Bipartite Matching is in Catalytic Logspace

Aryan Agarwala
Max-Planck-Institut für Informatik
aryan@agarwalas.in

Ian Mertz*
Charles University
iwmertz@iuuk.mff.cuni.cz

Abstract

Matching is a central problem in theoretical computer science, with a large body of work spanning the last five decades. However, understanding matching in the time-space bounded setting remains a longstanding open question, even in the presence of additional resources such as randomness or non-determinism.

In this work we study space-bounded machines with access to catalytic space, which is additional working memory that is full with arbitrary data that must be preserved at the end of its computation. Despite this heavy restriction, many recent works have shown the power of catalytic space, its utility in designing classical space-bounded algorithms, and surprising connections between catalytic computation and derandomization.

Our main result is that bipartite maximum matching (MATCH) can be computed in catalytic logspace (CL) with a polynomial time bound (CLP). Moreover, we show that MATCH can be reduced to the lossy coding problem for NC circuits (LOSSY[NC]). This has consequences for matching, catalytic space, and derandomization:

- **Matching**: this is the *first* well studied subclass of P which is known to compute MATCH, as well as the *first* algorithm simultaneously using sublinear free space and polynomial time with *any* additional resources. Thus, it gives a potential path to designing stronger space and time-space bounded algorithms.
- **Catalytic space**: this is the *first* new problem shown to be in CL since the model was defined, and one which is *extremely* central and well-studied. Furthermore, it implies a strong barrier to showing CL lies *anywhere* in the NC hierarchy, and suggests to the contrary that CL is even more powerful than previously believed.
- **Derandomization**: we give the *first* class $\mathcal{C}$ beyond L for which we exhibit a natural problem in LOSSY[$\mathcal{C}$] which is not known to be in $\mathcal{C}$, as well as a *full derandomization of the isolation lemma* in CL in the context of MATCH. This also suggests a possible approach to derandomizing the famed RNC algorithm for MATCH.

Our proof combines a number of strengthened ideas from isolation-based algorithms for matching alongside the compress-or-random framework in catalytic computation.

*Partially supported by grant 24-10306S of GA ČR and supported by Center for Foundations of Contemporary Computer Science (Charles Univ. project UNCE 24/SCI/008).

1 Introduction

In this work we study a number of key questions and models of complexity between logarithmic space (L) and polynomial time (P). In particular we focus on the relationship between *bipartite maximum matching* (MATCH) and *poly-time bounded catalytic logspace* (CLP), as well as their implications for efficient parallel algorithms (NC) and efficient time-space algorithms ($\mathsf{TIME\text{-}SPACE}(\mathrm{poly}(n), n^{1-\Omega(1)})$). Lastly we draw on connections between both MATCH and CLP to problems in derandomization—for the former we focus on the *isolation lemma*, and with regards to the latter we discuss reductions to the *lossy coding problem*—to make progress therein.

1.1 Matching and catalytic computation

Matching. In the MATCH problem, we are given a bipartite graph G as input and our goal is to return a subset of edges of maximum size such that no two edges share an endpoint. MATCH has been a central problem in the study of complexity since its inception, and was one of the earliest problems to be studied with respect to time; it has been known for 70 years that MATCH can be solved in P [Kuh55; HK73]. However, we are not aware of any well-studied class[1] $\mathcal{C} \subseteq \mathsf{P}$ such that $\mathsf{MATCH} \in \mathcal{C}$.

Question 1: Is MATCH in any subclass of P?

For two such classes $\mathcal{C}$ in particular, namely NC and SC, proving this would be a major breakthrough [Lov79; KUW85; MVV87; MN89; BBRS98; ARZ99; MV00; DKR10; FGT16; GG17; ST17; AV19; AV20]. With regards to parallel complexity, a long line of work culminated in showing that MATCH can be solved in randomized NC (RNC) [MVV87] and in quasi-polynomial size NC ($\mathsf{Quasi\text{-}NC}$) [FGT16], but despite decades of research no true NC algorithms are known to this day.

Question 2: Is $\mathsf{MATCH} \in \mathsf{NC}$?

Matching holds an even more central place in the study of space and time-space efficiency. It is widely conjectured that logarithmic space is *insufficient* to solve MATCH (i.e. $\mathsf{MATCH} \notin \mathsf{L}$), and proving so would give a breakthrough separation between L and P. Similarly, with regards to time-space complexity we have no algorithms which compute MATCH in polynomial time and sublinear space even given additional resources such as non-determinism or randomness, and it is unclear whether such algorithms should exist or not.

Question 3: Do there exist any resources $\mathcal{B}$ such that
$\mathsf{MATCH} \subseteq \mathcal{B}\mathsf{TIME\text{-}SPACE}(\mathrm{poly}(n), n^{1-\Omega(1)})$?

In this work we study another such class called *catalytic logspace* (CL), and in particular its poly-time variant CLP, which is of great interest in relation to both NC and SC. In doing so we give a first-ever solution to Questions 1 and 3, as well as a potential barrier—or approach—to resolving Question 2.

[1] Throughout this paper, when we discuss classes containing MATCH we ignore granular poly-time classes which immediately follow as a direct consequence of the above algorithms, e.g. $\mathsf{TIME}[n^2]$, or those that contain matching by definition, i.e. the class of problems reducible to MATCH.

Catalytic computing. In catalytic computing, a space-bounded machine is given additional access to a much longer "catalytic" tape, which is additional memory already full of arbitrary data whose contents must be preserved by the computation. CL is the class of problems solvable by a logspace machine augmented with a polynomial length catalytic tape, and CLP is the subclass of CL where the machine is additionally required to run in polynomial time.

The framework of catalytic computation was formally introduced by Buhrman et al. [BCKLS14] in order to understand the power of additional but used memory. It was informally conjectured earlier, in the context of the *tree evaluation problem* [CMWBS12], that used memory could not grant additional power to space bounded machines. However, [BCKLS14] showed that CL and even CLP contain problems believed to not be in L:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{TC}^1 \subseteq \mathsf{CLP} \subseteq \mathsf{CL} \subseteq \mathsf{ZPP}$$

In the same work, they state that it is unclear what their result implies about the strength of CL. Particularly, *is* $\mathsf{L} \subsetneq \mathsf{CL}$*, or is the intuition that catalytic space does not grant additional power indeed correct, thus giving an approach to proving* $\mathsf{L} = \mathsf{NL} = \mathsf{TC}^1 = \mathsf{CL}$*?*

Question 4: Where does CL lie between L and ZPP?

Since their result, many works have studied the utility and power of catalytic computation (see e.g. [BKLS18; GJST19; DGJST20; BDS22; CM22; GJST24; Pyn24; CLMP25; FMST25; KMPS25; PSW25]). These results and techniques have also seen applications for ordinary space-bounded computation, such as 1) work on the *Tree Evaluation Problem* by Cook and Mertz [CM20; CM21; CM24] as well as a subsequent breakthrough by Williams [Wil25] for simulating time in low space; and 2) win-win arguments for derandomization and other questions in logspace [DPT24; LPT24; Pyn24] such as a recent result of Doron et al. [DPTW25] showing that, in an instance-wise fashion, either $\mathsf{NL} \subseteq \mathsf{SC}$ or $\mathsf{BPL} \subseteq \mathsf{NL}$. We refer the interested reader to surveys of Koucký [Kou16] and Mertz [Mer23] for more discussion.

However, despite all of the aforementioned work, *no problems outside of* TC^1 *have been shown to be in* CL[2]. Thus, it is still possible that $\mathsf{CL} \subseteq \mathsf{TC}^1$, which has led to conjectures, such as that of [Kou16; Mer23], that CL can indeed be computed *somewhere* in the NC hierarchy.

Question 5: Is $\mathsf{CL} \subseteq \mathsf{NC}$?

Our results (1). In this work we address and connect all of the aforementioned questions by showing that catalytic logspace can compute bipartite matching in polynomial time:

Theorem 1.1.

$$\mathsf{MATCH} \in \mathsf{CLP}$$

We make a few notes on the power of Theorem 1.1:

1. This is the first subclass of P which has been shown to contain MATCH (**Question 1**).

[2] Li, Pyne, and Tell [LPT24] give a *search* problem in CL which is not known to be in TC^1; however, unlike with MATCH, the corresponding decision problem is in TC^1 and thus is not applicable for studying the power of CL with regards to decision problems.

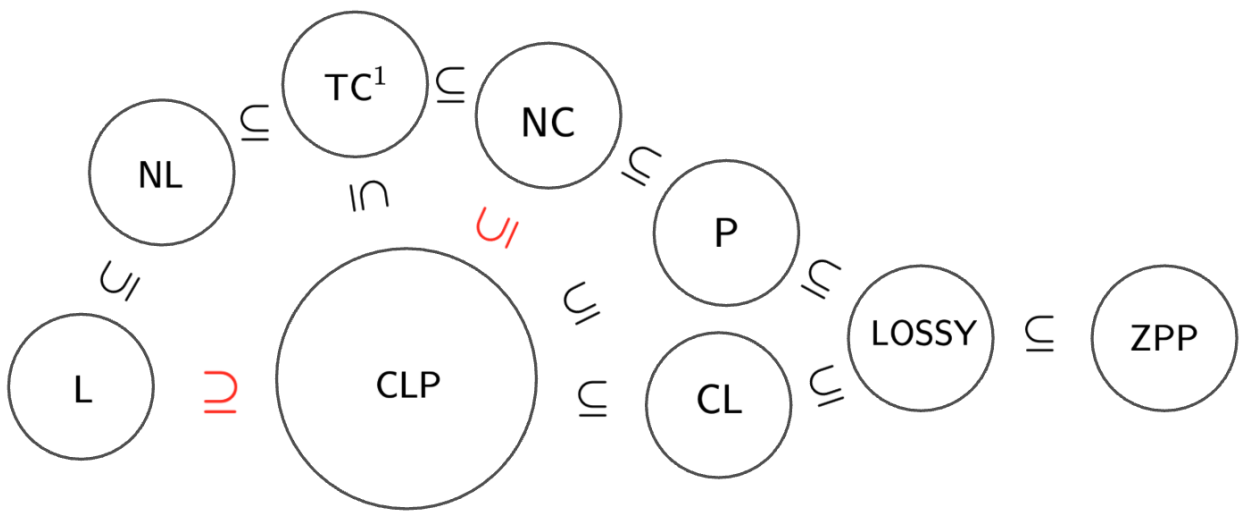

Figure 1: Barriers: black inclusions have been proven before this work, and red inclusions were conjectured in [BCKLS14; Kou16; Mer23]; our inclusion of MATCH in CLP acts as a barrier to all of those conjectures.

