Brute Force Search and Oracle-Based Computation *

Ryan Williams†

(Extended Abstract)

1 Introduction

We revisit the notion of searching for a solution over exponentially many possibilities, and introduce a simple model similar to the bottleneck/safe-storage machines of Cai and Furst [4] to represent this kind of search. One may think of our model as a precise machine-theoretic characterization of the concept of *perebor*, or an exhaustive "brute-force" search for a solution through all possible answers. (For a highly informative survey of this work, see Trakenbrot's historical account [17].) The model consists of a logspace solution-checker performing sequential computations on each polynomial-sized candidate in lexicographic order, starting with the smallest, and it may accept or reject at any point during the computation. It follows almost immediately from a construction in Cai and Furst's model that the class of languages accepted by this model is equal to $PSPACE$, even we allow the machine to pass only 3 bits of information from one solution check to the next one. However, unlike their model, when the logspace machine is forced to reset to its initial configuration every time a new solution is considered, the model recognizes exactly $\Delta_2^p = P^{NP}$, i.e. the set of languages accepted by polynomial time Turing machines with access to an oracle for SAT. The Δ_2^p result demonstrates a deterministic, physically intuitive model capturing a well-studied, oracle-based complexity class. One application of the equivalence is a quantifier-based characterization of the Δ_2^p languages, while in the past Δ_2^p has been notoriously free of many natural complete problems.

2 Defining the model

Our models of brute-force search will have two basic components: a "solution checker", an algorithm using little resources that verifies a given candidate solution, and a "search mechanism" that provides a list of solutions to check. This model has the potential to run in exponential time, as the list may be very long; but once acception/rejection occurs, the search halts. Since our search is completely uninformed (by definition), we will essentially prefer no solution over another during checking. Thus the ordered list of possible solutions will be independent of the input. For this paper, we will use the lexicographic ordering on strings as a list.

A natural question to ask of this model is: how much information about the computation should be preserved, from the checking of one solution to the next? Should all solution checks be independent, in the sense that no information is carried from one check to the next (other than the fact that all of the candidates lexicographically smaller than the current one have already been examined)? Or should we allow a small amount of information, to pass from one phase of checking to the next? As one might expect, this measure directly determines the power of the model. We will first assume no restrictions on the amount of "passed" information, and will experiment with this idea later.

*Topics: **Computational and structural complexity; Automata and formal languages.**

†Dept. of Comp. Sci., Cornell University, Ithaca, NY, 14850, USA. Email: rrw9@cornell.edu. Supported by a National Science Foundation Graduate Research Fellowship.

We propose the following model: a logspace Turing machine with two-way read-only access to a polynomial tape (which we will call a "reference") containing a binary string that is initially all zeroes. The three special sets of states for this machine will be the usual accept and reject states, and additional *increment states*. When a machine enters such a state, the string on the reference, treated as a binary number, is incremented by one. Thus the kind of access to the reference is twofold: reading, and incrementing. We will call such access to the reference *increment access*, for obvious reasons. With increment access, we will define the following class. Inspired by the elegant notation of descriptive complexity [8], the "plus" notation is being used to express the conjunction of logspace machines with polynomial increment devices.

Definition 2.1 *$L + PI$ (logspace plus polynomial increment) is the class of languages accepted by logspace Turing machines with 2-way read-only increment access to polynomial reference. We enforce acception and rejection behavior: when the reference contains all ones, the machine must accept or reject in this position.*

This model is quite powerful; we will see that such machines are as powerful as fully general polynomial space machines. Naturally, after having jumped from NP (a class that came out of studying universal search) to $PSPACE$ with $L + PI$, we wonder if one can modify the model to capture classes between these two. Perhaps it may seem that logspace is a large amount of working storage to give to a machine that is performing brute-force search of a polynomial-sized solution space. A result of Vollmer et. al. [7] along with correlations between our model and one of Cai and Furst [4] shows that if we use AC^0 circuits in place of logspace machines, then the $PSPACE$ equivalence is still achieved. Rather than reducing the resources, we will make the working storage volatile, i.e. we limit the time for which data can be stored. Let $k \geq 0$ be a constant integer.

