Lower Bounds for Accessing Information on Pure Pointer Machines

Brian Cloteaux[1] **and Desh Ranjan**[2]
[1]National Institute of Standards and Technology, Gaithersburg, MD, USA
[2]Department of Computer Science, New Mexico State University, Las Cruces, NM, USA

Abstract— *We study the complexity of accessing bits on a Pure Pointer Machine (PPM) or a pointer machine without arithmetic capabilities. In particular, we show that lower bounds in access time for information retrieval on a PPM arise from two independent factors: the complexity of information being stored and the amount of antisymmetry in the information. This result contrasts with earlier work by Ben-Amram and Galil that showed that for pointer machines with arithmetic capabilities look-up time depends only on the complexity of the information stored. We then demonstrate the use of these bounds to show optimal look-up times for comparing elements in partial and total orders on a PPM.*

Keywords: pointer machines, lower bounds

1. Introduction

In this paper we study the complexity of accessing information on *Pure Pointer Machines* (PPM) [1] – pointer machines with no arithmetic capabilities. Essentially, this problem is to find the time needed to retrieve a bit for a given set of indices.

This problem has been studied by Ben-Amram and Galil [2] for a stronger class of pointer machines that do have arithmetic capabilities. Their result showed that for these models, lower bounds on retrieval of information depend only on the complexity of information being stored. They extended their result to show that this bound holds even if we allow the data fields in the pointer machines to hold infinite precision numbers.

Our investigation examines pointer machines that have no data fields or arithmetic capabilities. We show that when these capabilities are removed from the pointer machine model, we see an additional restriction on look-up times based on the limited number of fixed radius neighborhoods we can construct.

The structure of this paper is to first introduce Pure Pointer Machines. We then give a bound based strictly based on the complexity of the information stored. We prove that for a structure on a PPM with I indices and holding C bits of information, where C is the Kolmogorov complexity of the information being stored, then there exists a bit that requires $\Omega(\lg(C/I))$ pointer dereferencings to retrieve. We then apply this result to prove that to compare two elements in an n element partial order on a PPM requires $\Theta(\lg n)$ time.

Finally, we show a bound that results from the amount of antisymmetry in the information stored. We define a metric on the information being stored. This metric ω_L is the clique number of a structure that we call the limiting graph. This limiting graph is constructed from the information being stored and from it we show a lower bound of $\Omega(\lg\lg\omega_L)$ for accessing bits. This bound is independent of the complexity of the information being stored. Lastly, we use this result to show that to compare two elements in a n element total order on a PPM requires $\Omega(\lg\lg n)$ steps.

2. Pure Pointer Machines

We first must define what model of computation we are considering when we talk about *pointer machines*. The term "pointer machine" has become overloaded throughout the years, and Ben-Amram [3] gives a good overview of several of the different definitions. The version of the pointer machine we are investigating was defined by Robert Tarjan in his paper [1] for establishing lower bounds for the Union Find problem. As Ben-Amram points out, Tarjan's versions of pointer machines are actually classes of algorithms and not true computational models.

Tarjan's version of a pointer machine (PM) is defined as a finite but expandable collection R of records, called *global memory*, and a finite collection of registers. Each record is uniquely identified by a label we call an *address* and is a collection of k fields with each field being either a data field or a pointer field. A pointer field can either specify a record or contain a *nil* pointer that is a special label used to designate an invalid address. All records are identical in structure.

A program on a pointer machine is finite set of instructions that allow copying of pointers between registers and/or record pointer fields, copying and applying of operators on data fields, creating of new records, and halting the computation. The only control instruction is the conditional jump based on a comparison on the values of two registers or record fields. The only comparison test allowed between pointer fields is testing for equality. No arithmetic operations are allowed on pointers. Tarjan defined the set of pointer machines that contain no data fields, only pointer fields, as *Pure Pointer Machines* (PPM). A study of the differences in computational power between types of pointer algorithms was given by Cloteaux and Ranjan [4].

An assumption that is made throughout this paper is that every PPM has records that contain exactly two pointer

fields. Justification for this assumption comes from Luginbuhl and Loiu (Lemma 5.4 [5]) who give a simple transformation from any PPM whose records contain k pointer fields to a PPM with records containing two fields that does not asymptotically change the run time.

