DSC Capstone LaTeX Template

Ryan Batubara rbatubara@ucsd.edu
Ivy Hawks mahawks@ucsd.edu
Darren Ho dah103@ucsd.edu
Ciro Zhang ciz001@ucsd.edu

Shachar Lovett slovett@ucsd.edu

Abstract

TBA

Repository: https://github.com/rybplayer/DSCCapstone

1 Introduction

2 Cryptography

3 Game Theory

3.1 Background

The field of game theory studies the mathematical models of interactions between rational decision makers. Game theory initially addressed the interactions between two parties in a zero sum game but has grown into an umbrella term on rational decision making.

3.2 Foundational Problems

3.2.1 Algorithmic Mechanism Design

Algorithmic mechanism design (AMD) focuses on designing a mechanism where (a) each party behave in a certain way and (b) is computationally efficient. AMD differs from regular mechanism design in that mechanism design assumes infinite compute power. AMD seeks to find a mechanism that performs with certain constraints such as polynomial time.

3.2.2 Vickrey Auction

Vickrey Auction is a sealed-bid second-place auction, where the highest bidder pays the second highest price. This is a classical example of AMD because it is in the best interest of each party to bid truthfully.

Say we have three parties participating in this auction, Alice, Bob, and Charlie. Their sealed bids are:

- Alice: $50
- Bob: $100
- Charlie: $70

In this scenario, Bob wins and pays $70.

Bob cannot improve his results by changing his bid, if he raises it the price he pays does not change, and if he lowers it he risks not winning the bid. Hence in this game, there is a dominant strategy to bid truthfully.

3.2.3 Nash Equilibrium

Nash Equilibrium are a foundational problem within game theory that involve finding a state within a scenario where no player can improve their outcome by unilaterally changing their own strategy.

3.2.4 The Prisoner's Dilemma

The prisoner's dilemma is one of the simplest examples of finding a **Nash Equilibrium**.

Two suspects, Prisoner A and Prisoner B, are arrested and interrogated separately. Each prisoner has two possible actions:

- **Cooperate (C)**: remain silent
- **Defect (D)**: betray the other prisoner

The payoff matrix (with payoffs in the form (A's payoff, B's payoff)) is given by:

	B: C	B: D
A: C	$(1,1)$	$(3,0)$
A: D	$(0,3)$	$(2,2)$

The interpretation is:

- If both stay silent (C,C), each receives a light sentence: 1.
- If one defects and the other cooperates, the defector goes free (0) while the cooperator receives the harshest sentence (3).
- If both defect (D,D), they each receive a moderate sentence (2).

Even though mutual cooperation leads to a better outcome than mutual defection, the dominant strategy for each prisoner is to defect, making (D, D) the unique Nash equilibrium.

3.3 Case Study: Pure Nash Equilibrium

We will be looking to see what the communication complexity of reaching equilibrium is. More specifically, the communication model in which players initially only know their own utility functions. We want to analyze how much information must be transferred between them to jointly compute the equilibrium point. We also assume that each player follows a predetermine protocol, abstracting any other incentives of each player. We will be focusing on the communication complexity of *pure* Nash Equilibrium. A pure Nash Equilibrium being a situation where there is guaranteed to be a unique best input for a all given parties.

3.3.1 Setting for Pure Action Games

There are $n \geq 2$ players, $i = 1, 2, ..., n$. Each player i has a finite set of actions A_i with $|A_i| \geq 2$. For the analysis of pure action games, we only consider binary action games, that is, for

each i, $A_i = \{0,1\}$. Let the joint action space $A = \prod_{i=1}^{n} A_i$. Each player has a private utility function $u_i : A \rightarrow \{0,1\}^n$ which we are assuming are finitely represented for simplicity.

For a joint action $a = (a_1, ..., a_n) \in A$ (for binary action games, a joint action is an n bit binary string), let a^{-1} denote the joint action of all players except player i. A joint action a is a **Pure Nash Equilibrium** if $u_i(a) \geq u_i(b_i, a^{-1})$ for every player i and any action $b_i \in A_i$. That is to say that for all players choosing the alternative option would result in an equal or worse outcome for them.

