Algorithmica (1996) 16: 181–214

Algorithmica

Blocking for External Graph Searching

M. H. Nodine,[1] M. T. Goodrich,[2] and J. S. Vitter[3]

Abstract. In this paper we consider the problem of using disk blocks efficiently in searching graphs that are too large to fit in internal memory. Our model allows a vertex to be represented any number of times on the disk in order to take advantage of redundancy. We give matching upper and lower bounds for complete d-ary trees and d-dimensional grid graphs, as well as for classes of general graphs that intuitively speaking have a close to uniform number of neighbors around each vertex. We also show that, for the special case of grid graphs blocked with isothetic hypercubes, there is a provably better speed-up if even a small amount of redundancy is permitted.

Key Words. External searching, Isothetic hypercubes, Blocking, Input/output complexity, Redundancy.

1. Introduction. External searching is a fundamental topic in computer science and databases [1]. By "external" we mean that the records to be searched cannot fit simultaneously in the internal memory. In this paper we consider searching in graphs. We assume a process that generates a path through a graph, and it is necessary to read a vertex into memory when it is visited in order to decide which vertex to visit next. We assume that internal memory is capable of holding M vertices, and that data are read from the secondary storage in blocks of B vertices, where of course $B \leq M$. The goal is to minimize the total number of block transfers.

Our main unit of measure is the blocking speed-up $\sigma(B)$, which can intuitively be described as the worst-case number of vertices that can be traversed before causing a page fault. We show that optimal blocking speed-ups can be achieved for many graphs as long as some degree of replication of data is present. We study this problem under a wide variety of assumptions concerning replication of data and structure of the underlying graph. For the case of grid graphs blocked using isothetic hypercubes, we show that with no duplication of data, the optimal blocking speed-up is provably worse than when duplication is allowed.

[1] Motorola Inc., 5918 W. Courtyard Dr., Suite 200, Austin, TX 78746, USA. Support was provided in part by an IBM Graduate Fellowship, by NSF Research Grants CCR-9007851 and IRI-9116451, and by Army Research Office Grant DAAL03-91-G-0035.

[2] Department of Computer Science, The Johns Hopkins University, Baltimore, MD 21218, USA. Support was provided in part by NSF Grants CCR-9003299, CCR-9300079, and IRI-9116843, and by NSF/DARPA Grant CCR-8908092.

[3] Department of Computer Science, Duke University, Durham, NC 27708-0129, USA. Support was provided in part by a National Science Foundation Presidential Young Investigator Award CCR-9047466 with matching funds from IBM, by NSF Research Grant CCR-9007851, and by Army Research Office Grant DAAL03-91-G-0035.

Received August 30, 1993; revised May 26, 1994. Communicated by D. P. Dobkin.

An algorithm for achieving a blocking speed-up has two parts:

1. An assignment of vertices of the graph to blocks, known as the *blocking*. It is here that storage blow-up is a consideration.
2. A *paging algorithm* to specify what vertices/blocks are in memory.

We are expanding on the research area of physical database design (efficient data storage), where in our case we have some knowledge (in the form of a graph) of the possible data accesses. We do not, however, know the sequence of accesses in advance. There are many applications for using blocks efficiently in external graph searching. These include A.I. searching in constraint networks, robot motion planning, the simulation of large deterministic finite automata (DFA), browsing in hypertext applications, accesses in object-oriented databases, and some matrix algorithms such as searching in monotone arrays [2]. Sometimes the graphs are well structured, as in the case of matrix searches and robot motion planning in a space discretized in a grid (which gives rise to grid graphs). At other times the graphs are unstructured, as in A.I. searching and DFA simulation. For the purposes of this paper, we assume that all graphs are undirected, an assumption that may not hold for certain applications such as hypertext and object-oriented databases.

One important assumption of our model is that any datum may be represented in multiple blocks. This assumption is stronger than that used, for example, by external B-trees, where a record of data may be pointed to from several places, although the data (exclusive of the key being indexed) exist in only one block [3]. We allow actual replication of the data and define the storage blow-up s to be the ratio between the actual number of blocks used to represent the data and the minimum number that could be used.

We know of no previous work for this problem, but there is related work. Borodin *et al.* considered the similar problem of traversing an access graph, but they considered only block size $B = 1$ and worked on developing an efficient paging algorithm that minimized the competitive ratio [4]. Alternatively, all the work done in the database community on B-trees could be viewed as a solution to our problem for complete trees with $s = 1$. Ullman and Yannakakis considered the I/O complexity of the transitive closure problem [5] with block size $B = 1$. Their work differs from this in that they do not explicitly consider blocking, and that their lower bounds only need to consider the graph structure, and not an adversary trying to generate a path that will cause many page faults.