Even though the Random Access Machine (RAM) is the most commonly used model in the study of algorithm complexity, the pointer machine has received attention as an alternative model. This is shown by the number of lower bound results using the model since Tarjan's original paper [6], [7], [8], [9], [10], [11], [12]. There are two reasons for this increase in popularity. Because of its simplicity, the pointer machine model is often more suitable for the analysis of lower bounds of time complexity. Pure pointer machines have the additional advantage of making explicit the time costs for all arithmetic operations. This contrasts with the RAM model that can hide the actual costs by allowing operations on numbers up to $\lg n$ bits (where n is the input size) in constant time.

3. Bounds Based on Information Complexity

A fundamental difference between PMs and PPMs is how they are able to address array representations. Since PPMs have no intrinsic way to represent integers, any type of query must be in terms of pointers to records rather than integers. We think of the array as represented on a PPM by a structure of records D. An n dimensional array access on a PPM would be given by a query function $Q : I^n \to \{0, 1\}$ where $I \subseteq D$. The set I is called an *index set*.

An assumption we make about index sets is that the records in them are in some sort of total ordering. This ordering allows us to iterate through the values stored in the array. We make this assumption without affecting time bounds on access times since if a query algorithm can run in time t, putting the records of the index set in a linked list does not affect the run time of the algorithm. The additional links can simply be ignored.

From this formulation of the problem, we quickly see that there must be queries for array structures on PPMs that require nonconstant time. For instance, consider the problem of trying to create a query algorithm to represent an arbitrary two dimensional array. For a $m \times n$ array, we will use as our index set I the union of two sets of records; the first one M where $|M| = m$ represents the first dimension of the array and a second N where $|N| = n$ represents the second dimension. Thus a query would be of the form $Q(i, j)$ where $i \in M$ and $j \in N$. To show that there is no constant time algorithm for accessing information in a two dimensional array, we can use a simple incompressibility argument. Let us store a string S in our array whose Kolmogorov complexity is $m \cdot n$. If we are able to answer every query in constant time starting from the set I, then the size of our data structure is at most $c(m+n)$ records for some constant c.

The contradiction comes from the fact that we can encode any structure on a PPM with r records with a $2r \lg r$ bit string. Every pointer field in a record can be encoded with $\lceil \lg r \rceil$ bits. By ordering the records, we represent the structure uniquely with a string of length $r \cdot 2 \cdot \lg \lceil r \rceil$ bits. Thus for $c(m + n)$ records, we can encode it with $O((m + n) \lg(m + n))$ size string. For large enough m and n we quickly get a contradiction in the minimal number of bits needed to encode S.

More generally, we can extend this idea to show a relationship exists between the complexity of the information being stored and the size of the index set we use to retrieve individual bits.

Theorem 3.1: Suppose on a PPM, we store a string S whose Kolmogorov complexity is $K(S)$. For the index set I and the query function $Q : I^n \to \{1, 0\}$, if $K(S) > |I| \lg^c |I|$ where $c > 1$ and $|I|$ is large enough then there exists a query that requires dereferencing $\Omega\left(\lg \frac{K(S)}{|I|}\right)$ pointers.

Proof: Let us define $d = (c - 1)/2c$. Since $c > 1$ then $d > 0$. For a contradiction, let us assume that the number of records we need to dereference to retrieve any bit in S is less than $d \lg \frac{K(S)}{|I|}$ for any size of $|I|$. From this assumption, the number of records in the structure storing the string must be less than

$$|I| 2^{d \lg \frac{K(S)}{|I|}} = K(S)^d |I|^{1-d} \tag{1}$$

Using the string encoding in the example above, the number of bits needed to represent this structure is at most $K(S)^d |I|^{1-d} (d \lg K(S) + (1 - d) \lg |I|)$.

For an algorithm of size $\mathcal{A}$ that can retrieve the bits of S in order from the structure, there exists a constant $\mathcal{C} \geq \mathcal{A}$ such that

$$K(S) \leq K(S)^d |I|^{1-d} (d \lg K(S) + (1 - d) \lg |I|) + \mathcal{C} \tag{2}$$

rearranging the terms we get

$$\left(\frac{K(S)}{|I|}\right)^{1-d} \leq d \lg K(S) + (1 - d) \lg |I| + \mathcal{C} \tag{3}$$

which simplifies to

$$\left(\frac{K(S)}{|I|}\right)^{1-d} - d \lg \frac{K(S)}{|I|} \leq \lg |I| + \mathcal{C} \tag{4}$$

From $K(S) \geq |I| \lg^c |I|$

$$\lg^{c(1-d)} |I| - cd \lg \lg |I| \leq \lg |I| + \mathcal{C} \tag{5}$$

Since it is easy to show that $c(1 - d) > 1$, then for $|I|$ large enough we get a contradiction. Thus a query must exist that dereferences at least $d \lg \frac{K(S)}{|I|}$ records to return an answer. □

For an example of how to apply this bound, consider the problem of maintaining a partial order ($\mathcal{PO}$) of n elements on a PPM. In this case, the index set would be the set of records that represent elements in the partial order. The stored string holds bits that represent whether the $\leq$ relation holds between elements a and b. In other words,

$$Q(a,b) = \begin{cases} 1 & \text{if } a \leq b, \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Thus we get the following result.