3.3.2 Communication Complexity of Pure Nash Equilibrium

Theorem *Any pure Nash equilibrium procedure has communication complexity* $\Omega(2^n)$

Claim If there exists a reduction from the S-disjointness problem to an n-person pure Nash Equilibrium procedure that satisfies reducibility and constructibility properties then any pure Nash Equilibrium procedure has communication complexity of at least $|S|$ bits since it has been proven that $CC(DISJ_S) = |S|$.

We want to prove that there exists a reduction from the set disjointness problem where:

$$S_1 \cap S_2 = 0 \text{ or } S_1 \cap S_2 \neq 0$$

Our goal is to relate the disjointness problem into a game that has a pure Nash equilibrium iff the sets intersect.To do so, we use the *matching pennies reduction*.

Matching Pennies. The matching pennies game is a 2×2 zero-sum game played between two players. Each player chooses either *Heads* (H) or *Tails* (T). If the actions match, player 1 wins; if they mismatch, player 2 wins. The payoff matrix for player 1 is:

	H	T
H	1	-1
T	-1	1

and player 2 receives the negative of this payoff.

A key property of matching pennies is that it has *no pure Nash equilibrium*. This makes it useful as a "destabilizing gadget" in reductions, since attaching a matching-pennies subgame to an action profile guarantees that the profile cannot be a pure equilibrium.

For $n \geq 4$, the matching pennies reduction satisfies the reducibility and constructibility properties.

We first verify the *constructibility* property. Fix $\alpha \in \{1,2\}$. By the definition of the matching pennies reduction, the payoff function $u_{\alpha,i}(a)$ of every player (α, i) depends only on the following information:

- whether $a \in S_\alpha$, and
- the actions $a_{h,1}$ and $a_{h,2}$ in case $a \notin S_h$

Since S_α is the private input of the agents on side α in the disjointness problem, the event $a \in S_\alpha$ is computable by those agents, and $a_{\alpha,1}, a_{\alpha,2}$ are part of the observed joint action a. Hence each payoff $u_{\alpha,i}(a)$ is computable from (a, S_α, i) alone. This establishes constructibility.

We now prove the *reducibility* property. We must show that

$$S_1 \cap S_2 \neq \emptyset \qquad \iff \qquad G \text{ has a pure Nash equilibrium.}$$

Suppose $a \in S_1 \cap S_2$. Then by construction, every player in T_1 and T_2 receives payoff 2 at a, which is the maximal payoff any player can obtain. Thus no player can improve by deviating, and a is a pure Nash equilibrium.

Suppose $a \notin S_\alpha$ for some $\alpha \in \{1, 2\}$. Then the two designated players $(\alpha, 1)$ and $(\alpha, 2)$ play a matching pennies game at a. The matching pennies game has no pure Nash equilibrium: at every pure action profile of the two players, one of them strictly benefits from unilaterally deviating. Therefore, at a at least one of the players $(\alpha, 1)$ or $(\alpha, 2)$ has a profitable deviation, implying that a cannot be a pure Nash equilibrium of G. Hence no pure Nash equilibrium can lie outside $S_1 \cap S_2$.

Combining the two directions, we conclude that a pure Nash equilibrium exists in G if and only if $S_1 \cap S_2 \neq \emptyset$.

Since we have proven the claim, we know that the communication complexity of any pure Nash Equilibrium is $CC(DISJ_s) = |S|$. Remember that $S = \{0,1\}^n$ so therefore $|S| = 2^n$ proving the theorem.

3.4 Broad Applications

The communication complexity of Game Theory has many varied applications and is prevalent in any situation that involves optimizing the output in any multiparty game. Game Theory draws naturally has parallels to other fields such as multi-agent and distributed systems.

4 Machine Learning

There are lots of ways to use CC to prove lower bounds for data structures. Here we limit ourselves to the approximate nearest neighbor problem with static (unmodifiable and only queriable) data structures.

4.1 Introducing Nearest Neighbors (NN)

The *nearest neighbor problem* takes in a set S of n points in a metric space (X, ℓ). Most commonly this is $\mathbb{R}^d$ with the ℓ_2 norm. Here we use the *hamming cube* $X = \{0,1\}^d$ and ℓ is the Hamming distance. Let d be large, say $d = \sqrt{n}$.

The goal is to build a data structure D (as a function of $S \subseteq X$) that prepares for all possible nearest neighbor queries. A query is a poiny $q \in X$, and the end goal is to return $p^* \in X$ that minimizes $\ell(p,q)$ over all $p \in S$.