2. This is the first $\mathcal{B}$TIME-SPACE$(\text{poly}(n), n^{1-\Omega(1)})$ algorithm for MATCH for any additional resources $\mathcal{B}$ (**Question 3**). We believe that this gives hope for showing that MATCH can be computed in TIME-SPACE$(\text{poly}(n), n^{1-\Omega(1)})$, or perhaps, as was the case with Tree Evaluation, such catalytic algorithms are a reason to doubt our intuition that MATCH cannot be solved in $o(\log^2 n)$ space altogether.

3. This is the first problem outside TC^1 shown to be in CL, and thus the first strengthening of CL since the original work of Buhrman et al. [BCKLS14]; this also gives the strongest evidence thus far that $\mathsf{L} \neq \mathsf{CL}$ (**Question 4**).

4. This ties together **Question 2 & 5**, as showing CL or even CLP is contained in NC would immediately give the breakthrough result MATCH $\in$ NC. Conversely, if one believes that MATCH $\notin$ NC, then this gives evidence to contradict the conjectures of [BCKLS14; Kou16; Mer23] that $\mathsf{CL} \subseteq \mathsf{NC}$.

5. Many previous works have shown reductions from other graph problems to MATCH, and thus our CLP inclusion extends to these problems as well. We discuss the cases of *weighted matching*, *s-t max-flow*, and *global min-cut* in Section 7, although we stress this list is only a partial sample of such extensions of Theorem 1.1.

1.2 Derandomization

Lossy coding. One other important, and useful, aspect of catalytic computation is a connection to fundamental questions in derandomization, a connection exploited in recent line of work [DPT24; LPT24; Pyn24; DPTW25] for showing novel non-catalytic space-bounded algorithms. The lossy coding problem [Kor22] is defined as follows: given a pair of circuits $\mathsf{Comp} : \{0,1\}^n \to \{0,1\}^{n-1}$

and $\mathsf{Decomp} : \{0,1\}^{n-1} \to \{0,1\}^n$, output $x \in \{0,1\}^n$ such that $\mathsf{Decomp}(\mathsf{Comp}(x)) \neq x$. For a closed complexity class $\mathcal{C}$, $\mathsf{LOSSY}[\mathcal{C}]$ is the class of problems $\mathcal{C}$-reducible to lossy coding when Comp and Decomp are required to be computed and evaluated in the class $\mathcal{C}$. It is easy to see that

$$\mathcal{C} \subseteq \mathsf{LOSSY}[\mathcal{C}] \subseteq \mathsf{ZP}\text{-}\mathcal{C}$$

$\mathsf{LOSSY}[\mathsf{P}]$, which is often just referred to as LOSSY, has seen attention in recent years in the context of total function NP [Kor22], range avoidance [KP24], and meta-complexity [CLORS23], and was discussed at length in a recent survey by Korten [Kor25] on the topic. The problem of derandomizing LOSSY is seen as a stepping stone towards the major derandomization goal of $\mathsf{P} = \mathsf{ZPP}$.

Unfortunately, for most classes $\mathcal{C}$ it remains unclear whether $\mathsf{LOSSY}[\mathcal{C}]$ contains *any natural and well studied problem* outside $\mathcal{C}$. This was posed as an open problem in [Kor25].

Question 6: Does $\mathsf{LOSSY}[\mathcal{C}]$ admit a natural problem not known to be in $\mathcal{C}$ for any class $\mathcal{C}$?

The only known progress on this question comes via catalytic computation, as Pyne [Pyn24] showed that $\mathsf{BPL} \subseteq \mathsf{LOSSY}[\mathsf{L}]$. However, this remains unknown for all larger classes. Moreover, due to the fact that space-bounded randomized classes use read-once randomness, $\mathsf{LOSSY}[\mathsf{L}]$ is perhaps an overkill for derandomizing BPL.

Isolation lemma. One of the greatest contributions of the study of MATCH to complexity theory is a key tool, known as the *isolation lemma*, which is the backbone of all parallel algorithms for MATCH algorithms since its introduction by Mulmuley, Vazirani, and Vazirani [MVV87]. It has since turned out to be a very strong tool used in the design of randomized algorithms for a wide range of problems [OS93; NSV94; RA00; KS01; AM08; BTV09; LP15; KT16; GT17; VP19] [3]. Thus, derandomizing the isolation lemma, independent of a concrete problem, has become an important problem in itself [CRS93; AM08; AGT20; AV20; GTV21].

Question 7: Which deterministic classes $\mathcal{C}$ can implement the isolation lemma?

While the breakthrough $\mathsf{Quasi\text{-}NC}$ algorithm of Fenner, Gurjar, and Thierauf [FGT16] does exactly this, it both lies outside P and uses weights which are much larger than in all other applications. In fact this question is unresolved even for the "derandomization" class for NC discussed above, namely its LOSSY variant.

Question 8: Is the isolation lemma in $\mathsf{LOSSY}[\mathsf{NC}]$?

Our results (2). An analysis of our algorithm in Theorem 1.1 gives answers to all of the above questions:

Theorem 1.2.

$$\mathsf{MATCH} \in \mathsf{LOSSY}[\mathsf{NC}]$$

Again some discussion is in order:

1. To the best of our knowledge, this is the first well studied and natural problem shown to be in $\mathsf{LOSSY}[\mathcal{C}]$ and not known to be in $\mathcal{C}$ for any class $\mathcal{C}$ larger than L (**Question 6**).

[3] For a more comprehensive list of applications of the isolation lemma we direct the readers to [AGT20].

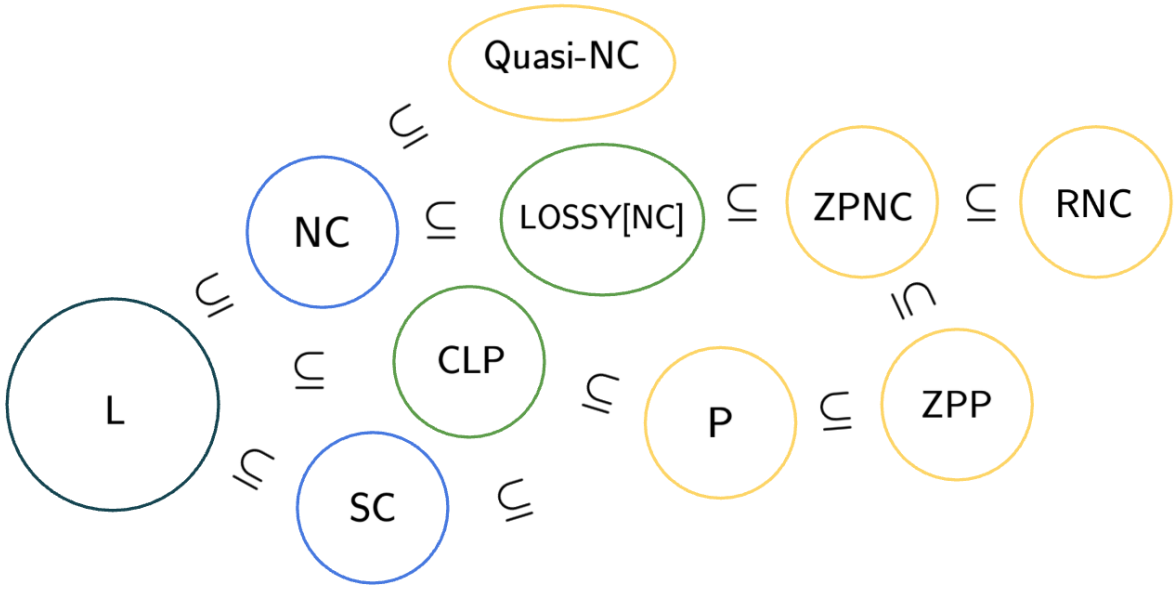

Figure 2: Contributions: yellow bubbles correspond to classes which were already known to contain MATCH, blue bubbles correspond to classes for which the MATCH inclusion is a long standing open problem, and green bubbles correspond to the inclusions shown in this paper.

2. Since NC $\subseteq$ LOSSY[NC] $\subseteq$ ZPNC $\subseteq$ RNC, this is a direct improvement over MATCH $\in$ RNC [Lov79; KUW85; MVV87] and MATCH $\in$ ZPNC [Kar86] (it is incomparable to all other NC-related results about MATCH). It also motivates the further study of lossy coding, since any derandomization of LOSSY[NC] would make progress towards the ultimate goal of MATCH $\in$ NC. The partial derandomization of BPL by Doron et al. [DPTW25] using LOSSY[L] gives hope that this approach may prove fruitful.

3. Our algorithm is built upon a catalytic derandomization of the isolation lemma, thus showing that it can be implemented in deterministic CLP (**Question 7**) as well as LOSSY[NC] (**Question 8**). We believe that the framework introduced in this work could be used to show similar CL and LOSSY results for the many other problems which admit isolation lemma based algorithms - similar to the exciting line of work which followed [FGT16] (see e.g. [KT16; GG17; GT17; ST17; AV19; VP19; AV20; GTV21]).

1.3 Open problems

We suggest a number of open problems coming from this work:

1. Can MATCH be computed using less catalytic space, say $O(\log^2 n)$, or alternatively can our algorithm be used as a subroutine for ordinary space or time/space-bounded computation?

2. Can CL prove related functions such as non-bipartite matching, matroid intersection, etc., or indeed can the inclusion $\mathsf{TC}^1 \subseteq \mathsf{CL}$ be improved to a stronger class, perhaps even NC^2?

Alternatively, can we use the CL derandomization of the isolation lemma to show novel derandomizations such as $\mathsf{RNC}^1 \subseteq \mathsf{CL}$?

3. Does $\mathsf{LOSSY[NC]}$ contain any other related functions such as non-bipartite matching or matroid intersection? Is $\mathsf{NC} = \mathsf{LOSSY[NC]}$?

1.4 Proof overview

We finish the introduction by giving an overview of our argument along with where in the paper each part will appear. Our algorithm will utilize the *isolation lemma* based framework of Mulmuley, Vazirani, and Vazirani [MVV87] for MATCH, combined with the *compress-or-random* framework introduced by Cook et al. [CLMP25], with novel insights for both.

Weighting and Isolating Matchings (Section 3). Let n be the number of vertices in the graph. Assume we have access to some fixed weight function W whose weights are at most $\text{poly}(n)$, and consider the set $\mathcal{M}$ of all perfect matchings M in G. We say that W is *isolating* on $\mathcal{M}$ if there exists a single matching $M_{\text{unique}} \in \mathcal{M}$ of minimum weight. [MVV87] prove that a random W is isolating on $\mathcal{M}$ with high probability.