There is considerable previous work on the embedding of one type of data structure into another [6]–[12]. This work is relevant because if we view the data structures as graphs, the goal of the embedding is to maintain locality in the original graph when accesses are done according to the target graph. If the data structures can be stored so that local references in the structure correspond to local references in storage, then it is hoped that such locality of reference would lead to fewer page faults. Of course, this is only a heuristic, and local references in the target graph do not always get translated into local references in the host. For example, Rosenberg considered the problem of preserving proximity in arrays [6] when mapping onto a linear access structure. A grid graph describes the proximity properties of arrays, so the embedding problem here is to map a grid graph into a semi-infinite number line. An interesting hypothesis is that blocking the data elements that are mapped into consecutive sets of B numbers on the number line would give a good performance for graph searching in a grid graph. Rosenberg showed

however that there is no linear mapping scheme that preserves proximity globally in arrays that are extendible in more than one dimension. DeMillo *et al.* considered the same problem and showed that proximity could be maintained if the storage mechanism was a binary tree [7]. Both concluded that the standard row-major (or column-major) way of storing arrays was asymptotically optimal for preserving locality in arrays. We find, however, that our earlier hypothesis concerning blocking does not hold even for finite arrays, as long as the array structure is much larger than the memory size. The embedding literature also fails to take into account the fact that data items may be multiply represented. An alternative approach is taken by Lipton *et al.*, who consider hierarchies of embeddings of graphs in which a given vertex can be replicated up to S times in going from one level of the hierarchy to the next, as well as a factor of T expansion in the distance between vertices [13]. Their paper, however, focuses on lower bounds for general graphs, and does not deal with the issue of how to find optimal embeddings. It is also not clear what the relevance of any optimal embeddings is to blocking.

In Section 2 we state the specifics of the model we are using for blocking in external graph searching. Section 3 gives some general results on paging strategies. Section 4 gives upper and lower bounds for searching in general graphs. Sections 5 and 6 consider upper and lower bounds for the blocking efficiency of searching in trees and grid graphs, respectively. Finally, Table 1 in Section 7 gives a summary of the results.

2. Model. We summarize the key assumptions of the model here, for the graph $G = (V, E)$.

1. The data are associated only with the vertices of the graph. Each vertex can be represented with a fixed amount of space.
2. Data for the vertices are stored in blocks, each of which can hold data for up to B vertices.
3. Data for a single vertex may be redundantly present in more than one block. Only one such block needs to be present in memory to satisfy a request.
4. The blocking algorithm must decide which data to store in which block before the search occurs, having no preknowledge of the search path.
5. The internal memory can hold data for at most M vertices, after which B elements will have to be flushed from memory to make room for a new block. In the weak model those B elements must comprise a single block; in the strong model any B elements may be flushed from the memory.
6. The total number of vertices in the graph is $n = \rho M$, where $\rho \gg 1$.
7. We will traverse a path of length L, where L could exceed n.

Assumption 3 is reasonable for searching (a read-only traversal in the graph), where the data for the vertices are not changed, but would not be a reasonable assumption if we needed to perform updates on the graph, since then all copies of a vertex would have to be present in memory to service the request. If the number of blocks of storage used to hold the graph is S, we define the storage blow-up $s = S/(n/B)$. Intuitively, s is the average number of blocks containing each vertex in the graph. A major effort of this research is to see what effect storage blow-up has on the efficiency of blocking algorithms, which in turn determines how much gain can be achieved by blocking for

a read-only application versus an application that is required to update the data structures.

If a vertex is present in internal memory at least once, we call the vertex *covered* (by analogy with graph pebbling); conversely those vertices present nowhere in memory are *uncovered*. We use the term *page fault* to refer to an event where the path we are tracing through a graph extends to an uncovered vertex, and for which the searching algorithm therefore needs to read at least one block from the secondary memory. If a path generates a page fault only every σ steps (on average), we say that it achieves a *blocking speed-up of* σ. We usually wish for σ to be an increasing function $\sigma(B)$. We refer to the vertex currently being traversed at any time as the *pathfront*.

When a page fault causes a block to be brought into memory, it may be the case that it is necessary for that block to overwrite data that are already in memory so as not to exceed the memory size. If the data are always flushed out by blocks, we say that we are using the *weak* paging strategy. A *strong* paging strategy would allow any B copies of vertices to be flushed, independent of whether they were originally in the same block. We count "copies of vertices" rather than the vertices themselves, since any vertex may be present multiple times in memory. It is the paging algorithm that determines whether we are in the strong or the weak model.

Notice that an upper bound on the blocking speed-up σ derives from theoretical considerations independent of the algorithm, and a lower bound on σ is an algorithm to achieve that speed-up. This is the reverse of the usual upper-bound/lower-bound situation.

All algorithms presented in this paper use the weak model. Conversely, our upper bounds all assume the strong model. It is conceivable that some of the difference between our upper and lower bounds could stem from the fact that these algorithms use a weaker model than the upper bounds.