We use the Hamming cube because it is easier to map into communication complexity, though the key high-level ideas are the same for analyzing Euclidean and other metric spaces.

If computing $\ell(p,q)$ take $O(d)$ time, then the brute force algorithm takes $O(dn)$ time. Alternatively, precomputing the answer to all possible queries takes $\Theta(d2^d)$ space. The exact NN problem thus suffers from an exponential-size data structure in terms of d. This is known as the *curse of dimensionality*.

4.2 Approximate Nearest Neighbors (ANN)

The $(1+\epsilon)$-approximate NN, where we only need to return p where $\ell(p,q) \leq (1+\epsilon)\ell(p^*,q)$, we can do much better. Let $\epsilon = 1$ or 2, so it is not too small but still practically relevant.

The high-level idea is to hash nearby points into the same bucket, then precompute the answer for each bucket. Therefore the key challenge is to make sure nearby points hash to the same bucket.

4.2.1 Decision ANN

Consider the *decision ANN* version, where given $L \in \{0,1,\ldots,f\}$, we wish to distinguish between

1. There exists a point $p \in S$ with $\ell(p,q) \leq L$.
2. $\ell(p,q) \geq (1+\epsilon)L$ for every $p \in S$.

If neither applies, both answers are correct.

Note that if $|S| = 1$, this is oddly similar to equality: Equality is the same as deciding between hamming distance 0 or not. The public coin protocol where Alice takes the inner product mod 2 of x with r_1 and r_2 (2 bits) and sends it to Bob always accepts $x = y$ and accepts $x \neq y$ with probability $1/4$.

It is also similar to gap-hamming, which we showed was hard in Chapter 2 using a reduction from index.

Consider the CC problem where Alice and Bob want to decide if $\ell_H(x,y)$ between inputs $x,y \in \{0,1\}^d$ is at most L or at least $(1+\epsilon)L$. Call this *ϵ-Gap Hamming*. Define the following protocol:

1. Alice and Bob create $s = \Theta(\epsilon^{-2}\log(?))$ random strings $R = \{r_1,\ldots,r_s\} \in \{0,1\}^d$ where $d = \theta(\epsilon^{-2})$ from public coins. Each entry is independent but not uniform: there is a $1/2L$ probability of each entry in r_i being equal to 1.
2. Alice sends s inner products of r_i with x. This is the "hash value" $h_R(x) \in \{0,1\}^s$.

3. Bob accepts if any only if the Hamming distance between $h_R(y)$ and $h_R(x)$ differs by some small number of bits.

We defer the analysis of the small number of bits to the book. The key idea is as follows:

1. When selecting a string r_i, we are choosing a coordinate to be relevant (be a 1, and thus contribute to the inner product later) with probabiliy $1/L$. Consider it irrelevant otherwise. Think of this as selecting relevant coordinates for this r_i hash.
2. The second stage, the inner product, is a modified equality with these coordinates only. We see that

$$\Pr_j[\langle r_j, x\rangle \not\equiv \langle r_j, y\rangle \mod 2] = \frac{1}{2}\left(\left(1-\left(1-\frac{1}{L}\right)^{l_H(x,y)}\right)\right).$$

This results in CC $\Theta(\epsilon^{-2})$.

4.2.2 Data Structure Decision ANN

We now convert the above algorithm into a data strcuture for the $(1+\epsilon)$ decision ANN problem.

1. Given a point set P of n points in $\{0,1\}^d$, choose a set of R of $s = \Theta(\epsilon^{-2}\log n)$ as above.
2. Define $h_R : \{0,1\}^d \to \{0,1\}^s$ by setting the jth coordinate to $\langle r_j, x\rangle \mod 2$.
3. Make a table with $2^s = n^{\Theta(\epsilon^{-2})}$ buckets, indexed by s-bit strings.
4. For all $p \in P$, insert p into every bucket b which $\ell(h_R(p), b) \leq (t + 1/2 h(\epsilon))s$ where $h(\epsilon) = \frac{1}{2\epsilon^2}(1-\epsilon^{-\epsilon})$.

This data structure has probability at least $1-\frac{1}{n}$ to give the correct answer in $n^{O(\epsilon^{-2})}$ space.