By taking the Edmonds matrix N of G, where $N[i,j] = 2^{w(i,j)}$ for an isolating weight assignment W, it is additionally proven in [MVV87] that the unique min-weight perfect matching M_{unique} can be derived from the low-order bit of $\mathsf{DET}(N)$, and thus is in CLP given access to W. We describe modifications that can find a matching of any fixed size $k \in [n]$ as well, again provided that such a matching is the unique minimum weight matching of size k.

Maximum Matching via Isolation (Section 4). Let k be a value for which W isolates a matching of size k, and let M^k_{unique} be the matching in question. There are three possibilities: M^k_{unique} is the largest matching in G, there is a unique min-weight matching of size $k+1$ under W, or there are at least two min-weight matchings of size $k+1$ under W. To distinguish these cases, we will build a directed graph G^k based on G and M^k_{unique} which contains two additional nodes, s and t, such that, from every path P from s to t, we can extract a matching M_P of size $k+1$ along with its weight[4].

Thus if there are no s-t paths in G^k we can output k; here we use the fact that $\mathsf{NL} \subseteq \mathsf{CLP}$ to perform this search. Our main contribution in this section is on the structure of the graph G^k, which allows us to test whether a matching of size $k+1$ is isolated by W, and more importantly, handle the case where it is not isolated.

Compression of Weight Assignments (Section 5). The main observation in the current paper is in identifying and handling the remaining case, where there exist at least two min-weight matchings of size $k+1$, in CLP. Notice that for a random W this case will not occur with high probability due to [MVV87], but we have not yet specified where W comes from in our deterministic procedure. In fact, we will have W come from the (adversarial) catalytic tape itself, and we will

[4]For readers familiar with matching theory, G^k is exactly the residual graph of G with respect to M^k, and the path P is an augmenting path.

handle this case in the style of [CLMP25].

To recap, matchings of size $k+1$ can be found by checking for paths in G^k, and in particular we can find an edge e in G that is in *some* min-weight matchings of size $k+1$ *but not all* of them. Our key is to notice that the weight of e is in fact redundant; by constructing G^k, and from it M_1^{k+1} and M_2^{k+1} - which are min-weight matchings of size $k+1$ not containing and containing e respectively, we find that $w(e) = w(M_1^{k+1}) - w(M_2^{k+1} \setminus \{e\})$. Thus since W is written on the catalytic tape, we can erase $w(e)$ from the catalytic tape and free up $\Theta(\log n)$ bits. We will then iterate this procedure, using a new part of the catalytic tape as a replacement for the erased memory.

Final Algorithm (Section 6). To sum up, we begin by splitting the catalytic tape into two parts, one part for computation and another part for specifying a weight assignment W on the edges E of G, where each weight has some $\Theta(\log n)$ bits, plus $\text{poly}(n)$ reserve weights which we save for later.

Starting from $k = 1$ we certify that W isolates a min-weight matching of size k, which allows us to compute this matching M^k_{unique} in CLP. We then use M^k_{unique} to certify this for $k+1$, and if so then we proceed. If not, then either there are no matchings of size $k+1$, at which point we return k and halt, or we find a redundant weight in W.

In the latter case, we erase this section of the catalytic tape, replace it with a reserve weight that we had set aside, and start the algorithm over. Repeating this $\text{poly}(n)$ times, we either eventually isolate the largest min-weight matching in the graph or we free up enough space to compute the matching by brute force. At the end of the algorithm we recover the compressed section of the tape one by one to reset the catalytic tape.

2 Preliminaries

We use notation $\mathbb{Z}_{\leq c}$ to denote the non-negative integers of value at most c. The $\mathsf{determinant}$ function is denoted by DET.

2.1 Graphs, matching, and weights

We denote by $G = (V, E)$ a graph on vertices V and edges E. For $v \in V$, we define $E(v) := \{e \in E \mid e \text{ is incident upon } v\}$.

Definition 2.1 (Matching)**.** *Let $G = (V, E)$ be an undirected graph. A* $\mathsf{matching}$ *is a set $M \subseteq E$ of edges such that*

$$\forall v \in V,\ |E(v) \cap M| \leq 1$$

A vertex $v \in V$ is matched by M (and otherwise unmatched) if

$$|E(v) \cap M| = 1$$

The size *of the matching M is $|M|$, and we call a matching* $\mathsf{perfect}$ *if every vertex is matched.*

In this paper we study matching in bipartite graphs $G = (V = L \sqcup R, E)$, i.e. where all edges exclusively go between L and R. We focus on the case where $|L| = |R| = |V|/2$, and for convenience we define $n := |V|/2$ rather than the size or number of nodes of G.

Definition 2.2. *The* matching function*, denoted by* MATCH*, takes as input an $n \times n$ bipartite graph G and returns a matching M in G of maximum size.*

We will also work with different sorts of graphs for our algorithm and analysis. First, we will sometimes work with directed graphs, and will make it clear from context whether G is directed or not. We will also consider weighted graphs, i.e. graphs where we are given a weight assignment $W : E \to \mathbb{Z}$; for any $S \subseteq E$ we extend the definition of W and define $W(S) := \sum_{s \in S} W(s)$.

Definition 2.3 (Symmetric Difference)**.** *Let $G = (V, E)$ be a graph. The* symmetric difference *of matchings $M_1, M_2 \subseteq E$, denoted by $M_1 \Delta M_2$, is defined as having edges $(M_1 \setminus M_2) \cup (M_2 \setminus M_1)$. Similarly for any matching M and set $S \subseteq E$, we define $M \oplus S$ as $(M \cup S) \setminus (M \cap S)$.*

Note that $M \oplus S$ need not be a matching, depending on S.

2.2 Catalytic computation

Our main computational model in this paper is the catalytic space model:

Definition 2.4 (Catalytic machines)**.** *Let $s := s(n)$ and $c := c(n)$. A* catalytic Turing machine *with space s and* catalytic space *c is a Turing machine M with a read-only input tape of length n, a write-only output tape, a read-write work tape of length s, and a second read-write work tape of length c called the* catalytic tape*, which will be initialized to an adversarial string τ.*

We say that M computes a function f if for every $x \in \{0,1\}^n$ and $\tau \in \{0,1\}^c$, the result of executing M on input x with initial catalytic tape τ fulfils two properties: 1) M halts with $f(x)$ written on the output tape; and 2) M halts with the catalytic tape in state τ.

Such machines naturally give rise to complexity classes of interest:

Definition 2.5 (Catalytic classes)**.** *We define* $\mathsf{CSPACE}[s, c]$ *to be the family of functions computable by catalytic Turing machines with space s and catalytic space c. We also define* catalytic logspace *as*

$$\mathsf{CL} := \bigcup_{d \in \mathbb{N}} \mathsf{CSPACE}[d \log n, n^d]$$

Furthermore we define CLP *as the family of functions computable by* CL *machines that are additionally restricted to run in polynomial time for every initial catalytic tape τ.*

Important to this work will be the fact, due to Buhrman et al. [BCKLS14], that CLP can simulate log-depth threshold circuits:

Theorem 2.6 ([BCKLS14])**.**

$$\mathsf{TC}^1 \subseteq \mathsf{CLP}$$

This gives a number of problems, such as determinant and s-t connectivity, in CLP. We will mention a few of these directly, as they will be necessary later. First, determinant over matrices with polynomially many bits is in GapL (see c.f. [MV97]) and thus in CLP:

Lemma 2.7. *Let N be an $n \times n$ matrix over a field of size $\exp(n)$. Then $\mathsf{DET}(N)$ is computable in* CLP*.*

Second, we extend the inclusion of NL to show that that *weighted* connectivity is also in CLP:

Lemma 2.8. *Let $G = (V, E)$ be a directed graph, let $s, t \in V$ be a given source and target vertex, and $W : E \to \mathbb{Z}_{\leq \mathrm{poly}(n)}$ be edge weights such that G does not have any non-positive weight cycles under W. Then there exists a CLP machine which, given (G, s, t, W), computes the minimum weight of a simple s-t path.*

Proof. The problem of deciding whether G contains an s-t walk of length $\leq |E|$ and weight $\leq \alpha$, for $\alpha \leq \mathrm{poly}(n)$, is trivially decidable in NL and thus in CLP. Binary searching over the value of α gives the minimum α^* such that G contains an s-t walk of length $\leq |E|$ and weight $\leq \alpha^*$. Since all cycles in G have strictly positive weight, the minimising walk is also guaranteed to be a simple s-t path. □

2.3 Other complexity classes

Besides catalytic computation, we also work with a few other classes. First we recall the classic NC definition of parallel complexity:

Definition 2.9 (NC)**.** *A language $\mathcal{L}$ is computable in (uniform) NC if there exists a (uniform) circuit family $\{C_n\}_{n\in\mathbb{N}}$ such that C_n has size $\mathrm{poly}(n)$, depth $\log^{O(1)} n$, and decides membership in $\mathcal{L}$ on all inputs of size n.*

We also define the class of problems reducible to the $\mathsf{lossy\ coding}$ problem over various objects:

Definition 2.10 (Lossy coding and $\mathsf{LOSSY}[\mathcal{C}]$)**.** *The $\mathsf{lossy\ coding\ problem}$ is defined as follows: given a pair of algorithms $\mathsf{Comp} : \{0,1\}^n \to \{0,1\}^{n-1}$ and $\mathsf{Decomp} : \{0,1\}^{n-1} \to \{0,1\}^n$ as input, our goal is to output any $x \in \{0,1\}^n$ such that $\mathsf{Decomp}(\mathsf{Comp}(x)) \neq x$.*

Let $\mathcal{C}$ be a complexity class. We define $\mathsf{LOSSY}[\mathcal{C}]$ as the set of languages reducible to the lossy coding problem whose input algorithms come from the class $\mathcal{C}$. When no class is given, we define $\mathsf{LOSSY} := \mathsf{LOSSY}[\mathsf{P}]$.

Of note, the above subroutines (weighted s-t connectivity and DET) are also in NC, which will be useful for Theorem 1.2.

3 Weighting and Isolating Matchings

Our first task is to set up the basic structure of our algorithm, which is to find matchings via *isolation*.

Definition 3.1. *Let U be a set and $W : U \to \mathbb{Z}$ be a weight assignment, and let $\mathcal{F} \subseteq 2^U$ be a family of subsets of U. We say that W is $\mathsf{isolating}$ for $\mathcal{F}$ if*

$$\exists!\ S_{unique} \in \mathcal{F} \text{ such that } W(S_{unique}) = \min_{F\in\mathcal{F}} W(F)$$

where $\exists!$ means that exactly one such set exists.

