILURE

- The large variety of project management systems available today focus on a limited number of project management topics mostly on Time Cost and Resources management.
- As a result, project managers and their teams are using a variety of legacy systems for Scope Management, Risk Management, Quality Management, Procurement Management, and Communication Management.

CURRENT PROJECT MANAGEMENT SOLUTIONS

- Integrating the data and information from different systems is difficult and therefore project managers and their teams waste a lot of time on integration efforts and still cannot get the complete picture.
- The ability of currently used project management systems to use past data to forecast the future and to support planning monitoring and control is very limited and forecasting errors and well as poor planning are the main reason for project failure.



OUR SOLUTION - CHANGE THE WAY PROJECTS ARE MANAGED

- By integrating all the project management topics in one system and by using machine learning and Artificial Intelligence to improve forecasting and planning the high rate of project failures can be reduced.
- Simple User Interface that will be accessible to all levels of the project and will be comprehensive- no need for other systems.

OUR SOLUTION - CHANGE THE WAY PROJECTS ARE MANAGED

- Integration is achieved through a full coverage of all the project management topics included in ISO 21500 : Integration Management, Stakeholders Management, Scope Management, Resources Management, Time Management, Cost Management, Risk Management, Quality Management, Procurement Management, and Communication Management.
- By using one integrated system and learning from past projects the system provides comprehensive one stop support for project managers and their teams.

WHY USE BRAIN?

- Use a single inclusive system for a project
- Simple and easy-to-use user interface
- Smart project management using AI and ML
- Predict points of possible failure and avoid them
- Use prior knowledge from similar projects for better results

CURRENT STATUS

- Working Proof Of Concept: simulator that provides satisfactory results
- Three project teams currently working on comprehensive modeling and a working prototype for the different modules that will be ready by the end of summer 2023.

PLANNED SCHEDULE FOR START DATE 1/9/22

Phase 1 -HR and beginning logistics				
Recruitement	Recruitment and HR	0%	9/1/22	10/31/22
Office and equipment	Account manger, CEO	0%	9/1/22	10/1/22
Company formal founding	Financial advisor, CEO	0%	9/1/22	9/21/22
Hardware and software aquisitions	CTO, Account manger	0%	10/1/22	10/8/22
External services contracts	CTO, Account manger	0%	10/8/22	10/22/22
Phase 2 - Preliminary design, algorithms and technological planning				
Preliminary design - UML schemes	CTO, Chief engineer, IM engineer	0%	10/9/22	10/23/22
ML algorithm design	Chief engineer, algorithm designer	0%	10/11/22	10/25/22
AI algorithm design	Chief engineer, algorithm designer	0%	10/25/22	11/8/22
Data collecltion and analysis	Senior developer, IM engineer	0%	10/25/22	12/4/22
Technological planning	Chief engineer, Senior developer	0%	11/8/22	11/28/22
External services for the system	Chief engineer, Senior developer	0%	11/28/22	12/12/22
Phase 3- Building a prototype system				
Implementing algorithms	Algorithm designer, Junior developers		12/12/22	12/17/22
Integreating algorithms and data	Senior developer, Junior developers		12/24/22	1/13/23
Designing and coding modules	Senior developer, Junior developers		1/18/23	2/17/23
Integrating modules and algorithms	Senior developer, Junior developers		2/24/23	3/16/23
Building basic system GUI	Senior developer, Junior developers		1/18/23	2/17/23
Phase 4- Building a full working Beta version system				
Upscaling system for big data	Senior developer, Junior developers		2/17/23	4/18/23
Algorithms improvement and scaling	Algorithm designer, Junior developers		2/17/23	4/18/23
Integrating data in the system	Senior developer, IM engineer		4/18/23	6/17/23
Integrating system in cloud	Chief engineer, Senior developer		6/17/23	7/17/23
Scaling and improving GUI	Senior developer, Junior developers		6/17/23	7/17/23
Testing and improvement rounds	Senior developer, Junior developers		7/17/23	8/31/23
Beta release	All		8/31/23	9/1/23

BUDGET PLAN

Expense	Period (in months)	Price per unit (NIS)	Amount	Total estimated cost (NIS)
Salary- Junior full stack developer/QA and testing	18	13000	18* 5 developers	1170000
Salary- Senior full stack developer	18	23000	18* 2 developers	828000
Salary- Chief engineer, software architect, team head	18	30000	18	540000
Salary- Algorithm designer/implementer	18	30000	18	540000
Salary- Industrial management engineer (PM expertise)	18	25000	18	450000
Salary- Financial advisor/Account manager	18	15000	18	270000
Salary- CEO	18	40000	18	720000
Salary- CTO	18	35000	18	630000
External cloud and storage services (AWS/ Azure, etc.)	18	6000	18	108000
Office rent and expenses	18	10000	18	180000
Marketing expenses- counseling, promotion	18	8000	18	144000
Initial hardware expenses (Servers, computers, etc.)	1	5000	10	50000
Initial software expenses (Licenses, developer tools, etc.)	1	2000	10	20000
Recruitment and HR- External or salary	6	10000	6	60000
Margin of error- 10%				571000

OUR TEAM

- PROFESSOR AVRAHAM SHTUB
- Professor Shtub is a Professor Emeritus from Technion where he was the Stephen and Sharon Seiden Chair in Project Management and head of the Project Management research center.
  He is the founder of Sandboxmodel an offspring of Technion focusing on advanced project management tools.

- NADAV VOLOCH
- Nadav is a Computer Science Ph.D. candidate at the Ben-Gurion University of the Negev and a Lecturer in Ruppin academic center in Cyber Security and Software Development. He is also a freelance Cyber Security and software algorithms researcher and has been teaching in Academia for more than a decade different Computer Science courses.

THANKS FOR LISTENING

Brain - https://www.brain-pm.com/
Project Management
Feel free to contact us at:
info@brain-pm.com

Leveraging Noise for Ensuring Cybersecurity of Satellite Link

Rajnish Kumar[1], Shlomi Arnon[2]
[1]rajnish@post.bgu.ac.il, [2]shlomi@bgu.ac.il

The satellite communication links are essential to provide many public services that includes critical infrastructures e.g. finance, air traffic control, electric grids, transport systems military, banking and early warning weather systems. These links are nowadays more vulnerable to cyber-attacks due to their broadcast propagation over a certain geographical area. The ground station antennas that point towards the direction of satellites can be spoofed by a fake transmitter if it lies in the point of view of the antenna. The spoofing transmitter may utilize an aerial platform like drone to place itself in the line of sight of the highly directional antennas based at the ground station and send a spoofing signal that mimics the legitimate satellite signal to trick the ground station to lock onto it. Once locked, it will be able to launch any type of cyber-attacks on the system assuming higher encryption layers are hacked. Currently, the first step in locking onto the fake signal is decided based on the frequency and the signal-to-noise ratio (SNR) of the signal at the receiver. However, a fake transmitter can tune its transmitter power and frequency to mimic a legitimate satellite signal. To address this extremely challenging task, we propose a novel idea to establish a link between the transmitter and the receiver based on the noise signal especially at high frequency such as sub-THz and THz bands. The noise at the receiver antenna terminal is not under the control of any transmitter or receiver and is dependent on the brightness temperature of sky and the ground and the thermal noise due to electronic circuit connected to the antenna terminal. Thus, the noise signal will have the spatial signature of the traversed path through the atmospheric channel. The noise at the receiver due to a satellite signal that traverses all the layers of atmosphere and that from a drone platform which will be only nearby to the receiver would have quite different spatial signature due to different paths traversed. Thus, the legitimate satellite signal and the spoofing drone signal will have different noise spatial signature. This could be potentially utilized to differentiate between two signals and identify the spoofing transmitter in real time. At the ground station receiver, we sample the SNR at a high sampling rate of 1 millisecond for 0.5 seconds and then use a filter bank of wavelet transform to extract the noise from the SNR time series signal. The extracted noise signal is then fed to deep learning network based on long-short term memory (LSTM) for detecting whether the noise signal comes from a legitimate satellite transmitter or a spoofing drone. We demonstrate that a very high accuracy of more than 95% is achieved with the proposed scheme that will go a long way to ensure the physical layer security of the satellite link including the low earth orbit (LEO), medium earth orbit (MEO) and Geostationary orbit (GEO).

SECORAMA

The first end to end Kubernetes Governance platform

Leadership

Ido Shimon

Co-Founder

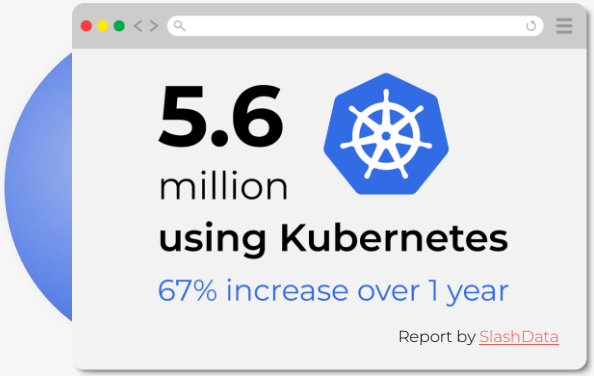
Over 10 years of software engineering.
Currently, Engineering manager and Risk Platform management system leader at proofpoint.
Frontend lead at Firelayers (Acquired by Proofpoint) one of the first R&D members.

proofpoint FIRELAYERS

Why Now?