Theorem 3.2: For the $\mathcal{PO}$ problem on a PPM, the time needed to compare two elements out of n elements is $\Omega(\lg n)$.

Proof: The amount of information needed to store an arbitrary n element partial order was proved by Kleitman and Rothschild [13] to be $n^2/4+o(n^2)$. Since $O(n^2)/n = O(n)$, then we can apply Theorem 3.1 that gives us a lower bound for the $\mathcal{PO}$ problem of $\Omega(\lg n)$. $\square$

Using a balanced binary tree to store the information, we see that this bound is optimal.

4. Bounds Arising from Antisymmetry

For many problems, the information bound from Theorem 3.1 does not apply. This occurs when the complexity of the information is not large enough compared to the number of indices. For example, consider the problem of implementing the $\leq$ operation between n elements in a total order. The amount of information needed to represent a total order of n elements is $\lg n! = O(n \lg n)$. Thus the ratio $K(S)/|I| = O(n \lg n)/n = O(\lg n)$ for all index sets does not allow Theorem 3.1 to apply to it.

Studying the total order problem allows us to see a fundamental difference in bounds between PMs and PPMs. A PM is able to implement a total ordering with constant time comparison by storing unique values in the data fields of its records. However, for a PPM we will show that this problem is not constant. The reason for this difference is that there are a limited number of unique graph structures with some fixed radius to encode information on a PPM. We show this difference by extending an idea originally presented by Ranjan, Pontelli, Gupta, and Longpré [12]; that is to look at the number of different neighborhoods possible around the elements in the index set and compare that to the number that we need to answer a query.

To see this relationship, let us first give a definition. For a query function $Q : I^2 \to \{0,1\}$, we define its *limiting graph* $L(Q) = (V, E)$ in the following manner: the set of nodes V are the index set I, i.e. $V = I$, and an edge (v_1, v_2) is in edge set E if and only if $v_1, v_2 \in I$ such that $Q(v_1, v_2) \neq Q(v_2, v_1)$. In other words, if we can swap the two indices in an instance of Q and get a different answer, then those two indices produce an edge in the limiting graph. We show that there is a basic relation between the query time needed for function Q on a PPM and the clique number ω of the limiting graph of Q where $\omega(L(Q)) = d$ where d is the greatest integer such that $K^d \subseteq L(Q)$. This bound results from the fact that we need to have enough unique neighborhoods to distinguish every possible query between elements in the maximum clique of the limiting graph.

To see this, let us first define what it means to be for two neighborhood to be indistinguishable from each other. We denote the c radius neighborhood centered on a record a as $N_c(a)$. The idea of examining the c radius neighborhoods around each member of the index set is formalized with the following definition.

Definition 4.1: We define the c-neighborhoods around two records a and b as being **indistinguishable** to each other if they have the following properties.

1) There exists a bijective mapping p from the records of $N_c(a)$ to the records of $N_c(b)$.
2) The function p maps a to b ($p(a) = b$).
3) The mapping respects the labeling of the edges. In other words, $N_c(x)$ has an edge (x, y) labeled by l if and only if $N_c(y)$ has an edge $(p(x), p(y))$ labeled by l. On a PPM, a label i corresponds to the pointer being stored in the ith pointer field in a record.
4) If $r \in N_c(x) \cap N_c(y)$, then $p(r) = r$. This implies that the records shared by both neighborhoods are in the same position in both neighborhoods.

If two neighborhoods are not indistinguishable to each other, then the neighborhoods are said to be *distinguishable*. From this definition, if we have two index records a and b whose c neighborhoods $N_c(a)$ and $N_c(b)$ are indistinguishable, then in c or less pointer dereferencings $Q(a,b)$ and $Q(b,a)$ will always return the same answer. This follows from the fact that the only comparison allowed on a PPM is pointer equality. Because any common records between the two indistinguishable neighborhoods are forced to occupy the same relative position from the center, all pointer comparisons will always return the same answer no matter the order of the neighborhoods. Thus if $N_c(a)$ and $N_c(b)$ are indistinguishable and $Q(a,b) \neq Q(b,a)$ then the algorithm cannot resolve the query in c or less steps. We now show that these indistinguishable neighborhoods implies lower bounds on functions Q in general with the following result.