Definition 2.2 *Define $L + PI_k$ as the class of languages accepted by logspace machines with increment access to polynomial reference, with the restriction that when M enters its increment state, the reference is incremented, and M writes blanks on* **all but k squares** *of its worktape, moves its tape heads to their leftmost positions, and resets to the initial state.*

So in this model, all but a constant amount of information is lost from one solution check to the next.

3 Previous work

A model very similar to the one directly above has been studied in the literature. Cai and Furst [4] define "safe-storage" (or "bottleneck") machines, which have a read-only input tape, read-write worktape, a polynomial binary counter, and a "safe-store" tape that may only be in one of a constant number of states. The machine begins with the counter at 0, and after a polynomial number of steps it moves to a distinguished state where the machine is reset: the worktapes are erased, the tape heads are reset to original positions, the counter is incremented and computation continues from the start state. (Note that the contents of the "safe-store" are left untouched.) They define SF_k as the class of languages accepted by bottleneck polynomial-time bounded Turing machines that safely store one out of k possible states.

It will turn out that the Turing machine's resources (logarithmic space versus polynomial time) is actually not an issue; with a polynomial counter, we can simulate polytime computations with logspace. The crucial difference in our model and theirs is that, in Cai and Furst's model, an input string is accepted if and only if, when the counter is all 1's, the "safe-store" is in a distinguished state. In ours, acceptance and rejection depends on the state of the original logspace Turing machine, and acceptance/rejection may occur at any point during the computation (independent of the counter).

In terms of $PSPACE$, this difference may be overcome, and $L+PI_3 = SF_5 = PSPACE$ may be obtained due to Cai and Furst, using a clever representation of logical operators first described in Barrington's work in bounded-width branching programs [1]. When no bits are safely stored, a large separation occurs between the two notions: while it is easily seen that $SF_0 = P$, we show that $L + PI_0 = \Delta_2^p$.

Ogihara [12] has given some generalizations of the safe-storage model *shr-*SF_k ("shrink" safe-storage, which changes the safe-store value via functions called *shrinks*) and *perm-*SF_k ("permutation" bottleneck machines, which use permutations for changing the safe-store), along with a few equivalences using these models, such as $SF_2 = \oplus P//\mathrm{Opt}P$, *shr-*$SF_2 = \Delta_2^p$, and *perm-*$SF_2 = \oplus P$ (parity-P). However, it is difficult to extract much new information about the classes Δ_2^p and $\oplus P$ from these equivalences, as the models are somewhat artificial in their construction.

In the sequel, we briefly describe how Cai and Furst's result fits in with our work.

4 $L + PI_3 = PSPACE$

$L + PI_3$ is a robust class: clearly it is closed under logspace reductions and has complete languages. [1] The ability to read and increment is intuitively more powerful than reading alone (so the model subsumes NP), but also less powerful than full write access (so it may appear that $PSPACE$ subsumes the class). Incrementing is a restrictive form of writing, in which sometimes only one bit is written, and other times all bits change.

Normally when one thinks of brute force or "universal" search, the class NP comes to mind. However, by the work of Cai and Furst, if a logspace machine is given the capability of accessing the possible solutions to a problem instance in a fixed order (and forgetting everything but three bits for each solution), this access can be exploited to do more than just find one or two or exponentially many solutions.

We first prove that our model subsumes the safe-storage model when the number of states saved by the safe-storage machines is a logarithm of the number of bits preserved during an increment in the $L + PI$ model. To simplify matters, we will assume that the worktapes of logspace machines in the model use the binary alphabet $\{0, 1\}$.

Lemma 4.1 $SF_k \subseteq L + PI_{\lceil \log k \rceil}$.

Proof. Let M be a polynomial time safe-storage machine that saves one of k states. We simulate M using an $L + PI_{\lceil \log k \rceil}$ machine N by representing the contents of its k state safe-store on $\lceil \log k \rceil$ designated worktape squares. The reference of N is enlarged by a polynomial factor for simulating valid computation histories of M directly on the reference. More precisely, if C represents the contents of M's counter, the reference of N will be of the form: $(C, valcomp)$, where *valcomp* is a polynomial length bit string representing a possible valid computation history of M on the current input and counter.