It is not obvious given this model that blocking can ever be used efficiently. In fact, there is a graph for which an adversary can limit $\sigma(B) \leq 1$, independently of B. For example, consider K_{M+1}, the complete graph on $M + 1$ vertices, with the path starting on any vertex. At any subsequent point, there will be at least one vertex that is not in memory. Extending the path to any such vertex will force a page fault. It would seem that restricting the graphs to planar ones would prevent such a terrible case from occurring (at least for $M > 4$). However, in a graph in which some vertex v is connected to M other vertices, an adversary can cause a page fault at least every other step, giving $\sigma(B) \leq 2$; this graph is obviously planar.

3. Paging Strategies. We can distinguish between paging algorithms based on whether they are on-line or off-line. An *off-line* paging strategy may consider the entire path before deciding what blocks need to be brought in. An *on-line* paging strategy must base its decision on what block to bring in at any point only upon the previous history of the path (indeed it suffices to know only the pathfront). Note that in both cases, however, the blocks are fixed *a priori* without knowledge of the path. Permitting a paging strategy to be off-line and to have an unlimited storage blow-up allows for very efficient blocking.

LEMMA 1. *There is a blocking and an off-line paging strategy that always achieves a speed-up of at least B, even if $B = M$.*

PROOF. Let the blocking be $\mathcal{B} = \{\text{vertices of all paths of length } B - 1\}$. Clearly, the storage blow-up is large for this blocking. The off-line algorithm is simple: at any page fault, pick a block that contains all the vertices belonging to the next $B - 1$ steps of the path. Thus, a page fault can occur at most every B steps. □

The interesting cases come, of course, with using an on-line paging strategy and with blockings using constant-degree storage blow-ups. We call a paging algorithm *lazy* if it only brings a block into memory that services an immediately preceding page fault. There is no distinction between an on-line and an off-line strategy if $s = 1$, since there is only one possible block to choose to satisfy the page fault. It is possible to show that we can restrict ourselves to lazy algorithms from the standpoint of proving upper bounds, at least if we are dealing in the weak model, as the following theorem shows.

THEOREM 1. *For any blocking $\mathcal{B} = \{B_i\}$, there is an optimal on-line weak paging strategy that is lazy.*

PROOF. We start with some algorithm A and for any path we convert it to one that uses a lazy paging strategy without increasing the number of blocks read. We do this conversion by eliminating any read that was not prompted by a page fault. Assume that algorithm A reads some block B in at time i that does not service a page fault. If B is flushed before anything is read from it, then we eliminate both the read and the flush from the algorithm. If some vertex is read from B before it is flushed, let the time at which the first such vertex is read be $j > i$. Algorithm A cannot have done a fault-prompted read at time j since the vertex was already in memory. Now move the read of block B from time i to time j. In either case the number of non-fault-prompted reads decreases by one without increasing the total number of reads. By applying this strategy inductively, all non-fault-prompted reads can be eliminated, yielding a lazy algorithm. □

It is possible that this theorem could be extended to the strong model by considering the multiset of vertices present in memory instead of the set of blocks and proceeding similarly. However, such an extension is not straightforward.

4. Searching in General Graphs. We here consider the problem of searching in general graphs. First, we give some definitions. Consider a connected graph $G = (V, E)$ such that $|V| = n = \rho M$, with $\rho > 1$.

DEFINITION 1. Let $N \subset V$. If $v \in N$, then we say that N is a *neighborhood* of v. Note that N is not necessarily connected. Let $\mathcal{N}_v = \{\text{all neighborhoods of } v\}$. If $N \in \mathcal{N}_v$ and $|N| = k$, then we say N is a *k-neighborhood* of v. Let $\mathcal{N}_v(k) = \{\text{all } k\text{-neighborhoods of } v\}$.

DEFINITION 2. Let $N \in \mathcal{N}_v$. Then $b(v, N)$, the *break-out distance* of v in N, is defined

to be the minimum distance from v to any point v' not in N, i.e.,

$$b(v, N) = \min_{v' \notin N} d(v, v'),$$

where $d(v, v')$ is the minimum path length from v to v'.

DEFINITION 3. We call any $N \in \mathcal{N}_v(k)$ for which $b(v, N)$ is a maximum over all k-neighborhoods of v a *compact k-neighborhood*. If N is a compact k-neighborhood of v, then we define $r_v(k) = b(v, N)$ to be the *k-radius of v*. We let $\mathcal{N}_v^*(k)$ denote the set of all compact k-neighborhoods of v.

We can show a simple lemma concerning compact neighborhoods.

LEMMA 2. *Assume we have a connected graph $G = (V, E)$. At least one compact k-neighborhood of vertex v is connected for any vertex $v \in V$.*

PROOF. Consider the sets

$$W = \{v' \in V \mid d(v, v') < r_v(k)\}$$

and

$$W' = \{v' \in V \mid d(v, v') \leq r_v(k)\}$$

and consider any $V' \in \mathcal{N}_v^*(k)$. By definition of $r_v(k)$, $|W| \leq k < |W'|$. Since V' is a compact k-neighborhood, it must be the case that $W \subseteq V'$. It can be shown by simple induction on k that W and W' are connected. So any set V' such that $W \subseteq V' \subset W'$ is connected, since $V' - W$ comprises only vertices connected to some vertex in W and W was connected. There is some such V' for which $|V'| = k$, which is a connected, compact k-neighborhood of v. □

DEFINITION 4. We define two k-radii for graphs: the *minimum k-radius* and the *maximum k-radius*, respectively, are

$$r^-(k) = \min_{v \in V} r_v(k)$$

and

$$r^+(k) = \max_{v \in V} r_v(k).$$

DEFINITION 5. Any class of graphs for which the ratio $r^+(k)/r^-(k)$ is bounded by a constant for some k is called a *k-uniform class of graphs*.