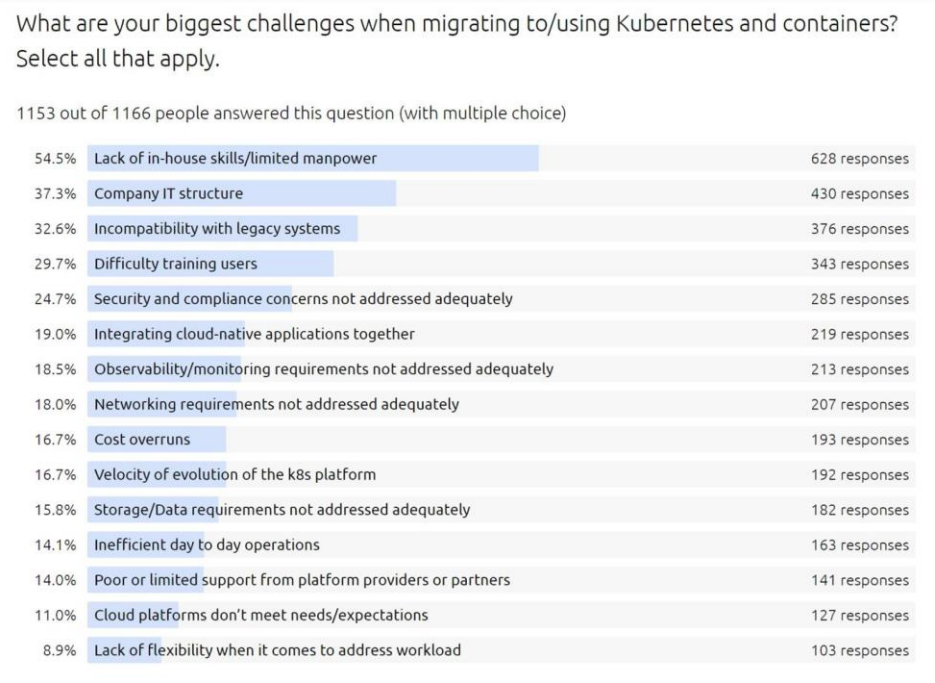

Top Container Deployment Challenges
Rank 1-3
Lack of Skilled Resources
30%
62%
Operational Complexity
10%
43%
Security Concerns
13%
35%
Immature Tooling
7%
25%
Container Enviro. Ongoing Upgrades and Maintenance
9%
22%
No DevOps Adoption
8%
22%
Obtaining Executive Buy-In
7%
21%
Developer Complexity
5%
20%
Necessity of Containerization Not Clear
7%
13%
No Digital Program
8%
Other Challenge
5%
Sum of Top 3
1st Choice
0%
35%
70%
n = 91, use or plan to use container technology
Q07. What are the top challenges your organization needs to address regarding its container deployment?
Source: 2020 Gartner Research Circle Container Adoption and Strategy Survey
Note: Values of 2% or less not indicated on graph
737153_C

link


5.6
million
using Kubernetes
67% increase over 1 year
Report by SlashData

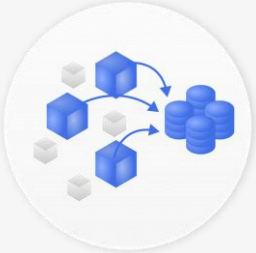

87%
87% of organizations around the globe will have Kubernetes deployed in mission-critical environments in the next 3 years, with one-third already relying on them today.
66%
Two-thirds of organizations around the globe that have Kubernetes deployed for mission-critical applications have no data protection in place for them.
48%
48% of organizations around the globe with Kubernetes deployed have already experienced a ransomware attack on their containerized environments.
veritas
VERITAS

Why Now?

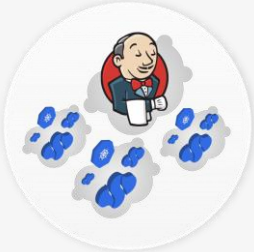

What are your biggest challenges when migrating to/using Kubernetes and containers? Select all that apply.
1153 out of 1166 people answered this question (with multiple choice)
54.5% Lack of in-house skills/limited manpower 628 responses
37.3% Company IT structure 430 responses
32.6% Incompatibility with legacy systems 376 responses
29.7% Difficulty training users 343 responses
24.7% Security and compliance concerns not addressed adequately 285 responses
19.0% Integrating cloud-native applications together 219 responses
18.5% Observability/monitoring requirements not addressed adequately 213 responses
18.0% Networking requirements not addressed adequately 207 responses
16.7% Cost overruns 193 responses
16.7% Velocity of evolution of the k8s platform 192 responses
15.8% Storage/Data requirements not addressed adequately 182 responses
14.1% Inefficient day to day operations 163 responses
14.0% Poor or limited support from platform providers or partners 141 responses
11.0% Cloud platforms don't meet needs/expectations 127 responses
8.9% Lack of flexibility when it comes to address workload 103 responses

Why Kubernetes is challenging?

Kubernetes serves as single place for many services and applications. They must run securely and all with all best practices.



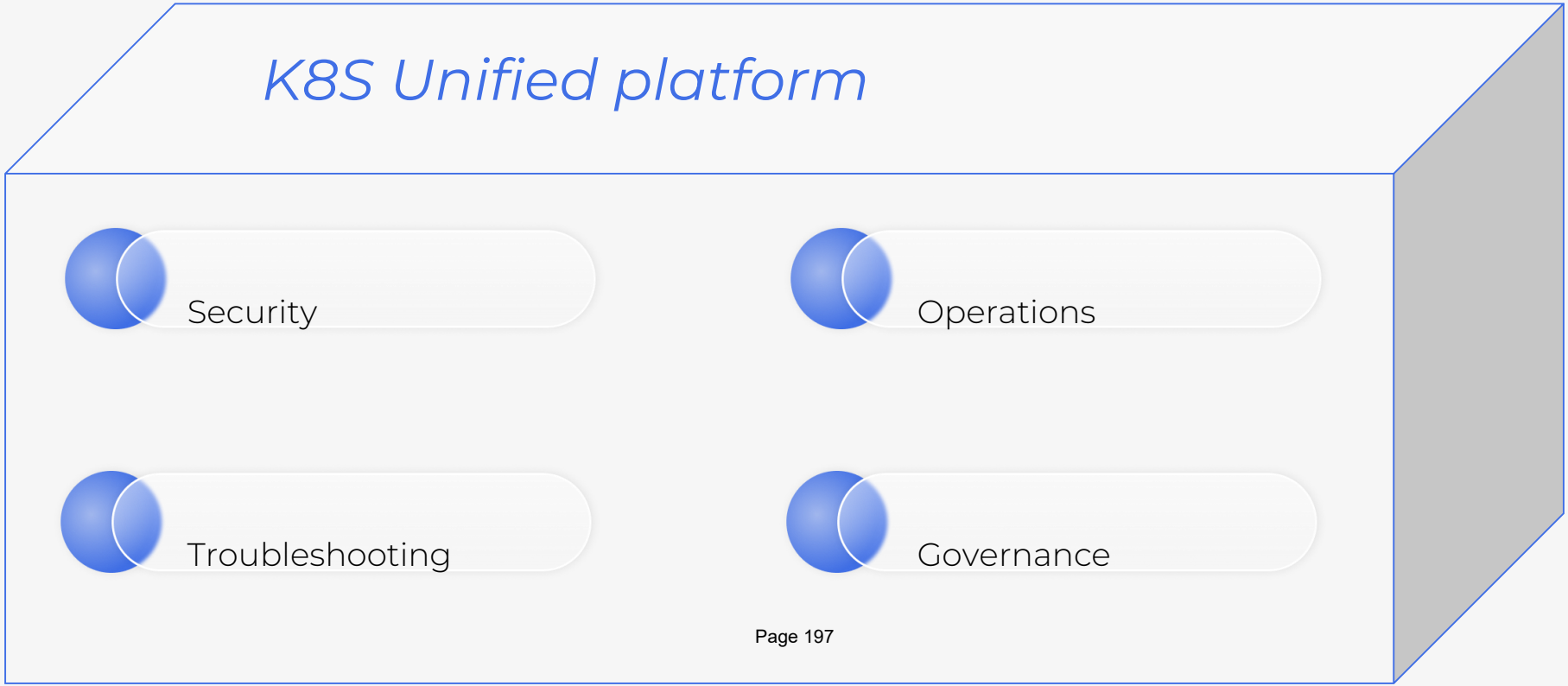
Troubleshooting failures in K8S is not an easy task, and only skilled engineers can make use of existing tooling.



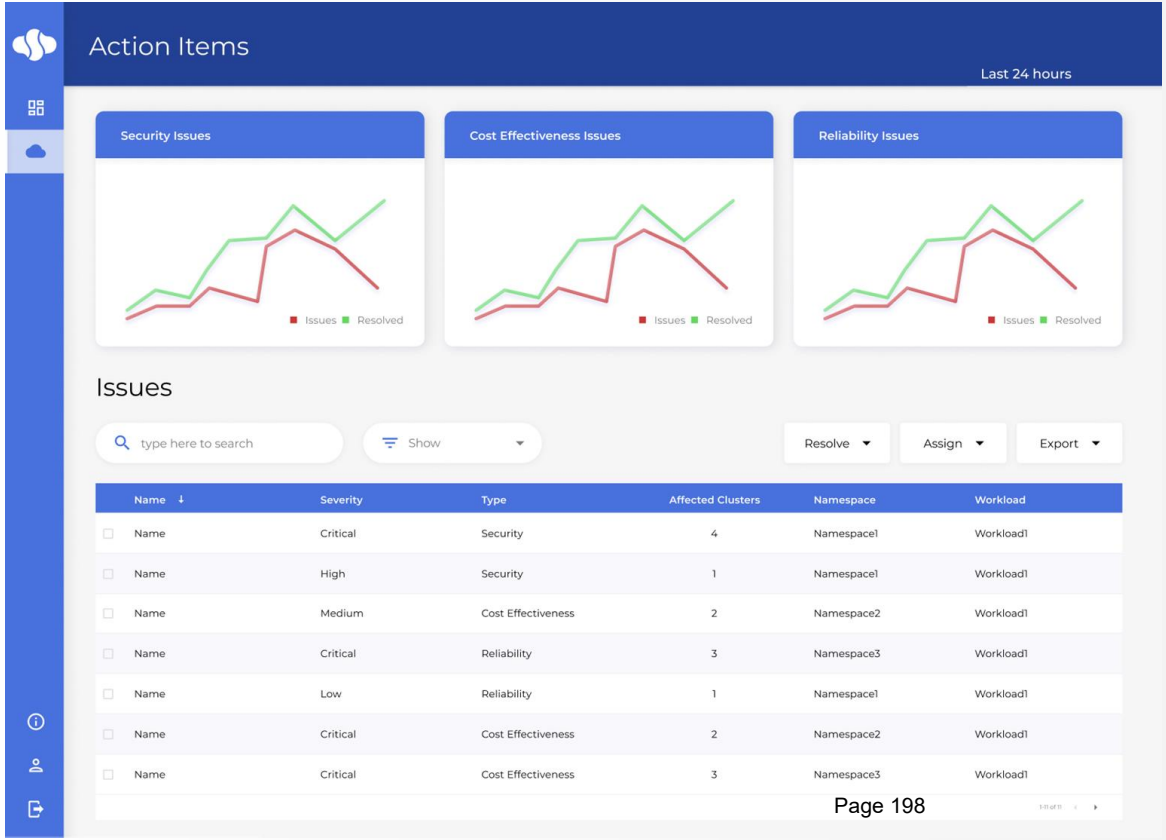
Management - K8S ecosystem has lots of open source tools to make the operations easier, but managing those tools at scale together is not trivial task.

Creating and managing in house tooling only increases the complexity of K8S operations and require more human resources to manage them at scale.