Initially, the reference is $(0 \cdots 0, 0 \cdots 0)$. Simulation of M for a particular C consists of repeatedly searching for a correct *valcomp* for M by incrementing the reference, until a valid one is found (which must exist, since the polynomial time machine is deterministic). In that case, N enters a "distinguished state" (analogous to that of M) where the counter is incremented until it is of the form $(C+1, 0 \cdots 0)$. Such a form can be checked in logspace since *valcomp* is of polynomial size. Then the computation proceeds as before, searching for a new valcomp of M starting from its initial configuration.

Note that this simulation of polynomial time requires no extra safe-storage bits. When M enters its distinguished state, our machine enters its increment state, and when the counter is all 1's we accept/reject accordingly.

□

[1]It is also closed under polytime reductions, since the reference may be used to encode valid computation histories of polynomial time Turing machines. This holds for $L + PI$ and $L + PI_k$ for any k; a construction of this encoding is described in the proof of Lemma 4.1.

Theorem 4.1 $PSPACE = L + PI_3$.

Proof. $L+PI_3 \subseteq PSPACE$ is true since the $L+PI_3$ model always uses no more than polynomial space. Cai and Furst [4] prove that $PSPACE = SF_5$, which is contained in $L + PI_3$ by the lemma.

□

5 A deterministic search formalism for Δ_2^p

In the case of $L + PI_0$, the model lends itself a bit more readily to parallelization; logspace pieces of the computation can be executed independently without interfering with other parts, as long as the references are different. Forcing our model to "reset" its configuration entirely cripples it in a major way. We will take advantage of this computation independence using a binary search, and show that $L + PI_0$ is equivalent to $\Delta_2^p = P^{NP}$, the class of languages accepted by polynomial time Turing machines with access to an oracle for SAT (or any other NP-complete problem, for that matter).

Theorem 5.1 $L + PI_0 \subseteq \Delta_2^p$.

Proof. Let $M_{\log}(\cdot,\cdot)$ denote the *logspace portion* of an arbitrary $L + PI_0$ machine M. That is, $M_{\log}$ is a logspace machine with read-only inputs x and a polynomially related y, with final states being either accepting, rejecting, or incrementing, and

$$\begin{aligned}(\forall x, y \in \Sigma^*, |y| \le |x|^c)[&M(x) \text{ accepts/rejects/increments on reference } y\\ &\iff M_{\log}(x,y) \text{ accepts/rejects/increments}],\end{aligned}$$

for some $c > 0$, i.e. the logspace portion represents what M does when the reference is fixed.

Clearly, specifying the logspace portion of the machine is sufficient for specifying an M. So it suffices to demonstrate that a particular language is both in Δ_2^p, and $L + PI_0$-complete under logspace reductions. Define the universal language

$$U_{L+PI_0} := \{\langle i, k, x, 1^{|x|^k}\rangle : (\exists y : |y| \le |x|^k)(\forall z < y)[M_i^k(x,z) \text{ increments } \wedge\ M_i^k(x,y) \text{ accepts}\},$$

where M_i^k is the ith logspace machine executing in n^k time with respect to some enumeration.

Lemma 5.1 *U_{L+PI_0} is $L + PI_0$-complete under logspace reductions.*

Proof. Let M_i^k be the logspace part of a $L + PI_0$ machine M, and x be an input. By the criterion given for acceptance and rejection, we know there is at least one string y that, when it is set as the reference, $M_i^k(x,y)$ either accepts or rejects. Furthermore, any string z preceding y in the computation of $M(x)$ must have resulted in an increment. To be precise,

$$\begin{aligned} M(x) \text{ accepts} \quad &\iff \quad (\exists y : |y| \le |x|^k)(\forall z < y)[M_i^k(x,y) \text{ accepts } \wedge M_i^k(x,z) \text{ increments}]\\ &\iff \quad \langle M_i^k, x, 1^{|x|^k}\rangle \in U_{L+PI_0}.\end{aligned}$$

Using a standard padding technique, we conclude that U_{L+PI_0} is $L + PI_0$-complete under logspace reductions.

□

Lemma 5.2 $U_{L+PI_0} \in \Delta_2^p$.