Mulmuley, Vazirani, and Vazirani [MVV87] showed that, given the ability to compute the determinant, isolation is sufficient to solve perfect matching.

Theorem 3.2 ([MVV87])**.** *Let $G = (V = L \cup R, E)$ be a bipartite graph, and let $W : E \to \mathbb{Z}_{\leq \mathrm{poly}(n)}$ be a weight assignment that isolates a perfect matching M_{unique} in G. Then there exists an $\mathsf{L}^{\mathsf{DET}}$ machine which, given input $(G, e \in E)$ and access to W, determines whether $e \in M_{unique}$.*

Thus Lemma 2.7 gives us a CLP algorithm for finding the isolated perfect matching of min-weight, as Theorem 3.2 computes DET on an $n \times n$ matrix whose entries have $\mathrm{poly}(n)$ bits.

We will also need to extend Theorem 3.2 to work for matchings of any fixed size $k \in [n]$. We sum this up with the observations above into our main algorithm for this section.

Lemma 3.3. *Let $G = (V = L \cup R, E)$ be a bipartite graph, and let $k \in \mathbb{N}$ be a parameter whose value is at most the size of the maximum matching of G. Let $W : E \to \mathbb{Z}_{\leq \mathrm{poly}(n)}$ be edge weights such that W isolates a size k matching M^k_{unique} in G. Then there exists a CLP machine which, given input $(G, e \in E)$ and access to W, determines whether $e \in M^k_{unique}$.*

Proof. Given G, we construct the following graph $G' = (V', E')$ and weight function W':

1. $V' = L' \cup R'$, where L' consists of L and $n - k$ new vertices and R' is constructed similarly.
2. E' consists of E along with a clique between the new vertices on each side with the old vertices on the other side. That is, $E' = E \cup \{(u, v) \mid u \in L, v \in R' \setminus R\} \cup \{(u, v) \mid u \in R, v \in L' \setminus L\}$
3. Index all vertices in G' by $[2(n+n-k)]$ arbitrarily. For $e \in E$, we define $W'(e) = W(e) \cdot 10 \cdot n^4$, and $e = (u, v) \in E' \setminus E$ we define $W'(e) = u \cdot v$ to be the the product of the indices of the vertices.

By construction, every perfect matching $M' \subseteq E'$ in G' corresponds to a size k matching $M \subseteq E$ in G. For any size k matching M of G and perfect matching M' of G', M' is defined to be an extension of M if $M' \cap E = M$. Note that for any perfect matchings M_1 and M_2 of G', if $W(M_1 \cap E) < W(M_2 \cap E)$, then

$$\begin{aligned}
W'(M_2) - W'(M_1) &= (W'(M_2 \cap E) + W'(M_2 \setminus E)) \\
&\quad - (W'(M_1 \cap E) + W'(M_1 \setminus E)) \\
&= (W'(M_2 \cap E) - W'(M_1 \cap E)) \\
&\quad + (W'(M_2 \setminus E) - W'(M_1 \setminus E)) \\
&\geq 10n^4(W(M_2 \cap E) - W(M_1 \cap E)) - W'(M_1 \setminus E) \\
&\geq 10n^4 - 10n^3 > 0
\end{aligned}$$

and thus $W'(M_1) < W'(M_2)$. Therefore the minimum weight perfect matching M' of G' must be an extension of M^k_{unique}.

We say an extension is a minimum weight extension if $W'(M')$ is minimum amongst all extensions of M. We claim that the minimum weight extension of any size k matching M^k of G is unique.

Let $L_{\text{unmatched}}$ be vertices in L and $R_{\text{unmatched}}$ be the vertices in R which are not matched by M^k, and let L_{new} and R_{new} be the vertices in $L' \setminus L$ and $R' \setminus R$ respectively. Any extension of M^k consists of a perfect matching in $(L_{\text{new}} \cup R_{\text{unmatched}}, E' \setminus E)$ and in $(L_{\text{unmatched}} \cup R_{\text{new}}, E' \setminus E)$,

both of which are disjoint bipartite cliques whose weight function W' is given by $w(u,v) = u \cdot v$.

We claim that both of these cliques have a unique perfect matching, and thus the min-weight extension of M^k is unique:

Claim 3.4. *Let $G = (V = L \cup R, E)$ be an $s \times s$ bipartite clique, and let $W : E \to \mathbb{Z}$ be weights defined as $W(e = (u,v)) = u \cdot v$ for an arbitrary indexing of V. Then W isolates a perfect matching in G.*

Proof. Let $L = \{l_1, \ldots, l_s\}$ and $R = \{r_1, \ldots, r_s\}$ be ordered by increasing indices. We claim that $\{(l_i, r_{n+1-i})\}_{i \in [n]}$ is the unique minimum weight perfect matching in G. For any other perfect matching M, there must exist $u_1 < u_2 \in L$ and $v_1 < v_2 \in R$ (again, ordered by indices) such that $(u_1, v_1), (u_2, v_2) \subseteq M$. Then for $M' = (M \setminus \{(u_1, v_1), (u_2, v_2)\}) \cup \{(u_1, v_2), (u_2, v_1)\}$, it is easy to verify that $W(M') = W(M) - (u_2 - u_1)(v_2 - v_1)$, and thus $W(M') < W(M)$, which shows that M is not a minimum weight perfect matching. □

Thus W isolates a perfect matching M^* in G', namely the unique minimum weight extension of the minimum weight perfect matching M^k_{unique} of size k. Applying Theorem 3.2 and Lemma 2.7 shows that extracting M^*, and from it M^k_{unique}, can be done in CLP. □

4 Maximum Matching via Isolation

In the previous section we showed how to extract a matching of size k in G; however, this requires us to know that W isolates such a matching for this value of k. In this section we show how to find the maximum k for which this holds.[5]

Our algorithm will do this for k inductively. This is easy to do for $k = 1$, as there are two min-weight matchings iff there are two min-weight edges, and there are no matchings iff there are no edges. Thus for the remainder of this section, assume that W isolates a unique min-weight matching M^k_{unique} of size k, computable in CLP by Lemma 3.3, which we will use to test $k + 1$.

4.1 Residual Graphs and Augmenting Paths

The important tool in our construction will be the *residual graph* of our matching M^k_{unique}. This construction is a standard tool in graph algorithms, and can be found, along with the other facts we state, in the monograph [KR98] or textbook [CLRS22]. For readers familiar with matching theory, this is exactly the residual graph one would obtain from the max flow instance corresponding to maximum matching.

Definition 4.1 (Residual Graph)**.** *Let $G = (V = L \cup R, E)$ be a bipartite graph, M be a matching in G, and $W : E \to \mathbb{Z}$ be edge weights. The* residual graph *of M, which we denote by $G^M_{residual} = (V^M_{residual}, E^M_{residual})$, is a directed graph whose vertex set $V^M_{residual}$ consists of V plus a new source s and a new sink t. The edge set $E^M_{residual}$ consists of the following edges where $u \in L$ and $v \in R$:*

[5] An earlier result of Hoang, Mahajan, and Thierauf [HMT06] shows how to determine whether edge weights isolate a perfect matching, assuming that one exists; however, our algorithm needs to additionally distinguish between the case where a size k matching simply does not exist, and the case where a size k matching exists but is not isolated.

1. *if u is unmatched by M we add an edge $(s \to u)$, and if v is unmatched by M we add an edge $(v \to t)$*
2. *if $(u, v) \in M$ we add an edge $(v \to u)$*
3. *if $(u, v) \in E \setminus M$ we add an edge $(u \to v)$*

We will drop the M superscript in cases where the usage is clear.

Because M^k_{unique} can be constructed in CLP, the graph G_{residual} can be constructed in CLP as well. As we will soon see, paths from s to t in G^M_{residual} (ignoring the first and last edges) are closely related to matchings in G.

Definition 4.2 (Augmenting Path)**.** *Let $G = (V, E)$ be a graph, M be a matching in G, and $W : E \to \mathbb{Z}$ be a weight assignment. A path $P = \{e_1, \ldots, e_m\}$ on vertices $\{v_1, \ldots, v_{m+1}\}$ is an* augmenting path *with respect to M if:*

1. *v_1 and v_{m+1} are not matched by M.*
2. *For all odd $i \in [m]$, $e_i \notin M$.*
3. *For all even $i \in [m]$, $e_i \in M$.*

We will be using a slightly different weight scheme for residual graphs and augmenting paths, which will be convenient for talking about matchings in their context.

Definition 4.3. *Let $G = (V = L \cup R, E)$ be a bipartite graph, M be a matching in G, and $W : E \to \mathbb{Z}$ be a weight assignment. We define the* alternating *weight function as*

$$W^M_{alt}(e) := \begin{cases} -W(e), & \text{if } e \in M \\ W(e), & \text{if } e \notin M \end{cases}$$

Furthermore, for residual graph $G^M_{residual}$ we define the residual *weight function as*

$$W^M_{residual}(e) := \begin{cases} 0, & \text{if } e \notin E \\ W^M_{alt}(e), & \text{otherwise} \end{cases}$$

Whenever we use weights in $G^M_{residual}$, we are referring to the weights $W^M_{residual}$, while for augmenting paths P with respect to M we use W^M_{alt}.

It is straightforward to observe that any augmenting path P in G can be extended to a path in the residual graph G^M_{residual} via two additional edges and vice versa, and furthermore both paths have the same weight; we will analyze this fact in more detail later.

Important to our algorithm is that augmenting paths to size k matchings in G are, in a sense, equivalent to matchings of size $k + 1$.

Fact 4.4 (Berge's Theorem [Ber57])**.** *Let $G = (V, E)$ be a graph, M be a matching in G, and $W : E \to \mathbb{Z}$ be a weight assignment. Then M is maximum iff there do not exist any augmenting paths with respect to M. Furthermore, for any augmenting path P, $M \oplus P$ is a matching in G of size $|M| + 1$.*

An immediate consequence of Fact 4.4 is that we can test if there are no matchings of size $k+1$.

Lemma 4.5. *Let $G = (V = L \cup R, E)$ be a bipartite graph and $W : E \to \mathbb{Z}$ be a weight assignment. Let $k \in [n]$ such that W isolates a size k matching M^k_{unique}. Then there exists a* CLP *machine which, given (G, W, k), decides whether M^k_{unique} is a maximum matching.*

Proof. By Lemma 3.3 we can compute M^k_{unique}, and from it we can build the residual graph G_{residual}. By Fact 4.4, M^k_{unique} is a maximum matching iff t is not reachable from s in G_{residual}, which we can check in CLP. □

A sequential algorithm could of course repeatedly apply Lemma 4.5 to search for larger matchings, as proposed in [Kuh55]. However, this requires large space to store the matching after a certain number of iterations. Instead, we will utilise a hybrid approach between the isolation lemma based approach of [MVV87] and the augmenting paths based approach of [Kuh55].