Clearly, for any particular graph, the ratio between the upper and lower k-radius is constant, so the uniformity definition applies only to infinite classes of graphs.

We start with a few examples of k-radii, followed by a few simple lemmas.

EXAMPLE 1. We can compute the k-radii of complete d-ary trees. It is simple to see that the root has $(d^{r+1} - 1)/(d - 1)$ vertices within a distance of r (assuming that the height of the tree is at least r). This fact leads to[4]

$$r_{\text{root}}(k) = \frac{\lg(k(d-1)+1)}{\lg d} - 1.$$

Similarly, a vertex that is at least a distance r from both the root and the leaves has radius

$$r_{\text{int}}(k) = \frac{\lg(k(d-1)+2) - \lg(d+1)}{\lg d}.$$

The leaves are a little harder to analyze, but each leaf of a tree with height at least r has

$$\frac{d^{\lfloor r/2 \rfloor + 1} + d^{\lceil r/2 \rceil} - 2}{d-1}$$

vertices within a distance r of it. This leads to a radius

$$r_{\text{leaf}}(k) = \min\left\{2\left\lceil \frac{\lg(k(d-1)+2) - 1}{\lg d} - \frac{1}{2}\right\rceil, 2\left\lceil \frac{\lg((k(d+1)+2)/d - 1) - 1}{\lg d}\right\rceil + 1\right\}.$$

Comparing these values, it follows that, for complete d-ary trees, $r^-(k) = r_{\text{int}}(k)$ and $r^+(k) = r_{\text{leaf}}(k)$. Those internal vertices that are within a distance r of the root have a number of vertices within a distance r of them that is intermediate between those of the internal vertices and the root; the radii are correspondingly intermediate when k becomes large enough. A similar fact holds for those internal vertices that are close to the leaves.

LEMMA 3. *The set of all complete d-ary trees is a uniform class of graphs.*

PROOF. The minimum and maximum radii are always within about a factor of 2. □

EXAMPLE 2. We can also compute the leading term of the radii for grid graphs. See Section 6 for a definition of grid graphs. We first note that all the vertices in a d-dimensional grid graph have the same radius $r_d(k)$ for any value k; consequently grid graphs comprise a uniform class of graphs. We start by computing the number of vertices $k_d(r)$ within a distance r of any vertex in a d-dimensional grid graph; the radii are found by inverting this function to get r as a function of k. The problem of computing how large the neighborhoods of radius r in d-dimensional grid graphs are was considered by Rosenberg [6], where he derived the answer $O(r^d)$. In the following derivation we find the exact coefficient of the r^d term.

It is immediately apparent that, in one dimension, there are $2r + 1$ vertices within a distance r of any vertex; in other words, $k_1(r) = 2r + 1$. We are only computing the

[4] We use the notation $\lg x$ to mean $\log_2 x$.

leading term, so we consider the functions $\tilde{k}_d(r)$ that are the same as $k_d(r)$ with all but the leading terms removed. Thus, $\tilde{k}_1(r) = 2r$. We observe that

$$k_d(r) = k_{d-1}(r) + 2 \sum_{0 \le r' < r} k_{d-1}(r')$$

or equivalently

$$\tilde{k}_d(r) = 2 \sum_{0 \le r' < r} \tilde{k}_{d-1}(r').$$

It is also clear that $\tilde{k}_d(r) = c_d r^d$ for some c_d. We can compute the recurrence for c_d as follows:

$$\begin{aligned} c_d &= \frac{2}{d} c_{d-1}, \\ c_1 &= 2 \end{aligned}$$

using the fact that

$$\sum_{0 \le i < n} i^d = \frac{1}{d+1} n^{d+1} + O(n^d).$$

We can solve the recurrence for c_d to show that

$$\tilde{k}_d(r) = \frac{2^d}{d!} r^d.$$

Inverting this formula, we arrive at our radius:

$$r_d(k) = \tfrac{1}{2} (d!\, k)^{1/d} + o(k^{1/d}).$$

We can use Stirling's formula to approximate the $d!$ term, giving the result[5]

$$r_d(k) \sim \frac{1}{2e} (2\pi d)^{1/2d}\, d k^{1/d}.$$

Finally, notice that $(2\pi d)^{1/2d}$ converges to 1 as $d \to \infty$ (and is never larger than about 2.5), so that

$$r_d(k) \sim \frac{1}{2e}\, d k^{1/d}.$$

Since all the vertices have the same radius, we get

$$r_d^+(k) = r_d^-(k) \sim \frac{1}{2e}\, d k^{1/d}. \tag{1}$$

[5] We use the notation $f(x) \sim g(x)$ to mean that $f(x) = g(x) + o(f(x))$; in other words, to say that $f(x)$ and $g(x)$ have identical leading terms, including coefficients.