Proof. On an input $\langle M_i^k, x, 1^{|x|^k}\rangle$, the following Δ_2^p procedure will test at most $|x|^k$ values for y with the predicate $R(x,y) = (\forall z < y)[M_i^k(x,z) = \text{increment}]$, which can be tested in one step using an NP oracle. A similar kind of predicate was used crucially in Ogiwara and Watanabe's more general proof of Mahaney's theorem; they define a "left set", which is essentially a set of strings that are less than some candidate witness, with respect to lexicographical order. [13]

We perform a simple binary search with this predicate to determine if $\langle M_i^k, x, 1^{|x|^k}\rangle \in U_{L+PI_0}$. Given a string $x = \sigma_1 \cdots \sigma_m \in \Sigma^*$, for $i = 1, \ldots, m$ we define $x[i] := \sigma_1 \cdots \sigma_i$.

Input: $\langle M_i^k, x, 1^{|x|^k}\rangle)$
  Initially, $u := \epsilon$, $v := 10^{|x|^k - 1}$.
  While $v \neq \epsilon$,
    Set $y := uv$.
    If $R(x,y)$ then
      If $M_i^k(x,y)$ accepts/rejects then **accept/reject**, else $u := u1$.
    else $u := u0$.
    End if
    $v := v[|v| - 1]$
  End while

Suppose $M(x)$ accepts/rejects. Then there exists a unique y with $M_i^k(x,y)$ accepting/rejecting, and all $z < y$ are such that $M_i^k(x,z)$ increments. For any string y', observe that $R(x,y')$ is true if and only if $y > y'$. This permits us to perform a binary search on the exponential number of possible y values. If $R(x,y')$ is true in the ith iteration ($i = 1, \ldots, |x|^k$), y' is increased by $2^{|x|^k - i}$. If $R(x,y')$ is false, y' is decreased by $2^{|x|^k - i}$. By repeatedly querying the "midpoint" of the current solution space (such as $10^{|x|^k - 1}$ in the list of strings from $0^{|x|^k}$ to $1^{|x|^k}$), each iteration of the while loop halves the number of solutions under consideration. (This proves that the algorithm executes in polynomial time, making $O(|x|^k)$ queries to the NP oracle.)

Conversely, if the simulation accepts/rejects, then the final y value obtained is such that $M_i^k(x,y)$ accepts/rejects, and the $z < y$ all increment, by definition of $R(x,y)$.

□

We now want to prove the opposite containment that $\Delta_2^p \subseteq L + PI_0$. As in the $L + PI = PSPACE$ proof, we will verify valid computation histories that claim the machine in question accepts or rejects on the given input. To do this, we formulate valid computation histories for Δ_2^p, using a simple extension of valcomps for nondeterministic machines.

Let N be execute in n^k nondeterministic time, and assume without loss of generality that $x1^{n^{2k}}$ is not a valid computation history of N for any x.

Definition 5.1 *The set of extended valcomps of N on x is*

$$\texttt{VALCOMP}^*(N,x) := \begin{cases} \texttt{VALCOMP}(N,x) & \text{if } N(x) \text{ accepts,} \\ \{x1^{|x|^{2k}}\} & \text{otherwise} \end{cases}$$

where $\texttt{VALCOMP}(N,x)$ is the well-known set of valid (accepting) computation histories of N on x.

Let us generalize the definition for P^{NP} machines. Let x be an input, and M^N be the P^{NP} machine under consideration, where the query machine $M^?$ and N execute in n^k and n^c time, respectively. We assume $M^?$ has a separate query tape with content y, and entering state $q^?$ of $M^?$ represents querying if $N(y)$ accepts or not.

Label the configurations of $M^N(x)$ that are in state $q^?$ as $C_{i_1}, \ldots, C_{i_Q}$, and their respective queries as $y_1, \ldots, y_Q$. Recall that each C_{i_j} is of the form $y_j z$, for some string z (the input is given first). We label configurations in the order by which they appear, i.e. $C_{i_j} \vdash^* C_{i_{j+1}}$ (with respect to $M^N(x)$). Let C_{n^k} be the final configuration.