4.2 Matching Weights and Augmenting Paths

For the rest of this section we assume that at least one augmenting path exists, and thus there is some matching of size $k+1$. Our goal is to use G_{residual} to understand all matchings of size $k+1$, and in particular the min-weight matchings.

Lemma 4.6. *Let $G = (V = L \cup R, E)$ be a bipartite graph and $W : E \to \mathbb{Z}$ be edge weights such that W isolates a size k matching M^k_{unique} in G. Let M^{k+1} be any minimum weight size $k+1$ matching in G. Then the symmetric difference $M^k_{unique} \Delta M^{k+1}$ is a single augmenting path with respect to M^k_{unique}.*

Proof. Define $H := M^k_{\text{unique}} \Delta M^{k+1}$, and assume for contradiction that H is not a single augmenting path. We will go through the different potential cases for H, showing that either M^k_{unique} or M^{k+1} can be modified to contradict either their minimality or, in the case of M^k_{unique}, its uniqueness.

The proof of the following claim is in the spirit of [DKR10], which introduced the relationship between alternating weights and isolated matchings.

Claim 4.7. *There does not exist any set $\emptyset \subsetneq S \subseteq H$ satisfying the following properties:*

1. *$M^k_{unique} \oplus S$ is a matching in G of size k.*
2. *$M^{k+1} \oplus S$ is a matching in G of size $k+1$.*

Proof. Note that, since every $e \in S$ belongs to either M^k_{unique} or M^{k+1} and not the other, we have

$$W_{\text{alt}}^{M^k_{\text{unique}}}(S) = \sum_{e \in S \cap M^k_{\text{unique}}} -W(e) + \sum_{e \in S \cap M^{k+1}} W(e) = -W_{\text{alt}}^{M^{k+1}}(S)$$

There are two cases:

1. $W_{\text{alt}}^{M^k_{\text{unique}}}(S) \leq 0$: In this case, $M^k_{\text{unique}} \oplus S$ is a distinct matching of size k with weight $W(M^k_{\text{unique}}) + W_{\text{alt}}^{M^k_{\text{unique}}}(S) \leq W(M^k_{\text{unique}})$, which contradicts the fact that M^k_{unique} is the *unique* minimum weight matching of size k.

2. $W_{\text{alt}}^{M^k_{\text{unique}}}(S) > 0$: In this case, $M^{k+1} \oplus S$ is a matching of size $k+1$ with weight $W(M^{k+1}) + W_{\text{alt}}^{M^{k+1}}(S) = W(M^{k+1}) - W_{\text{alt}}^{M^k_{\text{unique}}}(S) < W(M^{k+1})$, which contradicts the minimality of M^{k+1}. □

We now show that such an S must exist in any H which does not consist of a single augmenting path. Clearly any path in H alternates between edges of M^k_{unique} and M^{k+1}, and thus the connected components of H are of the following four types:

1. even length alternating cycles.
2. even length alternating paths.
3. augmenting paths with respect to M^k_{unique}.
4. augmenting paths with respect to M^{k+1}.

The former two cases are immediate; any even length cycle or path has an equal number of edges in both M^k_{unique} and M^{k+1}, and so taking S to be this contradicts Claim 4.7. Thus we can assume all connected components of H are augmenting paths with respect to M^k_{unique} or M^{k+1}.

Let P_k be the set of augmenting paths in H with respect to M^k_{unique}, and let P_{k+1} be the set of augmenting paths with respect to M^{k+1}. Since $|M^{k+1}| = |M^k_{\text{unique}}| + 1$, we have $|P_{k+1}| = |P_k| + 1$, and by assumption we do not have $|P_{k+1}| = 1$ and $|P_k| = 0$; thus $P_k \neq \emptyset$. Define $S = P \cup P'$ for any any $P \in P_k$ and $P' \in P_{k+1}$, which gives us a contradiction by applying Claim 4.7. □

Because paths in G_{residual} correspond to augmenting paths of equal weight, Lemma 4.6 implies that isolating $k+1$-size matchings is equivalent to isolating paths in G_{residual}.

Lemma 4.8. *Let $G = (V = L \cup R, E)$ be a bipartite graph and $W : E \to \mathbb{Z}$ be edge weights such that W isolates a size k matching M^k_{unique} in G. Then W isolates a size $k+1$ matching in G iff $W_{residual}$ isolates an s-t path in $G_{residual}$.*

Proof. We show that both are equivalent to showing W_{alt} isolates an augmenting path with respect to M^k_{unique}. Let M^{k+1} be a minimum weight size $k+1$ matching in G. By Lemma 4.6, $M^{k+1} \Delta M^k_{\text{unique}}$ is an augmenting path with respect to M^k_{unique} of weight

$$W(M^{k+1}) = W(M^k_{\text{unique}}) + W_{\text{alt}}(M^{k+1} \Delta M^k_{\text{unique}})$$

Moreover, for any augmenting path P with respect to M^k_{unique},

$$W(M^k_{\text{unique}} \oplus P) = W(M^k_{\text{unique}}) + W_{\text{alt}}(P)$$

By the minimality of M^{k+1},

$$W_{\text{alt}}(M^{k+1} \Delta M^k_{\text{unique}}) \leq W_{\text{alt}}(P)$$

for every augmenting path P with respect to M^k_{unique}. Thus, $M^{k+1} \Delta M^k_{\text{unique}}$ is a minimum weight augmenting path, and for any minimum weight augmenting path P, $M^k_{\text{unique}} \oplus P$ is a minimum weight size $k+1$ matching.

If W does not isolate a size $k+1$ matching, there exist at least two minimum weight matchings of size $k+1$, and we let M_1^{k+1} and M_2^{k+1} be any two such matchings. Since they are distinct matchings, we conclude that $M_1^{k+1} \Delta M_{\text{unique}}^k$ and $M_2^{k+1} \Delta M_{\text{unique}}^k$ are distinct minimum weight augmenting paths with respect to M_{unique}^k. Conversely, assume that there exist distinct minimum weight augmenting paths P_1 and P_2. Then $M_{\text{unique}}^k \oplus P_1$ and $M_{\text{unique}}^k \oplus P_2$ are distinct minimum weight size $k+1$ matchings.

Second, we show that W_{alt} isolates an augmenting path with respect to M_{unique}^k iff W isolates an s-t path in G_{residual}. As observed before, every augmenting path with respect to M_{unique}^k gives a simple s-t path in G_{residual} and vice versa. It is thus sufficient to show that the shortest s-t path in G_{residual} is always simple, i.e. G_{residual} does not contain any cycle C such that $W_{\text{residual}}(C) \leq 0$ with respect to W. Assuming otherwise, since C contains an equal number of edges in and not in M_{unique}^k, taking $M_{\text{unique}}^k \oplus C$ gives us a new matching of size k and weight

$$W(M_{\text{unique}}^k) + W_{\text{residual}}(C) = W(M_{\text{unique}}^k) + W_{\text{alt}}(C) \leq W(M_{\text{unique}}^k)$$

which contradicts the unique minimality of M_{unique}^k. □

5 Compression of Weight Assignments

In this section, we combine our previous algorithms with a new step for CLP, namely compressing the weight function in the case when it does not isolate a matching of some size k. This will allow us to use our catalytic tape itself as providing a weight function, since it will either a) be random enough to successfully run the algorithm as outlined above, or b) we will be able to compress enough space to compute MATCH in P.

5.1 Recursive Step: Termination, Isolation, or Failure

We begin by identifying the information that will be needed in the case when W does not isolate a matching of size $k+1$.

Definition 5.1 (Threshold Edge)**.** *Let $G = (V, E)$ be a directed graph, $s \in V$ be a source vertex, $t \in V$ be a target vertex, and $W : E \to \mathbb{Z}$ be edge weights such that G does not have any non-positive weight cycles under W. We say that an edge $e \in E$ is a* threshold edge *if there exist two minimum weight s-t paths P_1, P_2 in G such that $e \in P_1$ and $e \notin P_2$.*

By definition it is clear that a threshold edge exists iff W does not isolate an s-t path in G. We also observe that by computing reachability in CLP, we can find such an edge:

Lemma 5.2. *Let $G = (V, E)$ be a directed graph, $s \in V$ be a source vertex, $t \in V$ be a sink vertex, and $W : E \to \mathbb{Z}_{\leq \text{poly}(n)}$ be edge weights such that G does not have any non-positive weight cycles under W. Furthermore, assume that there exist at least two distinct min-weight s-t paths under W. Then there exists a* CLP *machine which, given (G, s, t, W), outputs a threshold edge.*

Proof. For any two vertices $u, v \in G$, let $\delta_{u,v}$ be the minimum weight of any u-v path. For each edge $e = u \to v \in E$, e is a threshold edge iff 1) $\delta_{s,u} + W(e) + \delta_{v,t} = \delta_{s,t}$, i.e. e lies on at least one min-weight s-t path; and 2) $\delta'_{s,t} = \delta_{s,t}$, where $\delta'_{s,t}$ is the min-weight s-t path in $G \setminus \{e\}$, i.e. there exists some min-weight s-t path which does not use e. All these tests can compute in CLP using Lemma 2.8, and so we loop over all e in order until we find a threshold edge. □

This finally brings us to our recursive procedure, which allows us to go from isolating matchings of size k to matchings of size $k+1$.

Lemma 5.3. *Let $G = (V = L \cup R, E)$ be a bipartite graph, let $W : E \to \mathbb{Z}$ be a weight assignment, and let $k \in [n]$ such that W isolates a size k matching M^k_{unique}. Then there exists a* CLP *machine A which, on input (G, k, W), performs the following:*

1. *if no matching of size $k+1$ exists, A outputs $\perp$*
2. *if W isolates a matching of size $k+1$, A outputs 1*
3. *if W does not isolate a matching of size $k+1$, A outputs a threshold edge e in $G_{residual}$ with the further promise that $e \notin M^k_{unique}$*

Proof. In CLP we can compute M^k_{unique} by Lemma 3.3, and thus we can construct the graph G_{residual}. By Lemma 4.5, if no matching of size $k+1$ exists, we can determine this in CLP. Otherwise, we run the algorithm of Lemma 5.2 to see if any threshold edge exists, and if not then W must isolate a matching of size $k+1$ by Lemma 4.8.