LEMMA 4. *For any graph* $G = (V, E)$, *if* $k \leq k'$, *then*:

1. $r_v(k) \leq r_v(k')$ *for any* $v \in V$.
2. $r^+(k) \leq r^+(k')$.
3. $r^-(k) \leq r^-(k')$.

PROOF. Assume that $k \leq k'$.

(1) Let N be a compact k-neighborhood of v. Let N' be N plus any $(k' - k)$ other vertices. Then N' is a k'-neighborhood with radius at least $r_v(k)$.

(2) Let v be a vertex such that $r_v(k') = r^-(k')$. Then we have

$$r^-(k) \leq r_v(k) \leq r_v(k') = r^-(k').$$

(3) Let v be a vertex such that $r_v(k) = r^+(k)$. Then

$$r^+(k) = r_v(k) \leq r_v(k') \leq r^+(k'). \qquad \square$$

The next lemma states that if we increase a compact j-neighborhood of any vertex by adding k additional vertices, we cannot increase the radius by more than $2r^+(k)$.

LEMMA 5. *For any vertex* v, $r_v(j + k) \leq r_v(j) + 2r^+(k)$.

PROOF. Figure 1 illustrates this proof. Assume by way of contradiction that for some j, k, and v, we have $r_v(j + k) > r_v(j) + 2r^+(k)$. We can find two compact neighborhoods N_j and N_{j+k} consisting of j and $j + k$ vertices, respectively, with $N_j \subset N_{j+k}$. By definition of the radius, those vertices on the border of N_j (i.e., those vertices v' such that $b(v', N_j) = 1$) have $b(v', N_{j+k}) > 2r^+(k)+1$. Choose vertex v' such that $b(v', N_{j+k})$ is minimized and consider a path that realizes the break-out distance. This path has length more than $2r^+(k) + 1$. Consider the vertex w that is the midpoint of the path. Define $N = N_{j+k} - N_j$; N is a k-neighborhood of w. From vertex w, it takes more than $r^+(k)$ steps to reach any vertex in N_j; likewise it takes more than $r^+(k)$ steps to reach any vertex

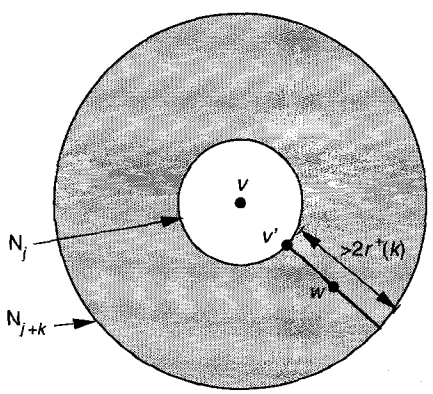


Fig. 1. Vertex w has a larger k-radius than $r^+(k)$.

in $G - N_{j+k}$. It follows that $b(w, N) > r^+(k)$, contradicting the definition of $r^+(k)$. Hence there can be no such j, k, and v. □

The previous lemma leads to the following important result.

LEMMA 6. *Assume that $k \leq k'$. Then*

$$r^+(k') \leq \left(2\frac{k'}{k} + 3\right) r^+(k).$$

PROOF. Let v be some vertex with radius $r_v(k') = r^+(k')$ and let $d = \lceil k'/k \rceil$. By Lemma 5, if we start from some compact k-neighborhood of v, we can increase the radius by at most $2r^+(k)$ for every additional k vertices we add to it. It follows that

$$r^+(k') = r_v(k') \leq r_v(k) + 2\,dr^+(k) \leq r^+(k) + 2\,dr^+(k) \leq \left(2\frac{k'}{k} + 3\right) r^+(k). \quad \square$$

4.1. *Upper Bounds.* Armed with the above definitions and lemmas, we are ready to prove some upper bounds. The first few upper bounds are trivial.

LEMMA 7. *The fastest worst-case speed-up that could be achieved in any graph is $\sigma \leq r^+(M)$.*

PROOF. Let the vertices in memory be some k-neighborhood of the pathfront just after a page fault has been serviced, for some $k \leq M$. By definition of $r^+(M)$, there is always some vertex within that distance of the pathfront that is not in memory, so it can take at most that many steps to cause the next page fault. Causing the next page fault leaves the same configuration as we assumed, so this process can be continued indefinitely. □

A second lemma is like this first one.