Definition 5.2 *The extended valcomp of* $M^N(x)$ *is* $\mathtt{VALCOMP}^*(M^N, x) :=$

$$C_1 C_2 \cdots C_{i_1} \mathtt{VALCOMP}^*(N, y_1) C_{i_1+1} \cdots C_{i_2} \mathtt{VALCOMP}^*(N, y_2) C_{i_2+1} \cdots\cdots C_{i_Q} \mathtt{VALCOMP}^*(N, y_Q) C_{i_Q+1} \cdots C_{n^k}.$$

Note that $\mathtt{VALCOMP}^*(M^N, x)$ is a set of strings, since $\mathtt{VALCOMP}^*(N, y_i)$ is a set. It is easy to verify that $M^N(x)$ accepts $\iff$ $\mathtt{VALCOMP}^*(M^N, x) \neq \varnothing$.

Theorem 5.2 $\Delta_2^p \subseteq L + PI_0$.

Proof 1. Given a P^{NP} machine M^N, we claim that the following is a correct simulation of $M^N(x)$ as carried out by a $L + PI_0$ machine:

Procedure $SIM_{M,N}(x)$:

> Attempt to verify that the current contents of the reference is a member of $\mathtt{VALCOMP}^*(M^N, x)$ via this process: verify the configurations for M in the usual way (by checking that each successive configuration follows from the previous one via some transition). For a proposed query $N(y_i)$, try to verify that the valcomp for $N(y_i)$ is accepting, and interpret $y_i 1^{|y_i|^{2c}}$ as a valcomp that represents a "no" answer to the query. If a mistake is found (or some $N(y_i)$ is not accepting), increment the reference and start the whole process over. Return the final outcome (acception or rejection of $M^N(x)$) of the first string found to be in $\mathtt{VALCOMP}^*(M^N, x)$.

Clearly, this procedure can be performed by machine as given in the definition of $L + PI_0$.

We now show that $SIM_{M,N}(x)$ accepts (resp. rejects) if and only if the contents of the reference is the lexicographically first member of $\mathtt{VALCOMP}^*(M^N, x)$, i.e. $M^N(x)$ accepts (resp. rejects).

For the 'if' direction, suppose the reference contents currently hold a bonafide extended valcomp of $M^N(x)$. Then it is clear that $SIM_{M,N}(x)$ will not find errors in its simulation, and thus will accept or reject accordingly.

For the 'only if' direction, note that for a given x, there is a unique list of queries $y_1, \ldots, y_Q$ made by $M^?(x)$. This is because M is deterministic, and each query y_i has one possible outcome. Let us recall the expression for valid computation histories of M^N:

$$C_1 \cdots C_{i_1} \mathtt{VALCOMP}^*(M, y_1) C_{i_1+1} \cdots\cdots\cdots C_{i_Q} \mathtt{VALCOMP}^*(N, y_Q) C_{i_Q+1} \cdots C_{n^k}.$$

The proof is by induction on Q. If $Q = 0$, then the valcomp above is simply that of a polytime machine; if $SIM_{M,N}(x)$ accepts, then clearly the resulting string is a member of $\mathtt{VALCOMP}^*(M^N, x)$.

For the induction step, observe that there is a point in the simulation where $SIM_{M,N}(x)$ has restarted itself, and the reference is of the form $C_1 C_2 \cdots C_{i_1} y_1 0000 \cdots 0000$, since every configuration of $M^N(x)$ (starting from the initial state) always has a successor, or is final. Now the incremental search over all possible strings of length n^{2c} for a member of $\mathtt{VALCOMP}^*(N, y_1)$ continues until either some accepting valcomp is found, or $1^{|y_1|^{2c}}$ appears. In the first case, it is obvious that the query answer must be "yes". In the second, we claim this implies the query answer must be "no". Suppose not, and that there is an accepting valcomp string v

of $N(y_1)$, which appeared on the reference tape before $1^{|y_1|^{2c}}$ due to the ordering. Then the answer would have been "yes", and the computation would have continued with searching for C_{i_1+1}. The crucial point is that v *would have remained on the reference for the rest of* $SIM_{M,N}(x)$*'s run.* This is because by induction, the remaining (correct) valcomp, consisting of $Q-1$ queries, will be returned by $SIM_{M,N}(x)$ before $1^{|y_1|^{2c}}$ is reached. Hence, if $1^{|y_1|^{2c}}$ is reached, then there was no accepting valcomp of $N(y_1)$. By induction, the rest of the string which is returned is valid, so the entire string is an extended valcomp of $M^N(x)$.