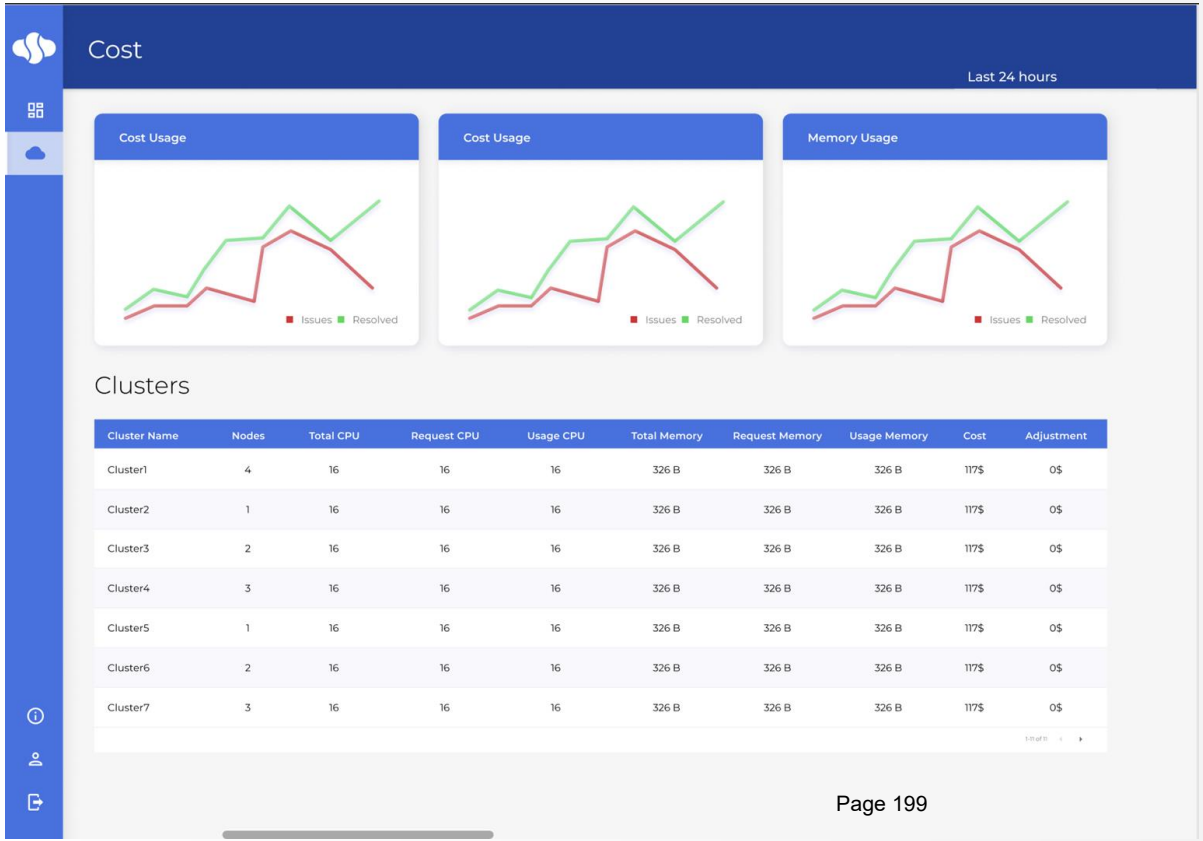
Secorama - Make Kubernetes easier



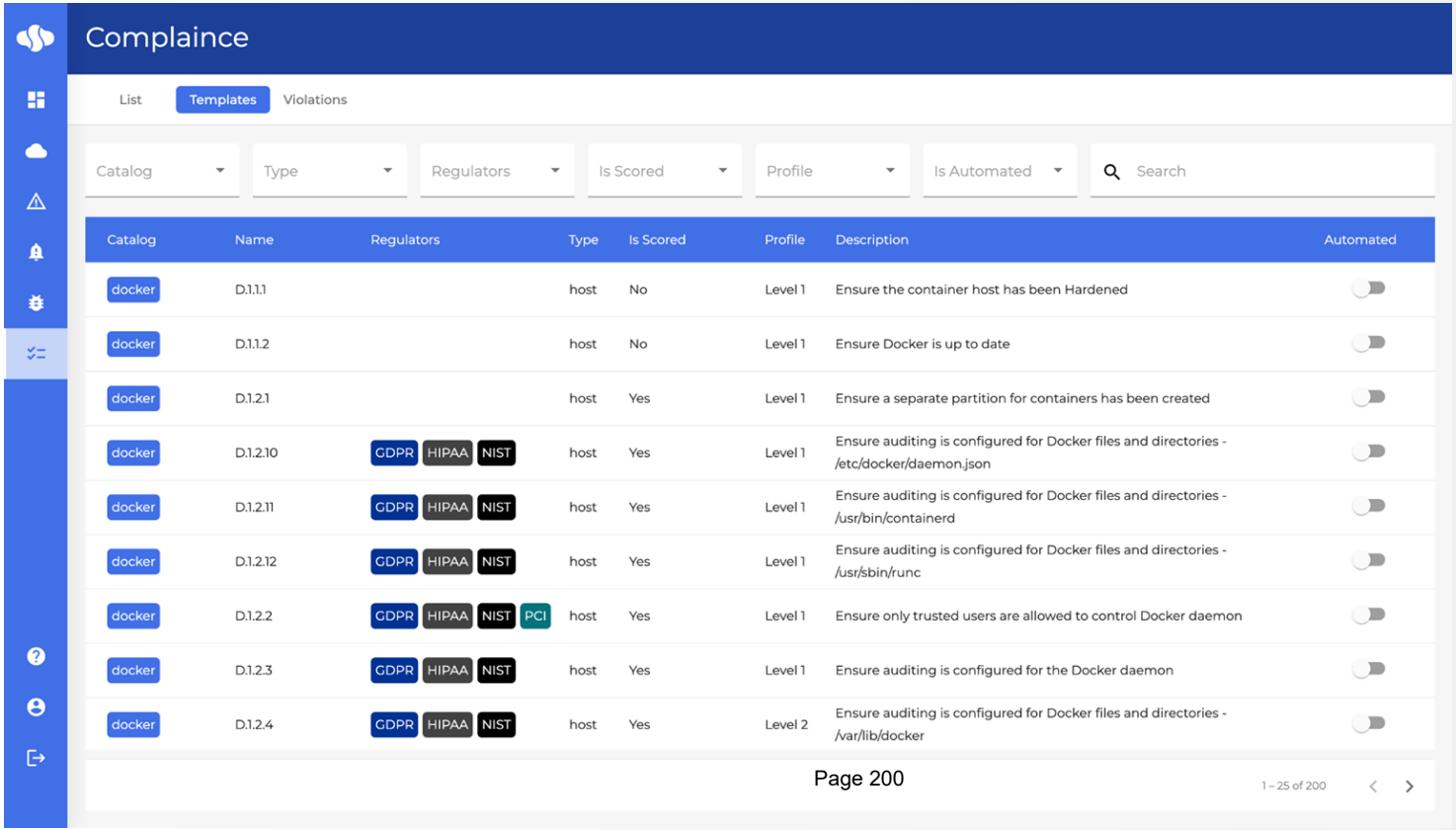
Action items



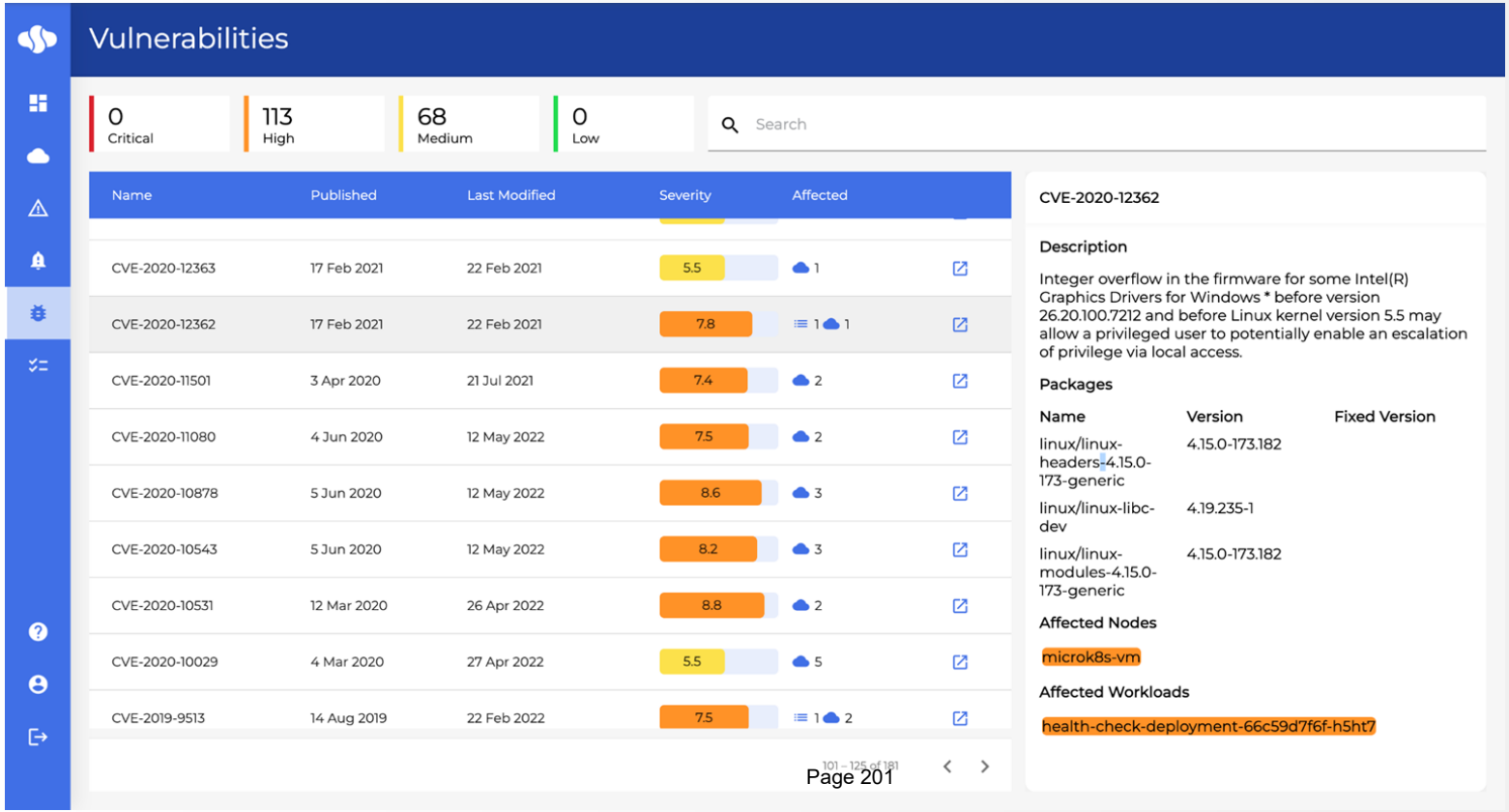
Cost Optimization



Compliance


Complaince
List
Templates
Violations
Catalog
Type
Regulators
Is Scored
Profile
Is Automated
Search
Catalog
Name
Regulators
Type
Is Scored
Profile
Description
Automated
docker D.1.1.1 host No Level 1 Ensure the container host has been Hardened
docker D.1.1.2 host No Level 1 Ensure Docker is up to date
docker D.1.2.1 host Yes Level 1 Ensure a separate partition for containers has been created
docker D.1.2.10 GDPR HIPAA NIST host Yes Level 1 Ensure auditing is configured for Docker files and directories - /etc/docker/daemon.json
docker D.1.2.11 GDPR HIPAA NIST host Yes Level 1 Ensure auditing is configured for Docker files and directories - /usr/bin/containerd
docker D.1.2.12 GDPR HIPAA NIST host Yes Level 1 Ensure auditing is configured for Docker files and directories - /usr/sbin/runc
docker D.1.2.2 GDPR HIPAA NIST PCI host Yes Level 1 Ensure only trusted users are allowed to control Docker daemon
docker D.1.2.3 GDPR HIPAA NIST host Yes Level 1 Ensure auditing is configured for the Docker daemon
docker D.1.2.4 GDPR HIPAA NIST host Yes Level 2 Ensure auditing is configured for Docker files and directories - /var/lib/docker
1 – 25 of 200