Finally, if any threshold edge exists, then there must exist one outside M^k_{unique}. If not, then every min-weight path contains the same set of edges outside M^k_{unique}, and since each vertex is only adjacent to at most one edge in M^k_{unique} every min-weight path must also contain the same set of edges within M^k_{unique}. Thus every min-weight matching of size $k+1$ is the same, which is a contradiction. □

5.2 Compressing Isolating Edges

Since it is clear how to proceed in the first two cases of Lemma 5.3, we now turn to the third case, when we obtain a threshold edge. The key observation is that $W(e)$ can be determined via the rest of the weight function.

Lemma 5.4. *Let $G = (V, E)$ be a directed graph, $s \in V$ be a source vertex and $t \in V$ be a target vertex, and $W : E \to \mathbb{Z}_{\leq \text{poly}(n)}$ be edge weights such that all cycles in G have strictly positive weight under W. Let $e = u \to v$ be a threshold edge in this graph. Then there exists a* CLP *machine which, given $(G, e, W \restriction_{E \setminus \{e\}})$, computes $W(e)$.*

Proof. This algorithm is essentially the same as Lemma 5.2. To recap, we again define, for any two vertices $u, v \in G$, the value $\delta_{u,v}$ be the minimum weight of any u-v path. Since $e = u \to v \in E$ is a threshold edge, it follows that $\delta_{s,t} = \delta_{s,u} + W(e) + \delta_{v,t}$, and furthermore that $\delta_{s,t} = \delta'_{s,t}$ where $\delta'_{s,t}$ is the min-weight s-t path in $G \setminus \{e\}$. Putting these two facts together we get that

$$W(e) = \delta'_{s,t} - (\delta_{s,u} + \delta_{v,t})$$

and we can calculate all three quantities in CLP using Lemma 2.8. Furthermore, none of these quantities involve $W(e)$; this is true for $\delta'_{s,t}$ by definition, while the other two hold because all cycles have positive weight and thus every min-weight path from s to u (or v to t) does not involve $u \to v$. □

This gives rise to our compression and decompression subroutines, which allow us to erase and later recover $W(e)$ from the catalytic tape.

Lemma 5.5. *Let $G = (V = L \cup R, E)$ be a bipartite graph, and let τ be a string of length* $\mathrm{poly}(n)$ *such that we interpret the initial substring of τ as a weight assignment $W : E \to \mathbb{Z}_{\leq n^5}$ which isolates a matching of size k but does not isolate a matching of size $k+1$. We interpret an additional $5 \log n$ bits on the catalytic tape as a reserve weight $r \in \mathbb{Z}_{\leq n^5}$.*

Then there exist catalytic subroutines $\mathsf{Comp}, \mathsf{Decomp}$ *with the following behavior:*

- $\mathsf{Comp}(G, k)$ *replaces (W, r) with $(W', k, e, 0^{2 \log n})$ for some edge e, where $W'(e') = W(e')$ for all $e' \neq e$ and $W'(e) = r$, and leaves all other catalytic memory unchanged*
- $\mathsf{Decomp}(G, k, u, v)$ *inverts* Comp

Proof. By Lemma 5.3, Comp can use the rest of its catalytic tape to find a threshold edge $e = u \to v$ which is outside of G^k_{unique}, given that such an edge must exist by assumption. Now Comp will erase $W(e)$ from the catalytic tape and replace it by r; we then erase the original copy of r and record k and the indices u, v of e in it. These indices take $\log n$ bits each and $k \leq n$ requires $\log n$ bits, while $W(e)$ takes $5 \log n$ bits on the catalytic tape, which gives us $2 \log n$ free bits as a result of this procedure.

For Decomp, since $e \notin M^k_{\text{unique}}$, we can determine M^k_{unique} by Lemma 3.3 given $(G \setminus \{e\}, k, W \setminus \{W(e)\})$, and from this we can construct G_{residual}; thus we can apply Lemma 5.4 to recover the value $W(e)$. We then erase k and e, move the weight in W' at location e to this memory, and replace it with the recovered value of $W(e)$. □

6 Final Algorithm

We finally collect all cases together to solve bipartite maximum matching in CLP and in $\mathsf{LOSSY[NC]}$.

6.1 Proof of Theorem 1.1

First we show the case of CLP, where our core algorithm will be spelled out in detail.

Theorem 6.1. *There exists a* CLP *algorithm which, given bipartite graph G as input, outputs the maximum matching in G.*

Proof. By [HK73], MATCH can be solved in time $T := O(|E|\sqrt{|V|})$. Our CLP algorithm will have three sections of catalytic tape:

1. a weight assignment $W : E \to \mathbb{Z}_{\leq n^5}$
2. a set of reserve weights $r_1 \ldots r_{T/(2 \log n)}$, each of size $5 \log n$
3. a large enough $\mathrm{poly}(n)$ catalytic space to run all CLP subroutines as needed

We set two loop counters c and k, both initialized to 0; c will record how many times we have compressed a weight value, while k will record the largest isolated min-weight matching found by W in the current iteration. Our basic loop for the current c and k will be to apply Lemma 5.3 for the current (G, k, W) and perform as follows:

1. if it returns $\perp$, record M^k_{unique} on the output tape and move to the decompression procedure (see below).
2. if it returns 1, increment k and repeat.
3. if it returns an edge e, increment c, apply the Comp subroutine of Lemma 5.5 using reserve weight r_c as r, and restart our algorithm for $k = 0$ with our new weight function W'

If our algorithm ever reaches the first case, we have successfully computed the maximum matching in G. This occurs unless we reach $c = T/(2\log n)$, and in this case we are left with T free bits on our catalytic tape, as each application of Comp frees $2\log n$ bits. We then apply our time T algorithm to solve MATCH directly. We record our answer on the output tape and move to the decompression procedure.

To decompress the tape, we apply the Decomp procedure of Lemma 5.5 in reverse order, starting from our final value of c and decrementing until we reach 0. By the correctness of Decomp each iteration will reset the catalytic tape to its state just before the cth run of the algorithm, meaning that our final state is the original catalytic tape τ, at which point we return the answer saved on our work tape.

We briefly analyze our resource usage. Our catalytic tape has length

$$\left(\binom{n}{2} + T/(2\log n)\right) \cdot (5\log n) + \text{poly}(n) = \text{poly}(n)$$

Our work tape will need to store loop variables $k \leq n$ and $c \leq T/(2\log n)$, plus free space to run our CLP subroutines, which is $O(\log n)$ in total.

All subroutines are CLP machines and thus run in polynomial time, while our loops for k and c run in time n and $T/(2\log n)$ respectively, and the decompression procedure again only uses CLP subroutines and runs for c steps. Finally if we reach the maximum value of c, we ultimately run the time T algorithm, which again take polynomial time. Thus our whole machine runs in polynomial time, logarithmic free space, and polynomial catalytic space, which is altogether a CLP algorithm. □

Remark 6.2. *Putting aside our runtime analysis and care with regards to using* CLP *rather than* CL *subroutines, it is also known due to Cook et al. [CLMP25]—in fact, by the same argument structure that we use here—that any problem in* $\mathsf{CL} \cap \mathsf{P}$ *can be solved in poly-time bounded* CL *generically.*

6.2 Proof of Theorem 1.2

We now prove our second theorem using the above algorithm; since most of the details are analogous we opt to be somewhat succinct in our proof.

Theorem 6.3. *There exist* NC *algorithms* $\mathcal{A}_1$, $\mathcal{A}_2$ *which, given a bipartite graph* $G = (V, E)$ *with as input, have the following behaviour:*

1. *$\mathcal{A}_1$ outputs* NC *circuits* Comp $: \{0,1\}^{f(n)} \to \{0,1\}^{f(n)-1}$ *and* Decomp $: \{0,1\}^{f(n)-1} \to \{0,1\}^{f(n)}$ *such that both* Comp *and* Decomp *have depth* $\text{polylog}(n)$ *and size* $\text{poly}(n)$*. Furthermore,* $f(n) \in \text{poly}(n)$*.*

2. *Given any* $x \in \{0,1\}^{f(n)}$ *such that* $\mathsf{Decomp}(\mathsf{Comp}(x)) \neq x$*,* $\mathcal{A}_2$ *outputs a maximum matching of* G*.*

Proof. Our proof follows from Theorem 6.1, and in particular Lemma 5.5, under a different setup. In particular, rather than using CLP subroutines and a weight function from the catalytic tape, we will use NC subroutines and a weight function as given by the input. Our outer c loop is unnecessary as we only need to compress once, while the inner k loop can be parallelized to keep our algorithm in low depth.

We now move to the details of how to construct our algorithms and circuits. We first describe the NC algorithm which corresponds to Comp:

1. We interpret the input of circuit C as the graph G along with edge weights $W : E \to Z_{\leq \text{poly}(n)}$ with $3\log n + 1$ bits per weight; thus $f(n) = |E| \cdot (3\log n + 1) = \text{poly}(n)$.

2. For all $k \in [0, n]$ in parallel, use Lemma 3.3 to compute a matching of size k, which succeeds if one is isolated by W.

3. For all $k \in [0, n]$ such that Lemma 3.3 gives a size k matching, use Lemma 5.3 to check whether a matching of size $k + 1$ is isolated.

4. Let k^* be the minimum k for which Lemma 3.3 returns a size k matching of G, but Lemma 5.3 returns a threshold edge e^*. If no such k^* exists, it implies that W isolates a maximum matching of G. In this case C is not required to compress accurately, so it can simply output the all 0s string $0^{f(n)-1}$.

5. For all $0 \leq k \leq k^*$, we are guaranteed that W isolates a size k matching in G. We know that W does not isolate a size $k^* + 1$ matching and, from the previous step, we have a threshold edge e^*.

6. Comp now outputs the original weight function W but with the $3\log n + 1$ bits corresponding to weight $W(e^*)$ replaced with $3\log n$ bits representing k^* and e^*, which we move to the end of the output for simplicity.

Now we describe the NC algorithm which corresponds to Decomp, and we only consider the case that the compression succeeds, i.e. does not output $0^{f(n)-1}$, as the other case is irrelevant:

1. Read the last $3\log(n)$ bits of our input and interpret them as k^* and e^* as described above.

2. Using Lemma 3.3, construct the isolated size k^* matching $M^{k^*}_{\text{unique}}$ in the graph $G \setminus \{e^*\}$.

3. Using $M^{k^*}_{\text{unique}}$, construct its residual graph G_{residual}, and then run the procedure described in Lemma 5.4 to obtain $W(e^*)$.