LEMMA 8. *The fastest worst-case speed-up that could be achieved in any graph is $\sigma \leq 2r^-(M)$.*

PROOF. Let v be a vertex such that $r_v(M) = r^-(M)$ (there must be at least one such vertex by the definition of minimum M-radius), and let

$$W = \{v' \in V \mid d(v, v') \leq r^-(M)\}.$$

By definition, $|W| > M$ and the distance between any two points $w, w' \in W$ is

$$d(w, w') \leq d(w, v) + d(v, w') \leq 2r^-(M).$$

Then as long as the pathfront is somewhere in W, it is always possible to extend the path to a vertex in W that is not in memory in fewer than $2r^-(M)$ steps, no matter which M vertices are in memory. We can begin at vertex v to establish the initial condition. □

These lemmas correctly handle the example of the vertex that is attached to M other vertices; $r^+(M) = 2$ in this case and, as we found before, $\sigma(B) \leq 2$ regardless of the block size. The difficulty with these two lemmas is that they are independent of the block size. Intuitively, it would seem that by using a block size of B, the fastest possible speed-up would be B or at least $O(B)$. The next lemma proves such a bound. First we need a definition.

DEFINITION 6. Let graph G be a tree. Then a *depth-first circuit* of the tree is a path that traverses every edge of the tree twice (once in each direction) and returns to the starting point.

LEMMA 9. *The fastest worst-case speed-up that could be achieved in a graph containing ρM vertices, $\rho > 1$, is*

$$\sigma \leq 2\frac{\rho}{\rho - 1}B.$$

PROOF. Assume that we are anywhere in the graph and there are at most M vertices in memory. We can walk through the entire graph using exactly $2\rho M$ steps by doing a depth-first circuit of some spanning tree; by definition, this tour returns us to the same node as we started. During this walk, we are guaranteed to cause at least $(\rho - 1)M/B$ page faults. □

This upper bound reduces to $2B$ as we let $\rho \to \infty$. In the special case of graphs containing a Hamiltonian cycle, we can clearly get an upper bound of B by having the adversary continually follow the Hamiltonian cycle.

We would like to prove an even stronger upper bound, specifically something that is proportional to $r^+(B) \leq B$. The next lemma gives a result along this line.

LEMMA 10. *The fastest worst-case speed-up that could be achieved for any graph is* $\sigma \leq (2(M/B) + 3)\, r^+(B)$.

PROOF. From Lemma 7, we know that $r^+(M)$ is an upper bound. However, from Lemma 6, $r^+(M) \leq (2(M/B) + 3)\, r^+(B)$. □

In particular, the previous lemma shows that we have a bound that is proportional to $r^+(B)$ as long as we use blocks that grow as a constant fraction of the memory size—in other words, as long as we have large blocks. As the block sizes become smaller, it becomes much more difficult to get an upper bound proportional to $r^+(B)$. The following lemma gives such a bound with some restrictions on the blocking. We later generalize the lemma to remove any such restrictions.

Before presenting the lemma, however, we want to make some observations about hypergraphs [14]. First, a hypergraph is a pair $G = (V, \mathcal{H})$, where V is a set of vertices and $\mathcal{H}$ is a set of subsets of V called *edges*. If each set $H \in \mathcal{H}$ has cardinality 2, then G is an ordinary graph. If $|H| = r$ for all $H \in \mathcal{H}$, then we say that the hypergraph is *r-regular*. Let $S \subset V$. Then if for every $H \in \mathcal{H}$ we have $|H \cap S| \leq 1$, then we say that S

is a *strongly stable set of vertices*. Notice that blocking the vertices of the graph in which we wish to search induces a B-regular hypergraph, where each of the blocks is an edge of the hypergraph. The next lemma says that we can get an upper bound proportional to $r^+(B)$ as long as we can find a strongly stable set of vertices of cardinality at least $\lceil n/B \rceil$.

LEMMA 11. *Assume that there are $\lceil n/B \rceil$ vertices such that there is no block containing any two of them. Then as $n/M \to \infty$, the blocking speed-up satisfies*

$$\sigma(B) \leq 8r^+(B).$$

PROOF. Define the *ball of radius r around vertex v* to be $\{v' \in V \mid d(v, v') \leq r\}$, denoted by $K_v(r)$. Define a *close packing* of balls of radius r in a graph to be a packing of balls of radius r such that each ball is touching at least one other ball (assuming there is more than one ball in the packing) and no vertex is contained in more than one ball. Two balls are touching if the distance between their centers is $2r$. Choose any maximal close packing of balls of radius $r^+(B)$ in the graph. Connect the centers of any touching balls together via paths of length $2r^+(B)$ and create a spanning tree of the resulting subgraph, as shown in Figure 2. This spanning tree is called the *skeletal Steiner tree*. Since each ball of radius $r^+(B)$ by definition contains at least B vertices of the graph, there are at most $\lfloor n/B \rfloor$ balls in the maximal close packing. Hence, the total length of this skeletal Steiner tree is no more than $2r^+(B)(\lfloor n/B \rfloor - 1)$.