□

Proof 2. A shorter but less educational proof of the same containment can be done by (a) giving a $L+PI_0$ machine M that accepts *Uniquely Optimal TSP*, which was shown by Papadimitriou to be Δ_2^p-complete [14], and (b) pointing out that $L+PI_0$ is closed under polynomial time reductions.[2] On input G, the reference will consist of a polysize string *cost*, followed by two polysize strings $tour_1$ and $tour_2$, which represent the proposed cost of the current tours, and two candidate tours. First, the machine checks if $tour_1$ designates a well-formed tour; if not, it increments. If $tour_1$ does not have cost *cost*, increment. If $tour_2$ is all ones, accept. If $tour_2 \neq tour_1$ and $tour_2$ has cost *cost*, reject. If the tape is all ones (there is no tour at all), reject. Otherwise, increment.

Though not explicit in its specification, the machine works in two phases. In the first, it finds the first string such that $tour_1$ is a Hamiltonian tour with cost *cost*. (Note by the ordering that $tour_1$ is optimal cost.) Secondly, it searches for a $tour_2$ with the same cost. If none is found ($tour_2$ is all ones, yet $tour_1$ has cost *cost*) then acceptance occurs; the tour is unique. Otherwise, some $tour_2$ has the same cost, and rejection happens.

□

The above result equates a complexity class defined in terms of an oracle based, "non-physical" model, with one defined by a deterministic, "mechanical" model. The dichotomy between the two models is striking: while one executes in polynomial time but receives advice from an oracle, the other uses exponential time via an uninformed search of a reference. This leads us to the question of "reference complexity", which will be addressed later.

This characterization allows us to formulate Δ_2^p problems more elegantly. Call any k-valued n-ary predicate R that can be evaluated in logspace a *$(k,n)L$-predicate*. The below follows from the $L+PI_0$-completeness of U_{L+PI_0}.

Corollary 5.1 *$\mathcal{L} \in \Delta_2^p$ if and only if there is a (3,2)L-predicate R and constant $c > 0$ such that*

$$\mathcal{L} = \{x : (\exists y : |y| \leq |x|^c)(\forall z < y)[R(x,y) = 1 \wedge R(x,z) = 2]\}.$$

It is intriguing that such a clean characterization of this class exists, given the historical difficulty in finding natural Δ_2^p -complete languages (e.g. [14, 18, 9, 11]). In a rough sense, we might say that finding lexicographically first solutions to NP problems characterizes those problems in Δ_2^p. This would not be entirely true, as $(3,2)L$ predicates have three values, not the traditional binary (accept/reject) values that NP problems have. But the lexicographic condition in itself is not completely a surprise; Miyano [11] found that the problem of determining if a given vertex v is contained in the lexicographically first maximal connected subgraph of a given graph G is Δ_2^p-complete, for particular types of subgraphs. As shown above, the *Uniquely Optimal TSP* problem can be encoded in a "lexicographically first" sense. Krentel [9] proved close connections between optimization problems (the class *OptP*) and Δ_2^p, while demonstrating that the lexicographically maximum satisfying assignment (with output) is complete for *OptP*. So the relationship between lexicographic ordering and P with an NP oracle has been vague, but present in past literature.

[2] The *Uniquely Optimal TSP* problem is: given an instance of TSP (a graph G with weights on edges), is there an optimal tour that is unique?

Access	Initial contents of Reference	Complexity Class
2-way, increment	Blank (all zeroes)	$PSPACE$
2-way, increment, reset	Blank (all zeroes)	Δ_2^p
2-way, read-only	Arbitrary (Nondeterministic)	NP
2-way, read, write-once	Blank	P
1-way, read-only	Arbitrary (Nondeterministic)	NL

Table 1: Some access complexity representations of a few classes from [19].