4. Erase the $3\log n$ bits in the suffix of the input and output the weight function given as input with the recomputed $3\log n + 1$ bit weight $W(e^*)$ in its appropriate position.

It is very easy to verify that all of the algorithms in the referenced lemmas work in NC, as they only use subroutines from L, NL, and TC^1, and operate in parallel for all k. Thus, both Comp and Decomp are in uniform NC, and the algorithm $\mathcal{A}_1$ simply constructs the circuits C and D corresponding to the uniform NC algorithms above.

For $\mathcal{A}_2$ we simply observe that the algorithm Comp gives us the matching in the case when it fails to compress, namely in the case where no k outputs a threshold edge. Thus, given such a weight function, the algorithm $\mathcal{A}_2$ can simply run over all $k \in [0, n]$ in parallel and use Lemma 3.3 to attempt to construct a size k matching. The largest matching M^k_{unique} for which this algorithm succeeds is guaranteed to be the maximum matching, which it can simply output. Thus $\mathcal{A}_2$ is also an NC algorithm by the same argument as Comp. □

6.3 Postscript: a note on the isolation lemma

To close the main section of our paper, we note an interesting feature of our approach as discussed in the introduction. Our key observation in Section 5 is that in the case where weights W are not isolating, an edge e exists which is in *some* minimum weight matchings, *but not all* of them. This is, in fact, a general feature of non-isolating weights on arbitrary families of sets. In the original proof of [MVV87], they refer to this element as being "on the threshold" (hence our use of the term "threshold edge"), and use it to analyse the probability of failure.

In theory, the weight of a threshold element could always be erased and reconstructed later. Thus, the approach we have described here could be used to derandomize in CL (or in $\mathsf{LOSSY}[\mathcal{C}]$) any algorithm which employs the isolation lemma. The bottleneck is of course designing efficient compression-decompression routines for these problems (which correspond to the circuits Comp and Decomp in the case of lossy coding). Our contribution is thus twofold: we observe that the weight of a threshold element can be erased and later reconstructed, and we design a novel approach to identify a threshold element and reconstruct its weight in the special case of MATCH.

7 Related Problems

In this section we discuss the implications of our result for related problems.

Corollary 7.1. *The following search problems are in* CLP*:*

1. *minimum weight maximum matching with polynomially bounded weights*
2. *directed s-t maximum flow in general graphs with polynomially bounded capacities*
3. *global minimum cut in general graphs*

Proof. We sketch how each point follows from our earlier algorithm in turn.

Min-weight matching. Let W_{input} be the edge weights given as input, with respect to which we want to find a minimum weight maximum matching. We will again iteratively use a weighting scheme $W_{\text{catalytic}}$ as given on the catalytic tape by using edge weights $W := W_{\text{input}} \cdot n^{100} + W_{\text{catalytic}}$. As with our original algorithm, in each iteration we either find a minimum weight maximum

matching according to W, which is clearly also a minimum weight maximum matching for weights W_{input}, or we find a redundant weight in W, which also gives a redundant weight in $W_{\text{catalytic}}$ which we can compress. As before we ultimately free up poly(n) space on the catalytic tape and again use to run any deterministic polytime algorithm for minimum weight maximum matching, such as the one in [Kuh55].

Directed max-flow. This directly follows by known reductions: Madry [Mad13] showed a logspace reduction from log-bit directed s-t maximum flow to log-bit bipartite b-matching, and there is a trivial reduction from log-bit bipartite b-matching to bipartite matching.

Global min-cut. We first note that this is immediate for log-bit weights, as one can compute the s-t minimum cut for every $(s,t) \in V \times V$ using the max-flow algorithm described above and simply take the minimum.

We now move onto the case of poly(n)-bit weights. Karger and Motwani [KM94] proved that the following algorithm converts poly-bit weights to log-bit weights, such that the global minimum cut with respect to the original weights remains a 2-approximate global minimum cut with respect to the new weights:

1. Construct a maximum spanning tree T. Let the minimum weight of any edge in this tree be w.
2. For edges e such that $W(e) > n^2 w$, set $W'(e) = n^{10}$. For all other edges e set $W'(e) = W(e) \cdot n^3/w$, rounded off to the nearest integer.

Using Reingold's celebrated result [Rei08] that undirected s-t connectivity is in log-space, one can construct a maximum spanning tree in log-space, since edge $e = (u,v)$ is in the maximum spanning tree iff u is not reachable from v using edges of weight $\geq W(e)$ (for ties, we additionally filter to edges e' with greater index $e' > e$). Using this fact, the aforementioned reduction is in CLP.

Thus, we simply need to enumerate the 2-approximate global minimum cuts of G with respect to a log-bit weight function. A recent result of Beideman, Chandrawekaran, and Wang [BCW23] shows that for every such cut $(C, \bar{C})$, there exist sets $S \subseteq C$, $T \subseteq \bar{C}$ with $|S|, |T| \leq 10$ such that $(C, \bar{C})$ is the unique minimum cut separating S from T. Thus, we can simply iterate over all such sets S and T, contract S and T into single vertices, and compute the minimum S-T cut again using our CLP max-flow algorithm and the max-flow/min-cut theorem. Finally, we output the cut that is minimal with respect to the original weights. □

Acknowledgements

The first author thanks Danupon Nanongkai and Samir Datta for lengthy and insightful discussions about bipartite matching and the isolation lemma. The second author thanks Michal Koucký, Ted Pyne, Sasha Sami, and Ninad Rajgopal for early conversations about compression and the isolation lemma. Both authors thank Samir Datta, Danupon Nanongkai, and Ted Pyne for detailed comments on an earlier draft, as well as Ted Pyne and Roei Tell for discussions on lossy coding.

References

[AGT20] Manindra Agrawal, Rohit Gurjar, and Thomas Thierauf. "Impossibility of Derandomizing the Isolation Lemma for all Families". In: *Electron. Colloquium Comput. Complex.* Vol. 27. 2020, p. 98 (cit. on p. 4).

[ARZ99] Eric Allender, Klaus Reinhardt, and Shiyu Zhou. "Isolation, Matching, and Counting Uniform and Nonuniform Upper Bounds". In: *Journal of Computer and System Sciences (J.CSS)* 59.2 (1999), pp. 164–181 (cit. on p. 1).

[AV19] Nima Anari and Vijay V Vazirani. "Matching is as easy as the decision problem, in the NC model". In: *arXiv preprint arXiv:1901.10387* (2019) (cit. on pp. 1, 5).

[AV20] Nima Anari and Vijay V Vazirani. "Planar graph perfect matching is in NC". In: *Journal of the ACM (JACM)* 67.4 (2020), pp. 1–34 (cit. on pp. 1, 4, 5).

[AM08] Vikraman Arvind and Partha Mukhopadhyay. "Derandomizing the isolation lemma and lower bounds for circuit size". In: *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques: 11th International Workshop, APPROX 2008, and 12th International Workshop, RANDOM 2008, Boston, MA, USA, August 25-27, 2008. Proceedings*. Springer. 2008, pp. 276–289 (cit. on p. 4).

[BBRS98] Greg Barnes, Jonathan F Buss, Walter L Ruzzo, and Baruch Schieber. "A sublinear space, polynomial time algorithm for directed st connectivity". In: *SIAM Journal on Computing* 27.5 (1998), pp. 1273–1282 (cit. on p. 1).

[BCW23] Calvin Beideman, Karthekeyan Chandrasekaran, and Weihang Wang. "Approximate minimum cuts and their enumeration". In: *Symposium on Simplicity in Algorithms (SOSA)*. SIAM. 2023, pp. 36–41 (cit. on p. 21).

[Ber57] Claude Berge. "TWO THEOREMS IN GRAPH THEORY". In: *Proceedings of the National Academy of Sciences* 43.9 (1957), pp. 842–844 (cit. on p. 12).

[BDS22] Sagar Bisoyi, Krishnamoorthy Dinesh, and Jayalal Sarma. "On pure space vs catalytic space". In: *Theoretical Computer Science (TCS)* 921 (2022), pp. 112–126 (cit. on p. 2).

[BTV09] Chris Bourke, Raghunath Tewari, and NV Vinodchandran. "Directed planar reachability is in unambiguous log-space". In: *ACM Transactions on Computation Theory (TOCT)* 1.1 (2009), pp. 1–17 (cit. on p. 4).

[BCKLS14] Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. "Computing with a full memory: catalytic space". In: *ACM Symposium on Theory of Computing (STOC)*. 2014, pp. 857–866 (cit. on pp. 2, 3, 8).

[BKLS18] Harry Buhrman, Michal Koucký, Bruno Loff, and Florian Speelman. "Catalytic Space: Non-determinism and Hierarchy". In: *Theory of Computing Systems (TOCS)* 62.1 (2018), pp. 116–135 (cit. on p. 2).

[CRS93] Suresh Chari, Pankaj Rohatgi, and Aravind Srinivasan. "Randomness-optimal unique element isolation, with applications to perfect matching and related problems". In: *Proceedings of the twenty-fifth annual ACM symposium on Theory of Computing*. 1993, pp. 458–467 (cit. on p. 4).

[CLORS23] Lijie Chen, Zhenjian Lu, Igor C. Oliveira, Hanlin Ren, and Rahul Santhanam. "Polynomial-Time Pseudodeterministic Construction of Primes". In: *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*. IEEE, 2023, pp. 1261–1270 (cit. on p. 4).

[CLMP25] James Cook, Jiatu Li, Ian Mertz, and Edward Pyne. "The Structure of Catalytic Space: Capturing Randomness and Time via Compression". In: *ACM Symposium on Theory of Computing (STOC)*. 2025 (cit. on pp. 2, 6, 7, 18).

[CM20] James Cook and Ian Mertz. "Catalytic approaches to the tree evaluation problem". In: *ACM Symposium on Theory of Computing (STOC)*. ACM, 2020, pp. 752–760 (cit. on p. 2).

[CM21] James Cook and Ian Mertz. "Encodings and the Tree Evaluation Problem". In: *Electronic Colloquium on Computational Complexity (ECCC)* TR21-054 (2021) (cit. on p. 2).

[CM22] James Cook and Ian Mertz. "Trading Time and Space in Catalytic Branching Programs". In: *IEEE Conference on Computational Complexity (CCC)*. Vol. 234. Leibniz International Proceedings in Informatics (LIPIcs). 2022, 8:1–8:21 (cit. on p. 2).

[CM24] James Cook and Ian Mertz. "Tree Evaluation Is in Space O(log n · log log n)". In: *ACM Symposium on Theory of Computing (STOC)*. ACM, 2024, pp. 1268–1278 (cit. on p. 2).