We claim that every vertex in the graph is within a distance $2r^+(B)$ of the skeletal Steiner tree. Assume that some point exists whose distance to the skeletal Steiner tree is greater than $2r^+(B)$. Then it must likewise have a distance at least $2r^+(B)$ from the centers of each of the balls, and consequently some vertex exists with distance exactly $2r^+(B)$ from some ball whose ball of radius $r^+(B)$ does not overlap any of the other balls, contradicting the assumption that the close packing was maximal.

Now choose $\lceil n/B \rceil$ vertices with no block containing two of them (a strongly stable set of vertices). We call these vertices the "must-visit" vertices, since, by visiting all of them, we guarantee that we cause at least $\lceil n/B \rceil - M$ page faults. This set of strongly stable vertices was assumed to exist in the lemma. Attach each of these vertices to the

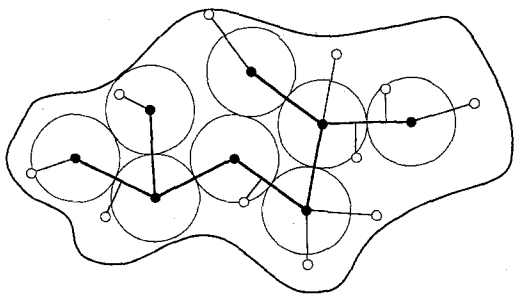


Fig. 2. Creating a spanning tree visiting all the "must-visit" vertices. ● Centers of close-packing balls. ○ "Must-visit" vertices.

skeletal tree using a path whose length is no more than $2r^+(B)$ to form an augmented Steiner tree of total length no more than

$$2r^+(B)\left\lceil \frac{n}{B} \right\rceil + 2r^+(B)\left(\left\lfloor \frac{n}{B} \right\rfloor - 1\right) < 4r^+(B)\left\lceil \frac{n}{B} \right\rceil.$$

Traversing this augmented Steiner tree using a depth-first circuit is guaranteed to touch on $\lceil n/B \rceil$ distinct blocks, with a total path length of no more than $8r^+(B)\lceil n/B \rceil$. Thus, we can have a blocking speed-up no more than

$$\frac{8r^+(B)\lceil n/B \rceil}{\lceil n/B \rceil - M} \to 8r^+(B)$$

as $n/M \to \infty$, since $M \geq B$ and therefore $n/B \to \infty$ as $n/M \to \infty$. □

This lemma has a simple corollary in the case that $s = 1$.

COROLLARY 1. *If the storage blow-up s is 1, then the blocking speed-up σ is less than $8r^+(B)$.*

PROOF. There are at least $\lceil n/B \rceil$ blocks required to cover the graph. Pick a representative of each of these blocks as the "must-visit" vertices. These vertices are guaranteed to satisfy the strongly stable condition of Lemma 11. □

In the case $s = 1$, choosing and connecting the $\lceil n/B \rceil$ "must-visit" vertices is an instance of what is described in the literature at the *group Steiner tree* problem. In the group Steiner tree problem, the input consists of several sets of vertices, and the goal is to find a Steiner tree that touches at least one representative of each set. Reich and Widmayer gave an approximation algorithm for the minimum group Steiner tree problem, but did not give any guarantees on how much worse than optimal their results could be [15]. Ihler has shown the problem to be NP-hard, even if the graph is a tree, and gave a trivial algorithm to approximate the optimal tree within a factor of $g - 1$, where g is the number of groups [16].

The problem with applying Lemma 11 to blockings where there are not $\lceil n/B \rceil$ strongly stable vertices is that we are not guaranteed that each of the "must-visit" vertices causes a page fault. Choosing a static set of "must-visit" vertices allows the paging algorithm possibly to satisfy several of the requests with a single block. In the extreme case where every set of B vertices comprises a block, the paging algorithm might always bring in the next B "must-visit" vertices, achieving a blocking speed-up of $Br^+(B)$. The following lemma is a generalization to Lemma 11 that gets around this problem by choosing the "must-visit" vertices dynamically.

LEMMA 12. *The fastest worst-case speed-up that could be attained in any graph is*

$$\sigma \leq 8r^+(B).$$

PROOF. We construct a skeletal Steiner tree in the graph identically to how we did in the proof for Lemma 11. However, instead of picking a fixed set of $\lceil n/B \rceil$ "must-visit" vertices, we instead consider all the vertices in the graph as potential "must-visit"

vertices. We show that every time we extend our path by $8\lceil n/B\rceil r^+(B)$, we cause at least $\lceil (n-M)/B\rceil$ page faults, giving us the desired upper bound.

We order all the vertices in the graph. We start by associating with each vertex of the skeletal Steiner tree a group consisting of all the vertices in the graph that are closer to it than to any other vertices of the skeletal Steiner tree. Vertices that are equidistant from several vertices of the skeletal Steiner tree are arbitrarily assigned to the group of one of them. We call the vertex on the skeletal Steiner tree the parent of its group. Each group has at least one vertex in it: its parent. We now pick some vertex of the skeletal Steiner tree as our distinguished start vertex, and assign it the number 1. We number any other vertices in its group in arbitrary order starting at 2. We complete the numbering by doing a depth-first circuit of the skeletal Steiner tree, numbering all the vertices of any previously unnumbered groups as we come across the group's parent node in the skeletal Steiner tree.