Contrast the following three formulations:

1. $\mathcal{L} \in \text{co-}NP \iff (\exists\ (2,2)\text{L-predicate } R)\ \mathcal{L} = \{x : (\forall^p z)[R(x,z) = 1]\}$.
2. $\mathcal{L} \in \Delta_2^p \iff (\exists\ (3,2)\text{L-predicate } R)\ \mathcal{L} = \{x : (\exists^p y)(\forall z < y)[R(x,y) = 1 \wedge R(x,z) = 2]\}$.
3. $\mathcal{L} \in \Sigma_2^p \iff (\exists\ (2,3)\text{L-predicate } R)\ \mathcal{L} = \{x : (\exists^p y)(\forall^p z)[R(x,y,z) = 1]\}$.

The Δ_2^p formulation has two quantifiers, like Σ_2^p, but places an ordering restriction on the universal quantifier, and has only a binary predicate, though this predicate is three-valued. A similar analogy can be drawn between NP, Δ_2^p, and Π_2^p.

Finally, it is instructive to consider a similar characterization of Δ_2^0 in the arithmetical hierarchy. Let $g : N^2 \to N$ and $f : N \to N$. We say that $f = \lim_y g$ if and only if $(\forall x)(\exists y)(\forall z \geq y)[g(x,z) = f(x)]$. So for any x, there exists a string y such that, for all z beyond that, g and f agree on x.

Lemma 5.3 *(Limit Lemma, [15]) Let $f : N \to N$ be given. Then $f \in \Delta_2^0$ if and only if there is a recursive function $g : N^2 \to N$ such that $f = \lim_s g$.*

Note that this lemma deals with the arithmetical hierarchy of functions. We do not prove the lemma here, but point out that it leads to the following equivalence for Δ_2^0 in the arithmetical hierarchy of sets. Let $\mathcal{L} \subseteq N$. Then $\mathcal{L} \in \Delta_2^0 \iff (\exists \text{ recursive } g : N^2 \to N)\ \ \mathcal{L} = \{x : (\exists y)(\forall z \geq y)[g(x,z) = 1]\}$.

By tweaking our model (or the ordering on strings) in a straightforward way, we similarly have in the polynomial time hierarchy that for $\mathcal{L} \subseteq \Sigma^*$,

$$\mathcal{L} \in \Delta_2^p \iff (\exists\ (3,2)L-\text{predicate } R)\ \ \mathcal{L} = \{x : (\exists^p y)(\forall z \geq y, |z| = |y|)[R(x,y) = 1 \wedge R(x,z) = 2]\}.$$

The pattern is dramatically clear: the predicate has an existentially quantified variable, followed by a universally quantified variable bounded from below by the previous variable. However, the recursive function may assume one of infinitely many values (with acceptance being only one of them), whereas the L-predicate is three-valued, and the three values are utilized in a less trivial way. Determining more precisely how the constants 2 and 3 arise in this setting, and how they lead to different complexity measures like the above seems like a promising research direction.

6 The model in perspective

The increment model arose from an angle of study the author has called "access complexity." The author has presented several models resembling the increment access model (with more obvious proofs of equivalence) for NL, P, and NP [19]; they are briefly presented in Table 1. Each features a logspace machine with some type of access to a polynomial length reference. The reference is either written to nondeterministically once the input is given, or is initially blank.

One may think of computation universally as a mechanical search for an answer to a well-defined problem. We advocate a view of complexity that distills the *robustness* of classes from a *search mechanism*. For example, logspace Turing machines gave our models the power of reductions (and thus complete sets), and the polynomial reference with increment access built upon this foundation, prescribing how solutions may be found. This view lets us further isolate the components of complexity classes that are similar from those parts that distinguish the classes.

7 Conclusions

We examined a simple but general model of universal search, and found correlations between the amount of information shared between solution checks and the polynomial time hierarchy. We hope that this "brute-force search" presentation of pieces in the polynomial time hierarchy will lead to better understanding of the relations between these well-known classes. Note that our techniques are not overly dependent on the sizes of resources involved: by replacing the logspace machine with a polyspace one, and the polynomial reference with an exponential one, a similar hierarchy can be derived in relation to the exponential space hierarchy.

We end with several open problems that have arisen:

- What is $L + PI_k$, for $k = 1, 2$? Is $L + PI_1 = SF_2 = \oplus P//\mathrm{Opt}P$ (the class of languages recognized by $\oplus P$ (parity-P) with input-dependent advice from an Opt$P$ function), as proven by Ogihara [12]?