Vulnerabilities



Competitive Landscape

Features	Secorama	Fairwinds	Komodor	Robusta	Aqua
Live Troubleshooting	V	X	V	V	X
Policy Enforcement	V	V	X	X	V
Cost optimization	V	V	X	X	X
Upgrade advisor	V	?	X	X	X
Runbooks automation	V	X	?	V	X
Open source integrations	V	V	X	?	X
ChatOps	V	X	X	V	X
Security and Compliance	V	V	X	X	V
RBAC insights	V	V	X	X	V

The tools from OSS comunity

- Security
  - trivy (vuln scanner)
  - grype (vuln scanner)
  - kube-hunter (compliance)
  - open policy agent (policy enforcement)
  - falco (runtime)
  - kubiscan (rbac)?
  - krane (rbac)?
  - audit2rbac
- Chatops
  - botkube
  - robusta
  - own tool?
- Capacity planning
  - goldilocks (based on vpa)
  - vpa
  - kubecost?
  - own tool?
- Troubleshooting
  - node problem detector
  - robusta
  - replicatedhq/troubleshoot
  - own tool?
  - kubewatch
  - salesforce/sloop
- Governance
  - polaris
  - pluto
  - own tool?
  - spekt8
  - clusterlint
  - pleco
  - kubernetes-event-exporter
- Scaling
  - keda
  - own tool?

Addressing All Stakeholders Needs

Security

We provide all the standard security governance features.

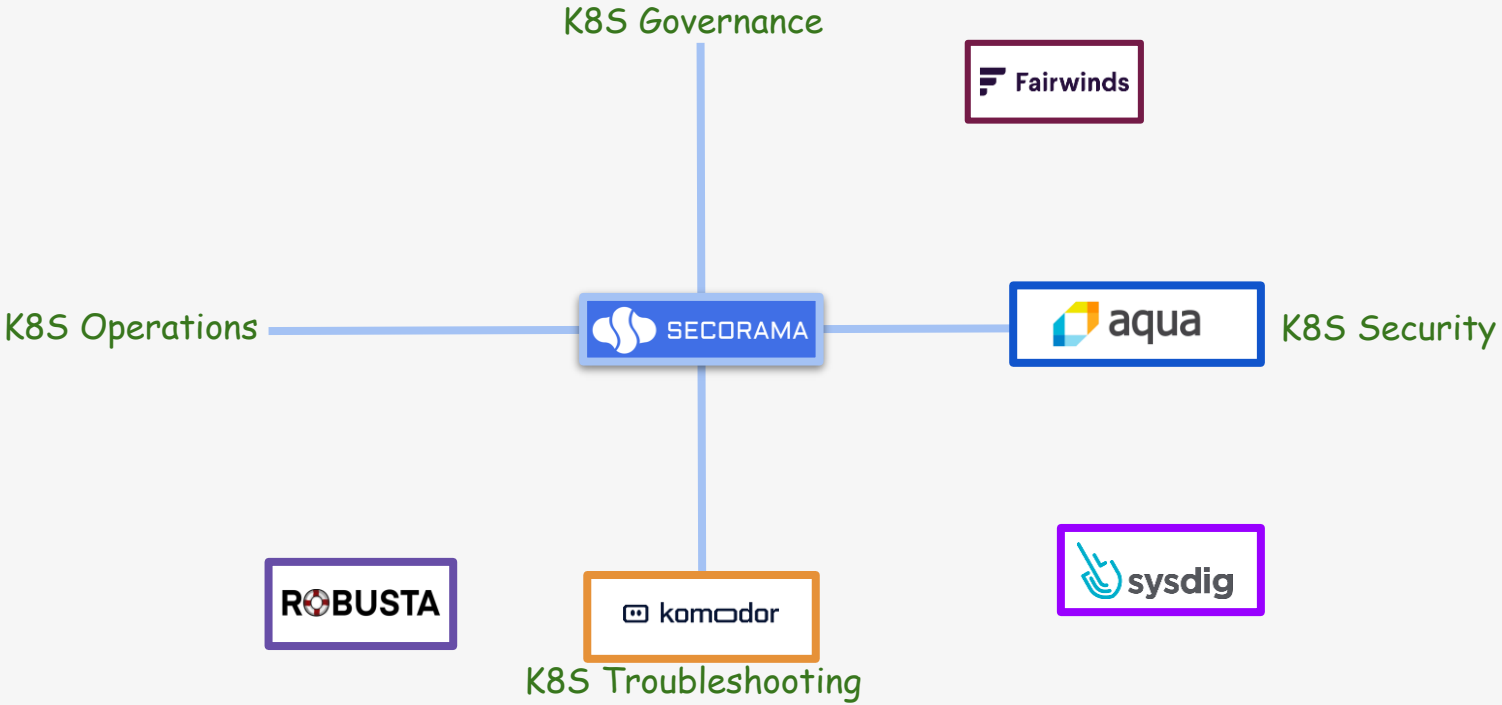
Management

We make capacity planning much more easier and optimize it. Get deprecation/upgrade advices

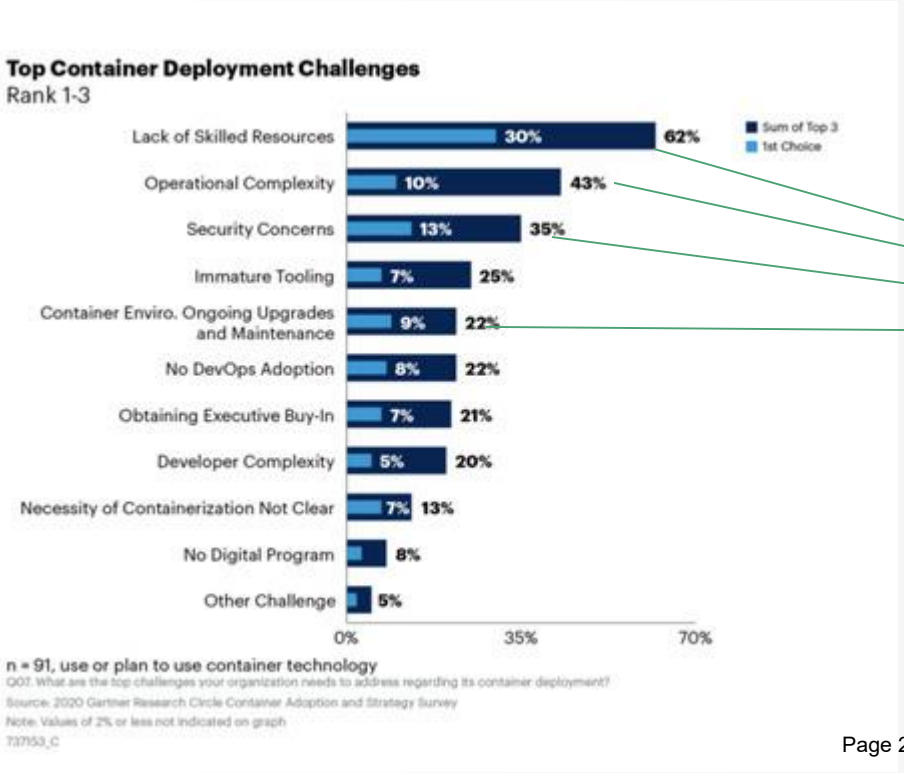
Troubleshooting

We help you detect easily the root cause and provide automatic fixes

Market Position



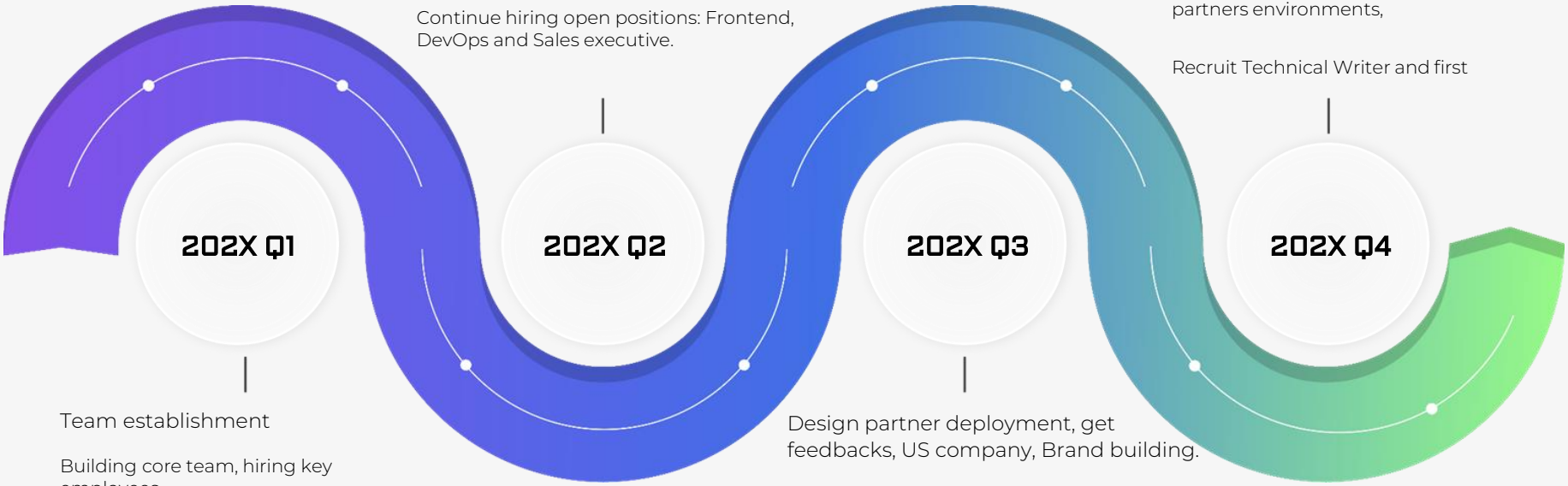
Why Secorama?



SECORAMA

Various Capabilities into ONE unified Management Console

Roadmap



Team establishment

Building core team, hiring key employees.

eBpf, backend developers, BizDev, Security researcher, Product Manager.

POC + Design partners + Continue hiring as needed

Deliver a POC, sign a design partner.

Continue hiring open positions: Frontend, DevOps and Sales executive.