We will show later that with this level of accuracy this space is optimal. Note that it is possible to do better by inreasing the query time.

Note that in the real problem we do not have this value L. We would want it to be the actual NN distance of a query point q (since L counts the number of differences / coordinates checked), but this can change from one q to another. Furthermore, the data strcuture may be wrong on as many as $2^d/n$ queries. Knowing the coin flips allow an adversary to exhibit queries that will be wrong.

Fix 1: Make d copies of the data structure for each value of L^2. Answering takes $O(\log d)$ lookups. Space blows up by $\Theta(d\log d)$. Query time blows up by $\Theta(\log\log d)$.

Fix 2: Make $\Theta(d)$ copies and take the majority vote. This is wrong in at most inverse exponential of d. Here not even an adversary who knows the coin flips knows whether a particular query will be incorrect ahead of time. This blows up space and query time by $\Theta(d)$.

Fix 3: Make $\Theta(d)$ copies and choose uniformly randomly one of the copies to answer. The space blows up by $\Theta(d)$ but query time is unaffected. The probability of a correct answer is at least $1-\Theta(\frac{1}{n})$.

Combining the three lemmas:

- Space: $O(d^2 \log d) \cdot n^{\Theta(\epsilon^{-2})}$
- Query time: $O(\epsilon^{-2} d \log n \log\log d)$.

That is, polynomial space for logarithmic query time.

4.3 The Cell Probe Model

We start by motivating the cell probe model. The goal is to create a database D to answer Q queries, known beforehand. The encoding scheme must work for all possible databases: In ANN, P is the database (unknown beforehand) and Q are elements of $\{0,1\}^d$ (known). Another example is set membership.

A parameter of the cell probe model is *word size* w. The design space if the number of ways to encode D as s cells of w bits each. We say s is the space used in the encoding. It must correctly answer all $q \in Q$; it does this by reading cells one at a time, giving w bits each. The query time of this algorithm is the max, over all D and all $q \in Q$, number of accesses to answer the queries.

w is typically large so a single element of the database can be stored in a cell, and not much larger than this.

For ANN, take w to be polynomial in $\max\{d, \log_2(n)\}$. The previous construction solves ANN cell probe with space $n^{\Theta(\epsilon^{-2})}$ and query time 1. We now show a matching lower bound in the cell-prove model; thus constant query time can only be achieved by encodings that us $n^{\Omega(\epsilon^{-2})}$ space.

4.3.1 Query Database

We start with a contrived problem and build connections later.

Consider the *query database* problem. Alice has a query q and Bob gets database D. We wish to compute the answer to q on the database D.

Let there be a cell-probe encoding of query database with word size w, space s, and query time t. Then there is a CC protocol with CC at most

$$\underbrace{t \log_2(s)}_{Alice} + \underbrace{tw}_{Bob}$$

Alice simulates the query-answering algorithm, sending $\log_2(s)$ to Bob for every cell requested. Bob sends back w bits to describe the contents of the selected cell. They need to go back and forth only t times (since the query time is also the number of $\Theta(1)$ queries, i.e. t queries).

[*Richness Lemma*] Let $f : X \times Y \to \{0,1\}$ be a Boolean function with corresponding $X \times Y$ boolean matrix $M(f)$. Assume that

1. $M(f)$ has at least v columns that each have at least u 1-inputs.
2. There is a deterministic protocol that computes f in which Alice and Bob send a and b bits respectively.

Then $M(f)$ has a 1-rectangle $A \times B$ with $|A| \geq u/2^a$ and $|B| \geq u/2^{a+b}$.

For every $\epsilon, \delta > 0$ and sufficiently large n, in every CC protocol that solves $(\frac{1}{\epsilon^2}, n)$-disjointness with a universe of size $2n$, either

1. Alice sends at least $\delta/\epsilon^2 \log_2 n$ bits, or
2. Bob sends at least $n^{1-2\delta}$ bits.

$(\frac{1}{\epsilon^2}, n)$-disjointness reduces to the query database problem for the decision problem of ANN with $(1+\epsilon)$.

Every data structure for the decision version of $(1+\epsilon)$-ANN with query time $t = \Theta(1)$ and word size $w = O(n^{1-\delta})$ for $\delta > 0$ uses space $s = n^{\Omega(\epsilon^{-2})}$.

The proof is quite involved and is deferred for later.