- Does this new interpretation give any insight into class separations? Fortnow showed that NL does not contain any non-constant level of the polynomial time hierarchy, and this result has been steadily improved by tighter simulations of determinism and nondeterminism by alternating machines [5, 6, 10]. Might we improve upon these results using the increment access model? Since a logspace machine is contained within our model, can we show that (under some conditions) increment access allows it to accept strictly more languages than before?

- Suppose we allow our model to have multiple references, each of which may incremented independently of the others. What happens then to the $L + PI_k$ hierarchy?

- Turing back to brute-force search (the original impetus for this model), recall that we only considered lexicographically ordered lists of the search space. Adding one to a polynomial length number is a very easy task (it is in AC^0, cf. [2]). What if a separate resource-bounded (e.g. logspace) machine generated this list instead? For any input x, the listing machine M would represent a bijection on $\{0,1\}^{p(|x|)}$, where $p(n)$ is a polynomial. When an increment state is reached, the current reference would be fed to M and the output would become the new reference. Would such a modification in "reference complexity" significantly change the complexity of the resulting classes?

8 Acknowledgements

The author wishes to thank his research advisor Juris Hartmanis for many enlightening discussions.

References

[1] D. Barrington. Bounded-width branching programs. Ph.D. Thesis, Department of Mathematics, M.I.T., May 1986.

[2] D. M. Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. Journal of Computer and System Sciences, 41(3):274–306, 1990.

[3] A. Chandra, D. Kozen, L. Stockmeyer. Alternation. Journal of the ACM, 28:114–133, 1981.

[4] J. Cai and M. Furst. PSPACE survives constant-width bottlenecks. International Journal of Found. of Comp. Sci., 2(1):67–76, 1991.

[5] L. Fortnow. Nondeterministic polynomial time versus nondeterministic logarithmic space: Time-space tradeoffs for satisfiability. In Proceedings of the 12th IEEE Conf. on Comp. Complexity, pp.52–60. IEEE, 1997.

[6] L. Fortnow and D. van Melkebeek. Time-space tradeoffs for satisfiability. In Proceedings of the 15th IEEE Conf. on Comp. Complexity, pp.2–13, 2000.

[7] U.Hertrampf, C.Lautemann, T.Schwentick, H.Vollmer, K.W.Wagner. On the Power of Polynomial Bit-Reductions. Proceedings of IEEE Structure in Complexity Theory, pp.200-207, 1993.

[8] N. Immerman. Descriptive Complexity. Springer-Verlag, New York, NY, 1999.

[9] M. Krentel. The complexity of optimization problems. Journal of Comp. and Sys. Sciences, 36(3):490–509, 1988.

[10] R. Lipton and A. Viglas. On the Complexity of SAT. In 40th Annual Symp. on Found. of Comp. Sci. (FOCS), pp.459–464, 1999.

[11] S. Miyano. Δ_2^p-complete lexicographically first maximal subgraph problems. Theor. Comp. Sci., 88(1):33–57, 1991.

[12] M. Ogihara. On serializable languages. International Journal of Found. of Comp. Sci., 5(3–4):303–318, 1994.

[13] M. Ogiwara and O. Watanabe. On polynomial time bounded truth-table reducibility of NP sets to sparse sets. SIAM J. Comput., 20(3):471–483, 1991.

[14] C. Papadimitriou. The complexity of unique solutions. Journal of the ACM 31, 492–500, 1984.

[15] J.R. Schoenfield. On degrees of unsolvability. Ann. Math. 69, 644-53, 1959.

[16] L. J. Stockmeyer. The polynomial-time hierarchy. Theor. Comp. Sci., 3(1):1–22, 1976.

[17] B. Trakhtenbrot. A survey of Russian approaches to perebor (brute-force search) algorithms. Ann. History of Computing, 6: 384–400, 1984.

[18] K. Wagner. More complicated questions about maxima and minima, and some closures of NP. Theor. Comp. Sci., 51:53–80, 1987.

[19] R. Williams. Access Complexity. Honors thesis in mathematics (unpublished manuscript), Cornell University, Ithaca, NY, 2001. URL: `http://www.people.cornell.edu/pages/rrw9/thesis.pdf`.