Theorem 4.1: On a PPM, for the query function $Q : I^2 \to \{0,1\}$ and its limiting graph $L(Q)$, there exists a query that requires dereferencing $\Omega\left(\lg \lg \omega\left(L(Q)\right)\right)$ pointers.

Proof: For a contradiction, we assume that every query call $Q(I^2)$ can be resolved in less than $b = \frac{1}{3} \lg \lg \omega(L)$ record inspections. We note that the number of records in a b-neighborhood must be $r \leq 2^{(1/3) \lg \lg \omega(L)} = \left(\lg \omega(L)\right)^{1/3}$.

The main idea of this proof is to show that there are a limited number of distinguishable neighborhoods with radius b. We then show that this number is less than $\omega(L(Q))$, thus forcing us to have a query where two index records

have indistinguishable b-neighborhoods, but give different answers depending on the order of the records.

To show that we must have indistinguishable b-neighborhoods, we use the Erdős-Rado sunflower lemma [14]. This lemma states that if we have a collection of nonempty sets $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_M\}$ whose maximum cardinality is m and where $M = (p-1)^m m!$, then there must exist a set $\mathcal{P} \subseteq \mathcal{S}$ (called a *sunflower* where each member set is called a *petal*) with at least p members such that each two members of $\mathcal{S}_i, \mathcal{S}_j \in \mathcal{P}$ contains the same common intersection $\mathcal{S}_i \cap \mathcal{S}_j = \mathcal{C}$ (called the *core*) for every $\mathcal{S}_i, \mathcal{S}_j \in \mathcal{P}$ where $\mathcal{S}_i \neq \mathcal{S}_j$. At the same time, the noncore elements are disjoint from all other sets in the sunflower, (i.e. $(\mathcal{S}_i - \mathcal{C}) \cap \mathcal{S}_j = \emptyset$ for all $\mathcal{S}_i, \mathcal{S}_j \in P$ where $\mathcal{S}_i \neq \mathcal{S}_j$).

Consider the collection of the b neighborhoods around a set of elements $W = \{w_1, w_2, ..., w_q\}$ that forms a maximal clique in $L(Q)$. Thus $|W| = \omega(L(Q))$. For the set of neighborhoods around the elements in W, $\{N_b(w_1), N_b(w_2), ..., N_b(w_q)\}$, the sunflower lemma tells us that this set must contain a sunflower with $p = |W|^{1/2r}$ petals, since $\left(|W|^{1/2r} - 1\right)^r r! \leq |W|$.

We want to show that the number of distinguishable graphs with radius b in the sunflower is less than the number of petals. We accomplish this by using a counting argument originally from Ranjan et al. [12] that encodes each possible b neighborhood in the sunflower as a string.

This encoding assumes that the core contains m elements. To represent a neighborhood, we use a vertex list. Each vertex is identified by an integer from 0 to m where 0 represents a vertex not in the core and the other integers identify core elements. An additional bit marks the center record for the neighborhood. The edges are a represented by a list of size at most $2m$ that contains pairs of vertexes. Since $m < r$, the total representation uses at most $2r \cdot (2 \lg r) + r < 5r \lg r$ bits. Now we can compare the number of strings versus the size of the petal set.

$$2^{5r \lg r} < \omega(L(Q))^{\frac{1}{2r}} \tag{7}$$

or equivalently,

$$2^{10r^2 \lg r} < \omega(L(Q)) \tag{8}$$

taking the logarithm of both sides we get

$$10 \cdot r^2 \lg r < \lg \omega(L(Q)) \tag{9}$$

finally substituting for r we get

$$\frac{10}{3} \cdot (\lg \omega(L(Q)))^{\frac{2}{3}} \cdot \lg \lg \omega(L(Q)) < \lg \omega(L(Q)) \tag{10}$$

This is always true for $|I|$ large enough, thus forcing the existence of query where the indices have indistinguishable b neighborhoods but the function gives different answers depending on the order of the indices. To resolve this contradiction, the bound must hold. □

Let us now reconsider the problem of lower bounds for a total order ($\mathcal{TO}$) on a PPM. Using this theorem, we have a simple application to get a nontrivial lower bound for the comparison operation in a total ordering.