Design partner deployment, get feedbacks, US company, Brand building.

Deploy the POC version in the design partner's environment, get feedbacks and adopt the product as needed

Establish US company. Hiring QA automation, Marketing executive.

Alpha version + Additional design partners + recruit technical writer

Deliver Alpha version, add additional design partner, install the version in the design partners environments,

Recruit Technical Writer and first

Raising $6M

- Proven team of experts
- Novel technology
- Market traction
- Design partner budget

Thank You!

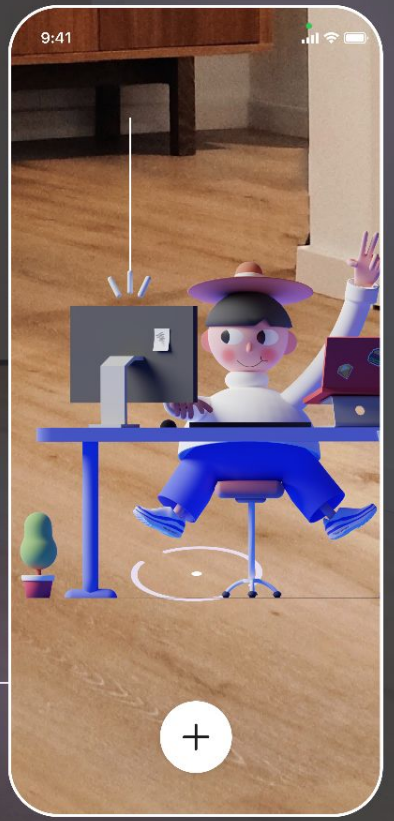
BUILD THE NEXT-GEN AR APPLICATIONS.

RESIGHT

resight.io | omri@resight.io


רח' אלנבי
ش. ألنبي
REHOV ALLENBY
Arcaffe
מרחוב נחלת בנימין
ممشى نحلات بنيامين
Nachlat Binyamin Market
כיכר מגן דוד
ميدان نجمة داود
Magen David Square
שוק הכרמל
سوق الكرمل
Carmel Market
פלורנטין
فلورنتين
Florentine
רחוב שינקין
شارع شينكين
Sheinkin St.
כיכר התרבות
ميدان هثربوت
Hafarbut Square

Resight seamlessly closed the gap of bringing 3D objects from editing tools (Unity, Unreal Engine) to the **Metaverse**

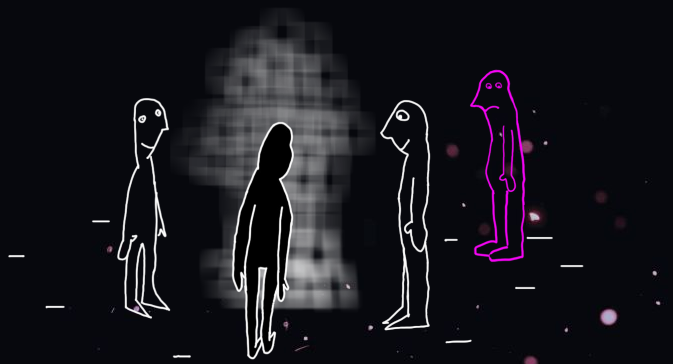


The Problem

No persistent

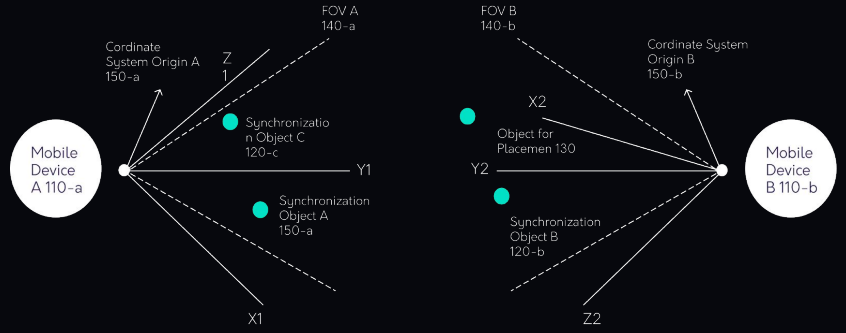
Latency

No discovery



Unique Technology

#on_device | #distributed | #RT | #3D | #privacy_first | #visual_index

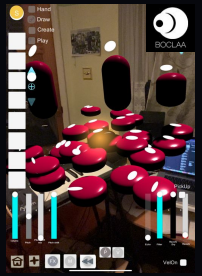


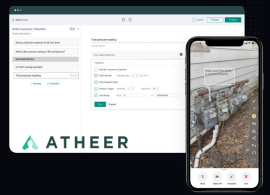
Scalable, No Hassle
AR Memory Layer

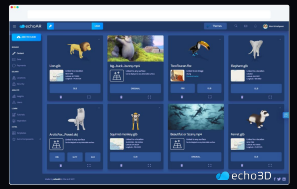
Low Latency, Real Time
Multiuser AR

mirror.

Early Adopters Use Cases







Signex.io

Helping investors pick the right NFT project with social intelligence

www.signex.io

SIGNEX.IO

itay@signex.io
+972-525397911

A Market Going **Mainstream**

Payments giant Mastercard partners with Coinbase to make **NFTs more accessible to everyone**

Samsung is introducing the **world's first TV screen-based NFT explorer and marketplace** aggregator, a platform that lets you browse, purchase, and display NFTs all in one place

Meta is developing ways to **create, display, and sell NFTs** on Facebook and Instagram

Nike **buys virtual shoe company** that makes NFTs. Recently sold 600 pairs of **NFT shoes – making over $3M**

Twitter is **Integrating NFTs**

*Jefferies investment bank Sees the NFT Market Reaching More Than $80B in Value by 2025

*A market worth now over $40B

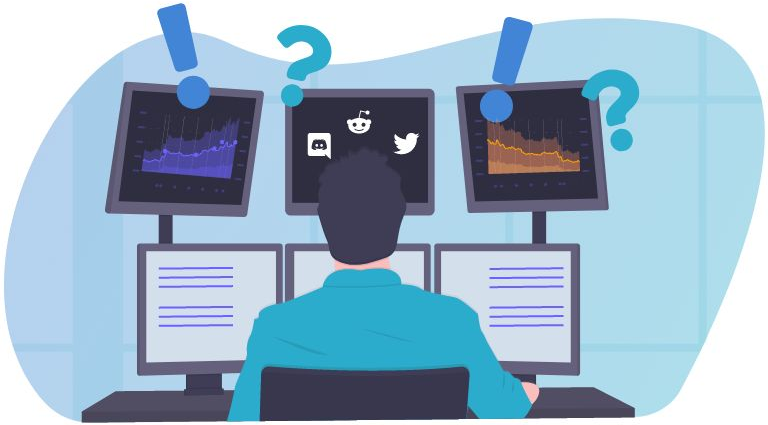
The **Pain** all **NFT** investors have

Follower gains and the **strength of the community** are the **Biggest factor** for deciding success or failure for an NFT project.

Yet there are nearly no products that follow this.

Millions of people in the **$41B market** ask the same question - *“What are the social trends for this NFT”?*

NFT Investor Problem



The Opportunity

Millions are looking for social trends and search for NFT community information to be able to:

See potential in fastly growing new projects
Discover investment opportunities
Evaluate projects quick based on social proof
Keep track of their favorite/owned NFT projects

Current Solution

*Check each project's **Twitter and Discord** accounts daily, check the **amount of followers**, count the amount of **messages and activity**, keep notes and compare to past numbers*

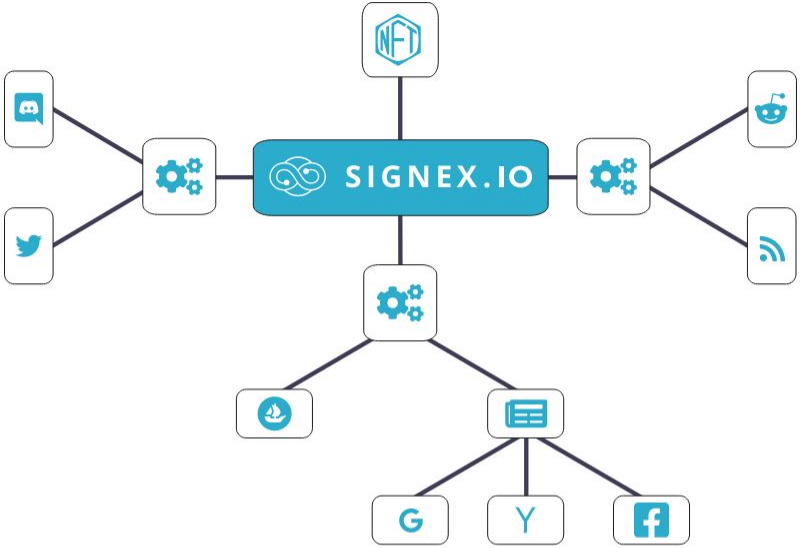
Our solution

Users get a **quick overview** of social trends and community behavior for each project

We follow the **size, growth, activity, sentiment** and more for each NFT community

Unveiling the **Biggest Component** behind the success of every NFT project

What Is Signex



NFT Data-Platform that goes through **social and news outlets**, checks the trends & social changes for each NFT project, giving a daily score for every project based on social metrics.

Total Addressable Market

2021

15M Beginner NFT Investors

28.6M Wallets

Subscription value
$600/Year

Customer Lifetime Value
$600.00

Total Addressable Market
$600.00 * 15M
$9B

Approximately 15M new NFT investors in 2021

2020 - 545K wallets holding NFTs
2021 - 28.6M wallets holding NFTs **(+5000% increase)**