[CMWBS12] Stephen A. Cook, Pierre McKenzie, Dustin Wehr, Mark Braverman, and Rahul Santhanam. "Pebbles and Branching Programs for Tree Evaluation". In: *ACM Transactions on Computational Theory (TOCT)* 3.2 (2012), 4:1–4:43 (cit. on p. 2).

[CLRS22] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022 (cit. on p. 11).

[DGJST20] Samir Datta, Chetan Gupta, Rahul Jain, Vimal Raj Sharma, and Raghunath Tewari. "Randomized and Symmetric Catalytic Computation". In: *CSR*. Vol. 12159. Lecture Notes in Computer Science (LNCS). Springer, 2020, pp. 211–223 (cit. on p. 2).

[DKR10] Samir Datta, Raghav Kulkarni, and Sambuddha Roy. "Deterministically isolating a perfect matching in bipartite planar graphs". In: *Theory of Computing Systems* 47.3 (2010), pp. 737–757 (cit. on pp. 1, 13).

[DPT24] Dean Doron, Edward Pyne, and Roei Tell. "Opening Up the Distinguisher: A Hardness to Randomness Approach for BPL = L that Uses Properties of BPL". In: *ACM Symposium on Theory of Computing (STOC)*. 2024, pp. 2039–2049 (cit. on pp. 2, 3).

[DPTW25] Dean Doron, Edward Pyne, Roei Tell, and Ryan Williams. "When Connectivity is Hard, Random Walks Are Easy With Non-Determinism". In: *ACM Symposium on Theory of Computing (STOC)*. 2025 (cit. on pp. 2, 3, 5).

[FGT16] Stephen Fenner, Rohit Gurjar, and Thomas Thierauf. "Bipartite perfect matching is in quasi-NC". In: *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*. 2016, pp. 754–763 (cit. on pp. 1, 4, 5).

[FMST25] Marten Folkertsma, Ian Mertz, Florian Speelman, and Quinten Tupker. "Fully Characterizing Lossy Catalytic Computation". In: *Innovations in Theoretical Computer Science Conference (ITCS)*. Vol. 325. Leibniz International Proceedings in Informatics (LIPIcs). 2025, 50:1–50:13 (cit. on p. 2).

[GG17] Shafi Goldwasser and Ofer Grossman. "Bipartite perfect matching in pseudo-deterministic NC". In: (2017) (cit. on pp. 1, 5).

[GJST19] Chetan Gupta, Rahul Jain, Vimal Raj Sharma, and Raghunath Tewari. "Unambiguous Catalytic Computation". In: *Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. Vol. 150. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 16:1–16:13 (cit. on p. 2).

[GJST24] Chetan Gupta, Rahul Jain, Vimal Raj Sharma, and Raghunath Tewari. "Lossy Catalytic Computation". In: *Computing Research Repository (CoRR)* abs/2408.14670 (2024) (cit. on p. 2).

[GT17] Rohit Gurjar and Thomas Thierauf. "Linear matroid intersection is in quasi-NC". In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*. 2017, pp. 821–830 (cit. on pp. 4, 5).

[GTV21] Rohit Gurjar, Thomas Thierauf, and Nisheeth K Vishnoi. "Isolating a vertex via lattices: Polytopes with totally unimodular faces". In: *SIAM Journal on Computing* 50.2 (2021), pp. 636–661 (cit. on pp. 4, 5).

[HMT06] Thanh Minh Hoang, Meena Mahajan, and Thomas Thierauf. "On the bipartite unique perfect matching problem". In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2006, pp. 453–464 (cit. on p. 11).

[HK73] John E. Hopcroft and Richard M. Karp. "An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs". In: *SIAM Journal on Computing* 2.4 (1973), pp. 225–231. eprint: `https://doi.org/10.1137/0202019` (cit. on pp. 1, 17).

[KT16] Vivek Anand T Kallampally and Raghunath Tewari. "Trading determinism for time in space bounded computations". In: *arXiv preprint arXiv:1606.04649* (2016) (cit. on pp. 4, 5).

[KM94] David R Karger and Rajeev Motwani. "Derandomization through approximation: An NC algorithm for minimum cuts". In: *Proceedings of the twenty-sixth annual ACM symposium on Theory of Computing*. 1994, pp. 497–506 (cit. on p. 21).

[Kar86] Howard J Karloff. "A Las Vegas RNC algorithm for maximum matching". In: *Combinatorica* 6.4 (1986), pp. 387–391 (cit. on p. 5).

[KUW85] Richard M Karp, Eli Upfal, and Avi Wigderson. "Constructing a perfect matching is in random NC". In: *Proceedings of the seventeenth annual ACM symposium on Theory of computing*. 1985, pp. 22–32 (cit. on pp. 1, 5).

[KR98] Marek Karpiński and Wojciech Rytter. *Fast parallel algorithms for graph matching problems*. 9. Oxford University Press, 1998 (cit. on p. 11).

[KS01] Adam R Klivans and Daniel Spielman. "Randomness efficient identity testing of multivariate polynomials". In: *Proceedings of the thirty-third annual ACM symposium on Theory of computing*. 2001, pp. 216–223 (cit. on p. 4).

[Kor22] Oliver Korten. "Derandomization from time-space tradeoffs". In: *37th Computational Complexity Conference (CCC 2022)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. 2022, pp. 37–1 (cit. on pp. 3, 4).

[Kor25] Oliver Korten. "Range Avoidance and the Complexity of Explicit Constructions". In: *Bulletin of the EATCS (B.EATCS)* 145 (2025), pp. 94–134 (cit. on p. 4).

[KP24] Oliver Korten and Toniann Pitassi. "Strong vs. Weak Range Avoidance and the Linear Ordering Principle". In: *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*. IEEE, 2024, pp. 1388–1407 (cit. on p. 4).

[Kou16] Michal Koucký. "Catalytic computation". In: *Bulletin of the EATCS (B.EATCS)* 118 (2016) (cit. on pp. 2, 3).

[KMPS25] Michal Koucký, Ian Mertz, Ted Pyne, and Sasha Sami. "Collapsing Catalytic Classes". In: *Electronic Colloquium on Computational Complexity (ECCC)* TR25-018 (2025) (cit. on p. 2).

[Kuh55] Harold W Kuhn. "The Hungarian method for the assignment problem". In: *Naval research logistics quarterly* 2.1-2 (1955), pp. 83–97 (cit. on pp. 1, 13, 21).

[LPT24] Jiatu Li, Edward Pyne, and Roei Tell. "Distinguishing, Predicting, and Certifying: On the Long Reach of Partial Notions of Pseudorandomness". In: *IEEE Symposium on Foundations of Computer Science (FOCS)*. to appear. 2024 (cit. on pp. 2, 3).

[LP15] Andrzej Lingas and Mia Persson. "A fast parallel algorithm for minimum-cost small integral flows". In: *Algorithmica* 72 (2015), pp. 607–619 (cit. on p. 4).

[Lov79] László Lovász. "On determinants, matchings, and random algorithms." In: *FCT*. Vol. 79. 1979, pp. 565–574 (cit. on pp. 1, 5).

[Mad13] Aleksander Madry. "Navigating central path with electrical flows: From flows to matchings, and back". In: *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. IEEE. 2013, pp. 253–262 (cit. on p. 21).

[MV00] Meena Mahajan and Kasturi R Varadarajan. "A new NC-algorithm for finding a perfect matching in bipartite planar and small genus graphs". In: *Proceedings of the thirty-second annual ACM symposium on Theory of computing*. 2000, pp. 351–357 (cit. on p. 1).

[MV97] Meena Mahajan and V. Vinay. "Determinant: Combinatorics, Algorithms, and Complexity". In: *Chic. J. Theor. Comput. Sci.* 1997 (1997) (cit. on p. 8).

[Mer23] Ian Mertz. "Reusing Space: Techniques and Open Problems". In: *Bulletin of the EATCS (B.EATCS)* 141 (2023), pp. 57–106 (cit. on pp. 2, 3).

[MN89] Gary L Miller and Joseph Naor. "Flow in planar graphs with multiple sources and sinks". In: *FOCS*. 1989, pp. 112–117 (cit. on p. 1).

[MVV87] Ketan Mulmuley, Umesh V Vazirani, and Vijay V Vazirani. "Matching is as easy as matrix inversion". In: *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. 1987, pp. 345–354 (cit. on pp. 1, 4–6, 9, 10, 13, 20).

[NSV94] Hariharan Narayanan, Huzur Saran, and Vijay V Vazirani. "Randomized parallel algorithms for matroid union and intersection, with applications to arborescences and edge-disjoint spanning trees". In: *SIAM Journal on Computing* 23.2 (1994), pp. 387–397 (cit. on p. 4).

[OS93] James B Orlin and Clifford Stein. "Parallel algorithms for the assignment and minimum-cost flow problems". In: *Operations research letters* 14.4 (1993), pp. 181–186 (cit. on p. 4).

[Pyn24] Edward Pyne. "Derandomizing Logspace with a Small Shared Hard Drive". In: *IEEE Conference on Computational Complexity (CCC)*. Vol. 300. LIPIcs. 2024, 4:1–4:20 (cit. on pp. 2–4).

[PSW25] Edward Pyne, Nathan S. Sheffield, and William Wang. "Catalytic Communication". In: *16th Innovations in Theoretical Computer Science Conference, ITCS 2025, January 7-10, 2025, Columbia University, New York, NY, USA*. Ed. by Raghu Meka. Vol. 325. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025, 79:1–79:24 (cit. on p. 2).

[Rei08] Omer Reingold. "Undirected connectivity in log-space". In: *Journal of the ACM (J.ACM)* 55.4 (2008), 17:1–17:24 (cit. on p. 21).

[RA00] Klaus Reinhardt and Eric Allender. "Making nondeterminism unambiguous". In: *SIAM Journal on Computing* 29.4 (2000), pp. 1118–1131 (cit. on p. 4).

[ST17] Ola Svensson and Jakub Tarnawski. "The matching problem in general graphs is in quasi-NC". In: *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*. Ieee. 2017, pp. 696–707 (cit. on pp. 1, 5).

[VP19] Dieter Van Melkebeek and Gautam Prakriya. "Derandomizing isolation in space-bounded settings". In: *SIAM Journal on Computing* 48.3 (2019), pp. 979–1021 (cit. on pp. 4, 5).

[Wil25] Ryan Williams. "Simulating Time in Square-Root Space". In: *ACM Symposium on Theory of Computing (STOC)*. 2025 (cit. on p. 2).