Theorem 4.2: On a PPM, the time needed to compare two elements in an n element total order is $\Omega(\lg \lg n)$.

Proof: This theorem follows immediately from the antisymmetry property of total orderings. Because of this property, the limiting graph ends up being the n clique. From Theorem 4.1 the query time must be $\Omega(\lg \lg n)$. □

There are several efficient algorithms for the temporal precedence problem [15], [12], [16], [17] that can be applied to this problem, and show that the given lower bound is optimal.

5. Conclusions

In this paper we have investigated the time needed for information retrieval on a Pure Pointer Machine. We have given two fundamental bounds, one based on the complexity of information stored and the other based on the amount of antisymmetry in the information. This contrasts with the bound given by Ben-Amram and Galil for pointer machines with arithmetic capability that depends only on the complexity of the information. Finally, we showed how to use these bounds to get optimal time bounds for the comparison operator in total and partial orderings on PPMs.

6. Acknowledgments

This work was done while the first author was at New Mexico State University and was partially supported by a Department of Education, Graduate Assistants in Areas of National Need (GAANN) grant.

References

[1] R. E. Tarjan, "A class of algorithms which require nonlinear time to maintain disjoint sets," *Journal of Computer and System Sciences*, vol. 77, pp. 110–127, 1979.

[2] A. M. Ben-Amram and Z. Galil, "On pointers versus addresses," *Journal of the ACM*, vol. 39, no. 3, pp. 617–648, 1992.

[3] A. M. Ben-Amram, "What is a "pointer machine"?" *SIGACT News*, vol. 26, no. 2, pp. 88–95, 1995.

[4] B. Cloteaux and D. Ranjan, "Some separation results between classes of pointer algorithms," in *DCFS '06: Proceedings of the eighth annual workshop on Descriptional Complexity of Formal Systems*, 2006, pp. 232–240.

[5] D. Luginbuhl and M. Loui, "Hierarchies and space measures for pointer machines," *Information and Computation*, vol. 104, pp. 253–270, 1993.

[6] N. Blum, "On the single-operation worst-case time complexity of the disjoint set union problem," *SIAM Journal on Computing*, vol. 15, no. 4, pp. 1021–1024, 1986.

[7] K. Mehlhorn, S. Näher, and H. Alt, "A lower bound of the complexity of the union-split-find problem," *SIAM Journal of Computing*, vol. 17, no. 6, pp. 1093–1102, 1988.

[8] H. La Poutré, "Lower bounds for the union-find and the split-find problem on pointer machines," *Journal of Computer System Science*, vol. 52, no. 1, pp. 87–99, 1996.

[9] B. Chazelle and B. Rosenburg, "Simplex range reporting on a pointer machine," *Computational Geometry: Theory and Applications*, vol. 5, pp. 237–247, 1996.

[10] D. Ranjan, E. Pontelli, and G. Gupta, "On the complexity of parallel implementation of logic programs," in *Proceedings of the 17th Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer-Verlag, 1997, pp. 123–137, lNCS 1346.
[11] E. Pontelli, D. Ranjan, and G. Gupta, "Complexity analysis of late binding in dynamic object-oriented languages," *Journal of Functional and Logic Programming*, vol. 1999, no. Special Issue 2, 1999.
[12] D. Ranjan, E. Pontelli, G. Gupta, and L. Longpré, "The temporal precedence problem," *Algorithmica*, vol. 28, no. 3, pp. 288–306, 2000.
[13] D. Kleitman and B. Rothschild, "The number of finite topologies," *Proceedings of the American Mathematical Society*, vol. 25, no. 2, pp. 276–282, June 1970.
[14] P. Erdős and R. Rado, "Intersection theorems for systems of sets," *Journal of the London Mathematical Society*, vol. 35, pp. 85–90, 1960.
[15] D. Ranjan, E. Pontelli, and G. Gupta, "Efficient algorithms for the temporal precedence problem," *Information Processing Letters*, vol. 68, no. 2, pp. 71–78, 1998.
[16] G. S. Brodal, C. Makris, S. Sioutas, A. Tsakalidis, and K. Tsichlas, "Optimal solutions for the temporal precedence problem," *Algorithmica*, vol. 33, no. 4, pp. 494–510, 2002.
[17] E. Pontelli and D. Ranjan, "A simple optimal solution for the temporal precedence problem on pure pointer machines," *Theory of Computing Systems*, vol. 38, no. 1, pp. 115–130, 2005.