Fully Functioning MVP

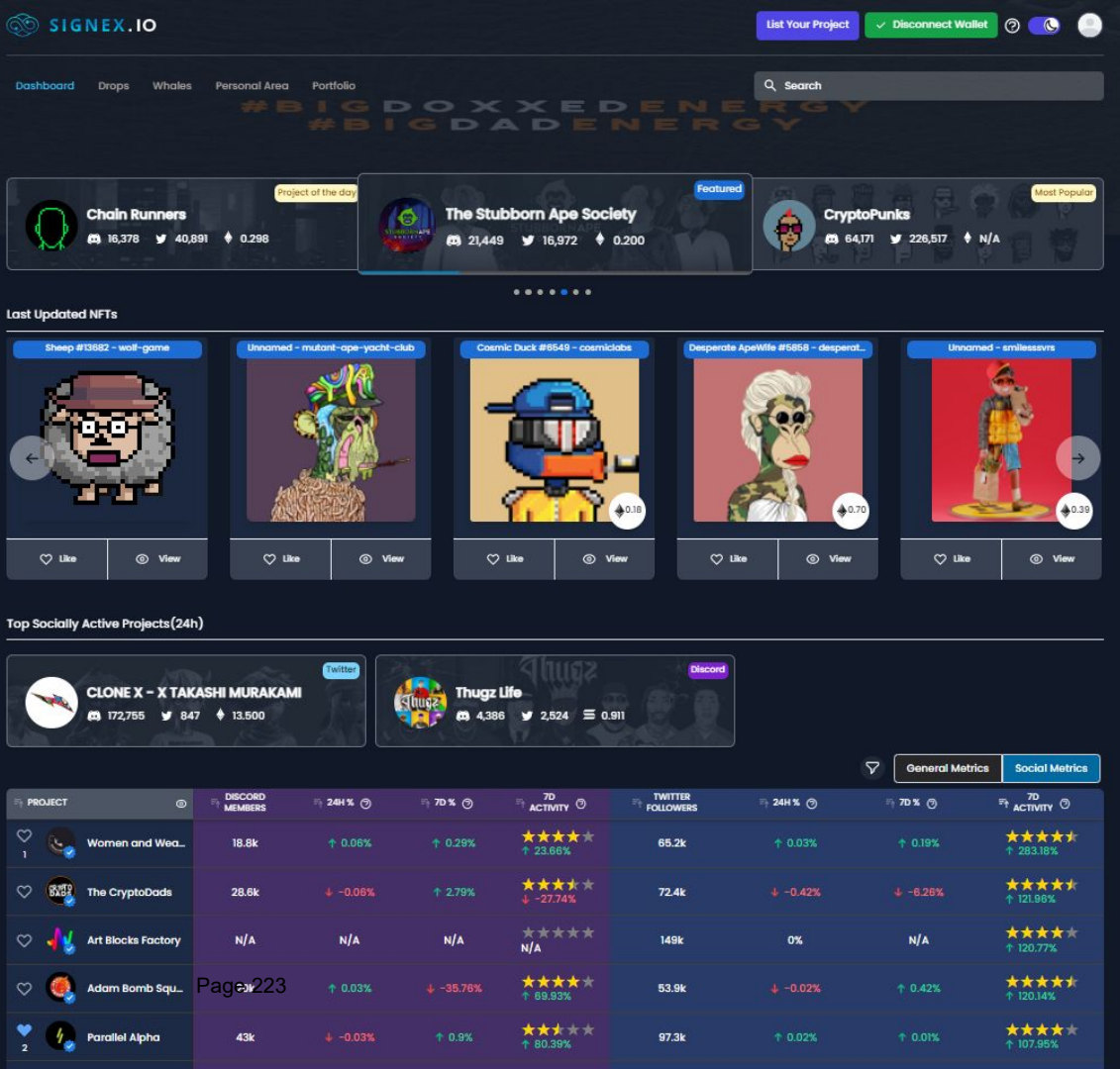
Signex.io

Our platform

Main Dashboard

Key Features -

- Community social stats
- Project daily score
- Supply, owners, trade volume, floor price, avg price
- Top performing projects

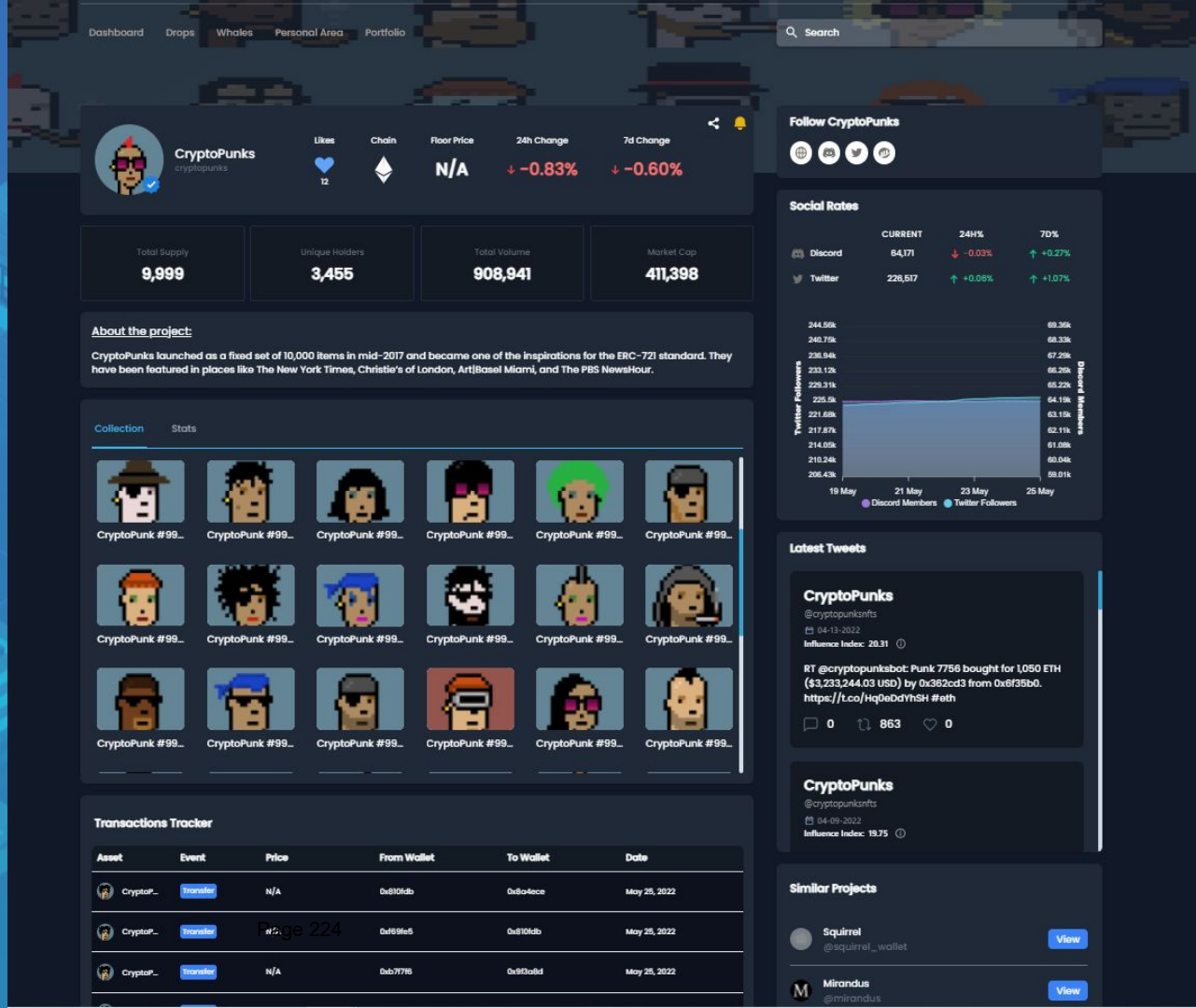


Our platform

Project Page

Key Features -

- NFTs available on OpenSea
- Alerts button
- Recent transactions
- Similar projects
- Project social outlets



Monetization

1

Affiliation with OpenSea.io

Users have the option to select an NFT and continue to opensea.io to make a purchase, an affiliation fee for every purchase will be made

2

Ads & Promotions

NFT projects who want to stand out, will be able to be featured and highlighted in the platform

3

Subscription

Freemium model, users can take full advantage of the platform with a subscription

*B2B Social Analytics Dashboard by 2023

GTM Strategy

Instagram, YouTube, Twitter, Reddit, Discord and telegram

Influencer Marketing

Affiliate and refer a friend program

Feature NFT projects in exchange for publicity

Competitors

	NFT Scoring	CryptoNews.net	Signex
Discord Growth Analytics	✓	✗	✓
Twitter Growth Analytics	✓	✗	✓
Twitter & Discord Activity Analysis	✗	✗	✓
Sentiment & Volume Analysis	✗	✗	✓
NFT Project Alerts*	✗	✗	✓

* based on social metrics

Social Analysis

*__Signex.io__

*NFTscoring.com

Rarity Score

*Rarity.tools

*RaritySniper.com

*TheNFTscore.com

*RankNFT.io

*HowRare.is

Marketplaces

*OpenSea.io

*MagicEden.io

*Binance.com

*Rarible.com

*Niftygatway.com

*Superrare.com

Technical Analysis

*Dappradar.com/nft

*NFT-stats.com

*Nonfungible.com

*Nansen.ai

*Icy.tools

*Crypto.com

*Moby.gg

* Bitdegree.org

Our Team

Amnon Sarig

Chairman

Experienced serial entrepreneur

4th startup, 2 exits and

Co-Founder @ **TuneWiki**

Math / CS, MBA.

Itay Dekel

CEO

2nd startup, cashout in 1st - **Speroll** (AdTech)

Sales, marketing and social executive.

Involved in Crypto & NFTs since 2017

Villi Katrih

CTO

Co-Founder & CTO @ **KLEVER**

CTO @ **Authentic**

Co-founder @ **Chatway**

Full stack engineer @ Seeya

Built various products in Cyber Security, IoT and SaaS enterprise platforms

Advisors

Maxime De Mey

R&D Engineer

Part owner of Discord server with over 150K

*CoFounder of "**Universols**" NFT*

Itay Gissin

Entrepreneur

*CEO of **investing.com** - 2014-2018*

*Co-Founder & CEO - **Grant Canyon***

Arthur Garmider

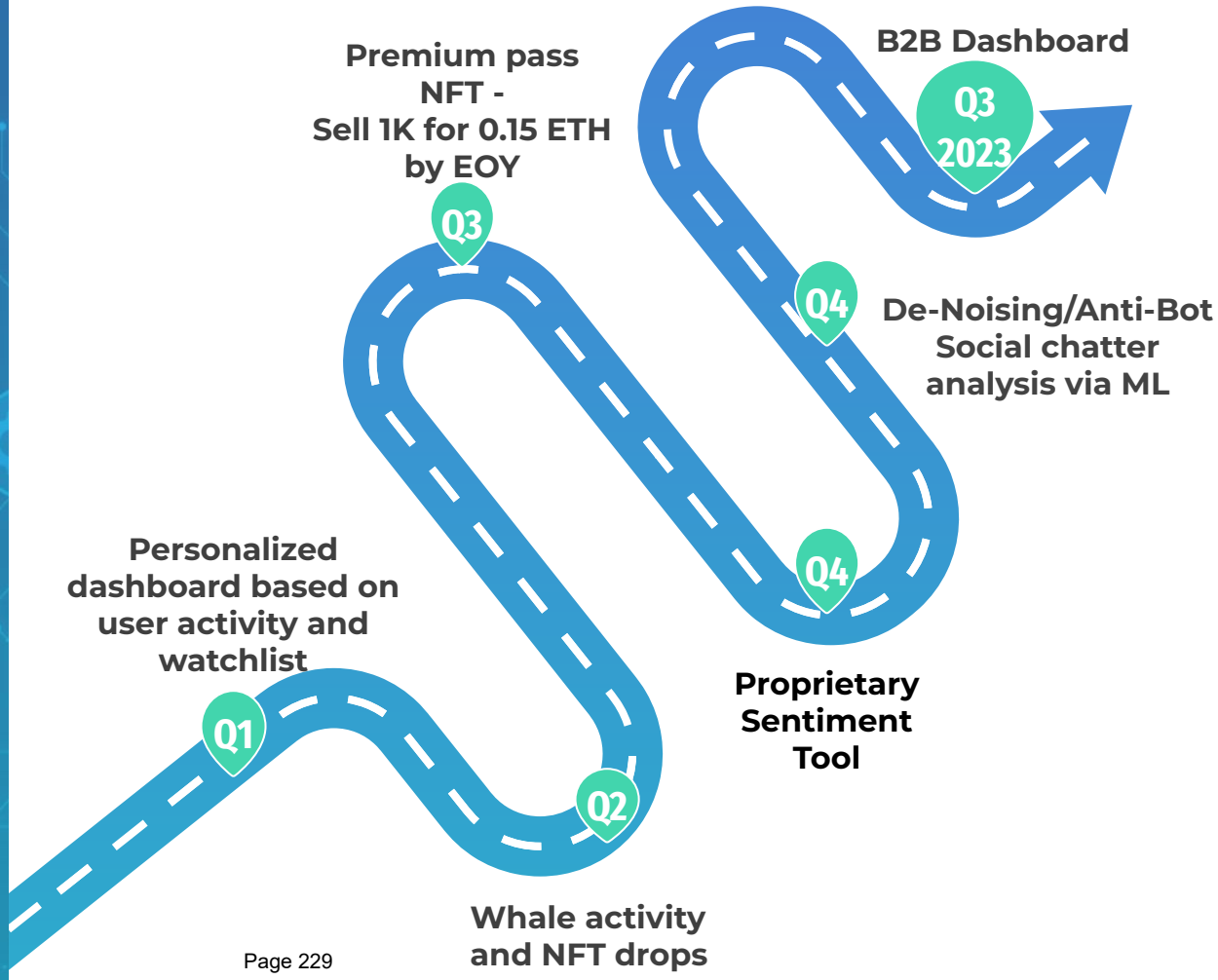
Experienced CTO and Data Scientist

*CTO @ **Xertive***

*Senior System Architect @ **Payoneer***

*Director of Data Science @ **WeissBeerger***

Product Roadmap



*Quarters 1-4 (1 year)

Product Roadmap

B2C Dashboard Selected Upcoming Features-

- Buy NFTs directly from platform including social or price change buy signals
- Upcoming drops with community score
- Reddit and Telegram community analysis
- Rarity Score
- Community actions breakdown (likes, emojis, retweets, comments)
- Users activity breakdown (Ranking community members' engagement from very active to not active)
- Assets supply and demand
- Technical analytics including total daily sales, sales volume, trending projects, trade chart
- Value rank - ratio between assets price and rarity
- Price change prediction

B2B Dashboard Selected Upcoming Features -

- Members activity break down
- Members activity score
- Members on other NFT servers break down
- Twitter influencer engagement and hashtags information
- Twitter influencers social graph (who is following who)
- NFT projects minting history details (followers/activity, sale price, NFTs sold)

1

Scan and increase volume and data sources supply

Scan and spot bots and fake activity using Machine learning, Big data and A.I to distinguish between real communities and fake ones

2

Expand our community and reach top tier influencers

Use high profile influencers to spread the word

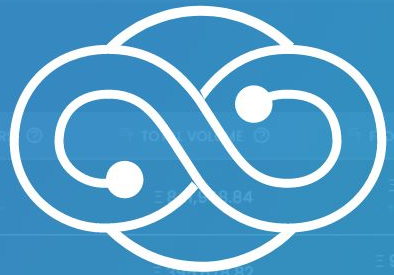
3

Expand the development and build marketing team

Create NFT and token to support an internal ecosystem

SIGNEX.IO

www.signex.io

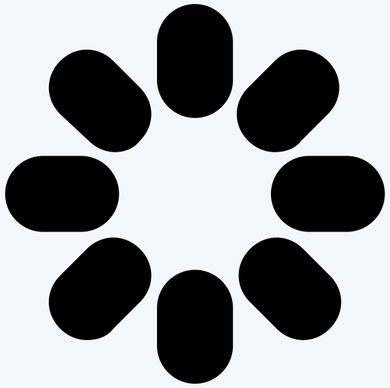


SIGNEX.IO

Itay Dekel

itay@signex.io
+972-525397911

The Problem



Extensive waiting list

Boring process

A poor solution to remote physical therapy

TheraPlay
we play therapy


TheraPlay
we play therapy


TheraPlay
we play therapy
Therapist
Player

Therapist interface

Patients interface

Play TheraPlay

TheraPlay
we play therapy

The Team

Tamir Elazar

CEO- Co-Founder

- Former business owner
- Project manager at Oazis venture builder and accelerator

Maor Oz

CTO- Co-Founder

- M.Sc. student in electrical and computer engineering
- Algorithm and deep learning developer at IAI

Aviv Ackerman

COO-Co-Founder

- BA Psychology and management
- Director of business & sales at K Logic Group

Prof. Ofer Hadar

CSO- Co-Founder

- Ph.D. in Electrical and Computer Engineering.
- EX Chairman of Communication Systems Engineering at BGU.

Gal Shavit

Chief Medical Officer

- B.O.T occupational therapy
- M.A special education
- Former team leader of the occupational therapists at the child development clinic at Sheba medical center

Advisory Borad:

Igal Rotem- CEO of Finaro, Founder of Power Design

Moshe Elazar- President & CEO of Aeronautics Group

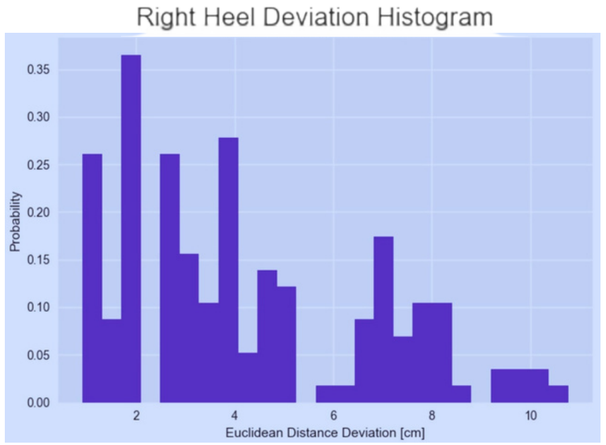
Gadi Bahat- Executive Director at LEADERS, former CEO of Compass-eos

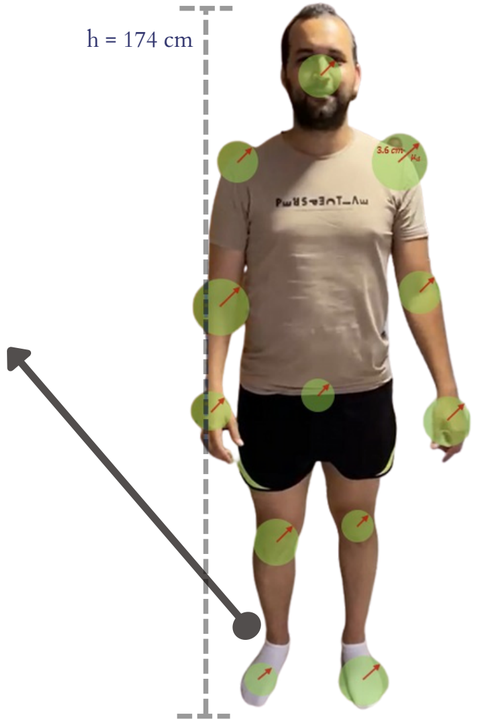
Strategic partnership:

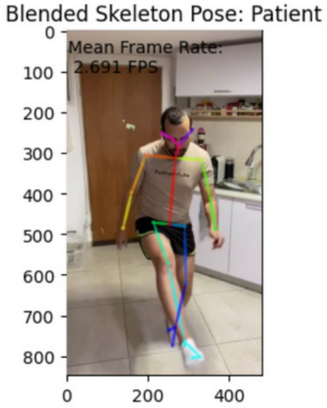
Michal Hochhauser .Ph.D.- Head of the OT department at Ariel University

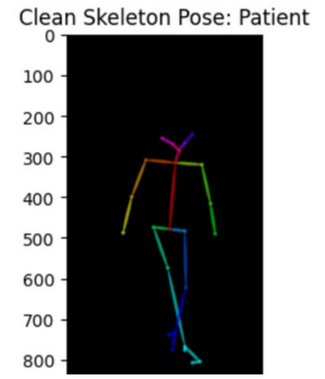
TheraPlay
we play therapy

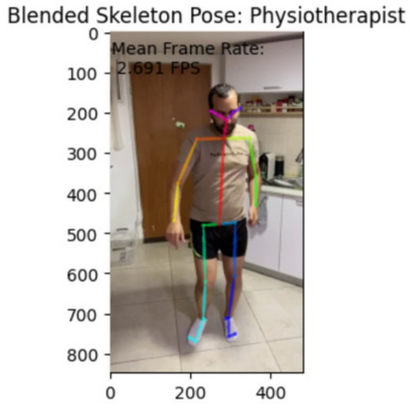
The Technology

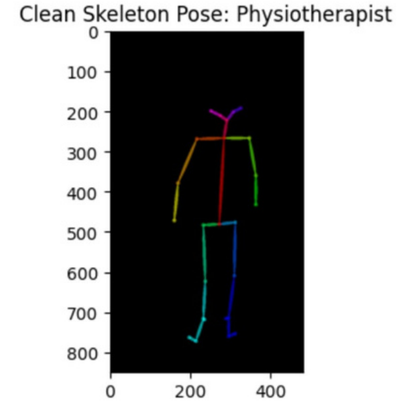




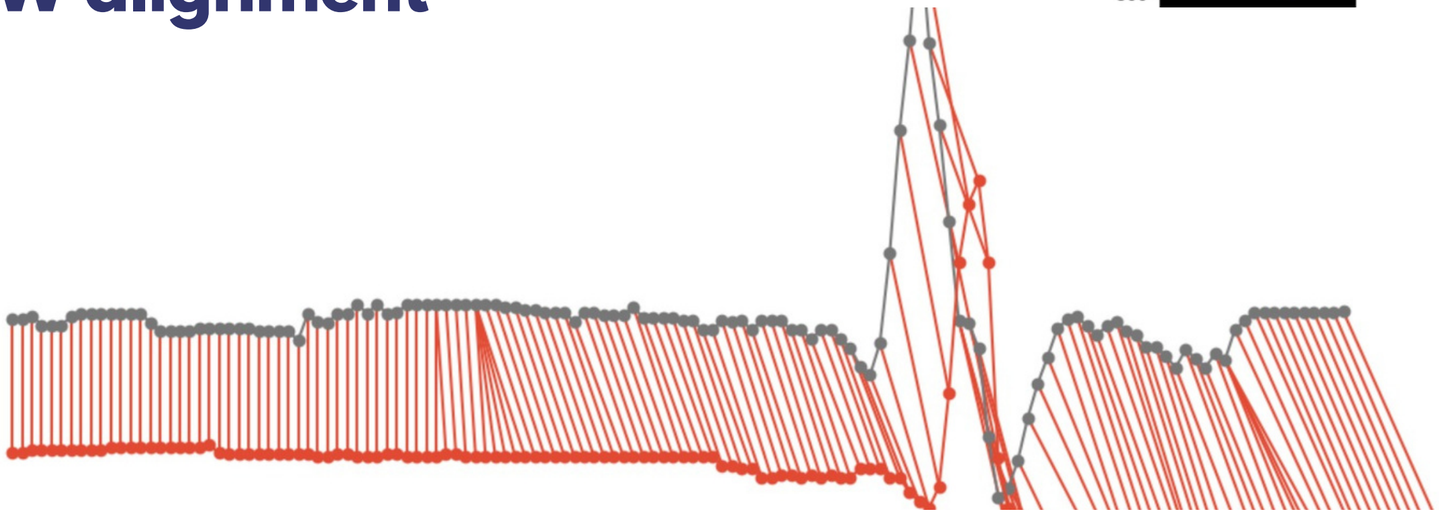








2D-DTW alignment



3D kinematics movement analysis

TheraPlay
we play therapy

Target Market

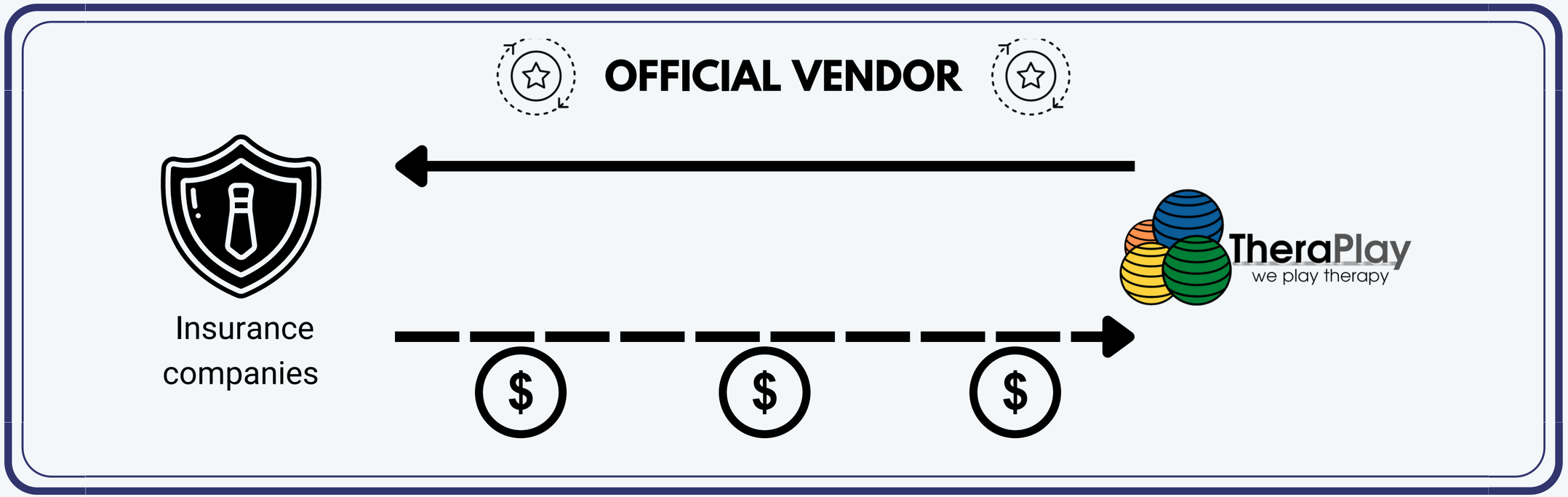
Physical and occupational therapy

Go To Market

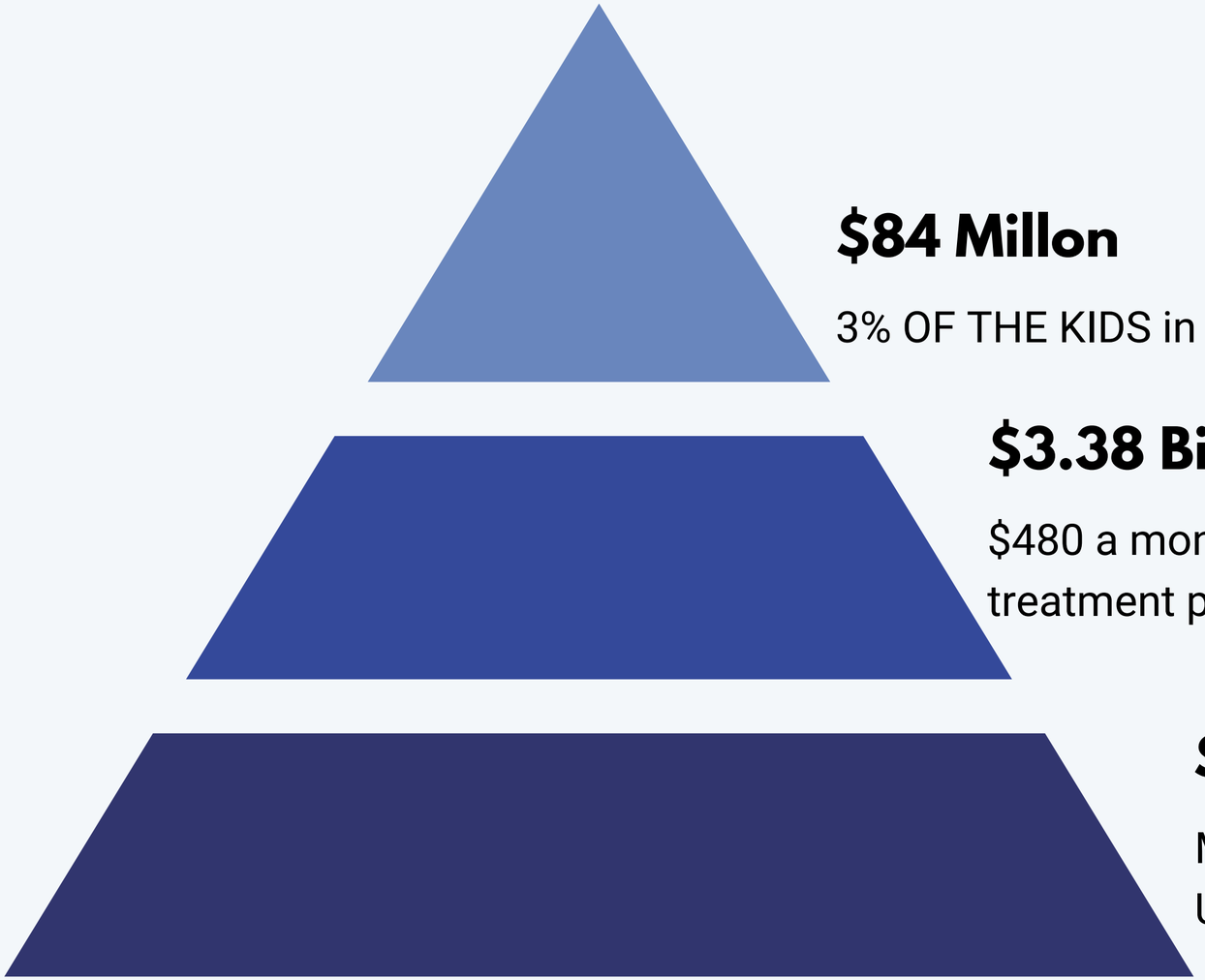
Pediatric Physical and Occupational Therapy, ages 4-12, with DCD-related challenges

TheraPlay
we play therapy

B2B


OFFICIAL VENDOR
Insurance companies
TheraPlay
we play therapy
$
$
$

The Market



$84 Millon

3% OF THE KIDS in our price (**SOM**)

$3.38 Billion

$480 a month X 4 months (average treatment process duration)(**SAM**)

$49.28 Billion

Motor therapy skills market in the United States (**TAM**)

Timeline

POC

We are here

US Launch

11/23

SAAS license model

11/26

12/22

MVP Pilot

6/25

Global Launch

TheraPlay
we play therapy

TheraPlay

we play therapy

tamirelazr@gmail.com
0542483166

Thank you!

Cyberblocks Value Proposition

Why now?

Cyber insurance turned unprofitable in 2020. Indeed, Erin Ayers (Advisen) warns:[1]

> *Cyber insurers "urgently" need to evaluate their approach to managing the risk from pricing to modeling to risk selection, otherwise the future of the line looks "grim."*

There is demand for novel solutions. Cyberblocks (CB) offers such a solution by providing a neutral platform intermediating cyber insurers and their clients. CB monitors and collects data in a way that aligns incentives and respects client privacy.[2] The CB platform combines both external and internal (client-specific) data, and pools it together for further risk analysis.

CB value added

CB creates value by enabling the cyber insurers to identify low risk customers and reward them with lower pricing. The added value of CB platform comes from:

1. Improvement of risk assessment & finer division into risk tranches.
2. Fraud reduction.
3. Expansion of cyber contracts offerings.

CB entry strategy

Initially, CB will work with logging data that each client already collects. CB will aggregate and analyze existing logging information.

After a 6 months of being connected to the CB platform, if the client satisfies the pre-specified security practices, they become eligible for “CB pricing”, which is more favorable than the present-day cyber insurance pricing.

Each client can opt out from using the CB platform, and continue to use their original pricing.

Insurers could offer the CB services to their own clients, as well as to prospective clients.

[1] See https://www.advisen.com/tools/fpnproc/fpns/articles_new_35/P/399627085.html

[2] Speaking technically, the CB platform improves information and reduces information asymmetry. CB creates value by alleviating principal-agent conflict between the cyber insurers and their clients.

SWOT for Cyber Blocks

S strengths

1. Positioning: to Improve incentives in cyber insurance ecosystem
   – Neutrality: unbiased platform
   – Transparency & Immutability
   – Possibility of full privacy of data collection
   – Low cost
   – High scalability

2. Team: cross domain team expertise; focus on Cyber Physical Systems, incl. large scale national critical infrastructures

W weaknesses

1. Nearly impossible to assess client benefits in advance: difficult to access risks due to
   – fat tails
   – dynamic nature of cyber risks (no stable risk distribution exists)
   – CB lack of access to insurer data: both, premiums and claims

2. Costs depend on blockchain ecosystem

O opportunities

1. First to market

2. Massive Data collection allows to improve
   – data standardization
   – risk models and risk assessment
   – risk-based client segmentation

3. Market growth: CB platform allows to
   – divide clients into tranches
   – richer cyber insurance contracts
   – market expansion

T threats

1. Competition
   – Replicating market entrants such as Security Information and Event Management (SIEM) vendors, Cloud providers, etc.

2. Technological breakthroughs, e.g.,
   – successful quantum computer

3. New laws / regulations, e.g.: wrt/
   – privacy
   – public blockchains