Assume for the moment that none of the vertices is in memory and that we start our path at the start vertex of the graph, vertex 1. In response to the request at vertex 1, the paging algorithm generates a page fault, bringing in some block of B vertices. We then consider the next "must-visit" vertex to be the first vertex according to the ordering that is not in memory (call it v). As in the proof to Lemma 11, we visit v by going along the skeletal Steiner tree in depth-first order until we get to its parent node and then taking a shortest path to v. After causing the page fault at v, we find the next "must-visit" vertex (call it w) in the same way, extend the path to w by going back to the skeletal Steiner tree reversing the path that brought us to v, taking the depth-first path in the skeletal Steiner tree to the parent of w, and taking a shortest path to w. We continue this process until we have caused $\lceil n/B\rceil$ page faults. At this point we have not completed the tour, since we had n potential "must-visit" vertices, but the paging algorithm could only bring in B at a time, so that we can always cause $\lceil n/B\rceil$ page faults. Complete the tour by considering vertex 1 as a dummy "must-visit" node and extending the path appropriately. This process creates an augmented Steiner tree among $\lceil n/B\rceil$ dynamically chosen "must-visit" vertices, with total path length no more than $8r^+(B)$, just as in the proof of Lemma 11. If there are vertices in memory, we can create the augmented Steiner tree in the same fashion, except that we can only guarantee $\lceil (n-M)/B\rceil$ "must-visit" vertices in the augmented Steiner tree. We can obviously continue cycling through the vertices indefinitely, choosing a different set of "must-visit" vertices at each pass. Thus, the overall blocking speed-up can be no more than

$$\sigma(B) \le \frac{8r^+(B)\lceil n/B\rceil}{\lceil (n-M)/B\rceil},$$

which gives the desired result as $n/M \to \infty$. □

The previous lemmas lead up to our upper bound theorem.

THEOREM 2. *The fastest worst-case speed-up that could be attained in any graph containing ρM vertices, $\rho > B$, is*

$$\sigma \le \min\left\{r^+(M),\ 2r^-(M),\ 2\frac{\rho}{\rho-1}B,\ \left(2\frac{M}{B}+3\right)r^+(B),\ 8r^+(B)\right\}.$$

PROOF. This theorem follows directly from Lemmas 7–12. □

4.2. *Lower Bounds.* If we do not care about the storage blow-up, we can easily obtain a lower bound for general graphs. As an aside, the algorithms specified in this paper differ according to the number of blocks the memory must be able to hold. Any lower-bound lemma that does not specify a minimum memory size works for any $M \geq B$.

LEMMA 13. *There is a blocking and an on-line paging algorithm that achieves a worst-case speed-up of* $r^-(B)$ *with average storage blow-up* $s = B$.

PROOF. Let the blocking $\mathcal{B}$ consist of one representative of $\mathcal{N}_v^*(B)$ for each vertex v in the graph. When a page fault occurs on some vertex w, we bring in the representative of $N_w^*(B)$, replacing whatever else is in the memory. By the definition of the radius, it takes at least $r_w(B) \geq r^-(B)$ steps to cause another page fault.

We can easily compute the storage blow-up of the blocking. The blocking has one block of size B for every vertex in the graph; thus, on average, each vertex is represented B times. However, there is no bound on the maximum number of times a vertex may be represented. □

It is natural to ask to what extent we can reduce the storage blow-up in Lemma 13. It seems excessive to center a block on every vertex. It is indeed possible to reduce the storage blow-up somewhat. To this end, we define a problem BALL COVER(r) that lets us pick some subset of the vertices around which to center blocks. The smaller the subset we can use, the less the storage blow-up.

Problem BALL COVER(r)

Input: Connected graph $G = (V, E)$ and distance r.
Output: A subset $V' \subseteq V$ of minimal cardinality such that for all $v \in V$, a $v' \in V'$ exists such that $d(v, v') \leq r$.

The problem of BALL COVER(r) is to pack a minimum number of balls of radius r into the graph in such a way that the entire graph is covered.

We can show a simple lemma relating BALL COVER(1) to VERTEX COVER. Recall that VERTEX COVER is the problem of finding a minimum cardinality subset $V' \subseteq V$ such that, for every edge $e = (u, v) \in E$, either $u \in V'$ or $v \in V'$.

LEMMA 14. *For a connected graph, any vertex cover solves BALL COVER*(1).

PROOF. Let V' be a solution to VERTEX COVER. Assume by way of contradiction that there is some vertex v such that $d(v, v') \geq 2$ for all $v' \in V'$. Since G is connected, v is connected to some vertex u. $u \notin V'$ by assumption since $d(v, u) = 1$. However, then the edge (v, u) has no representative in V', so V' was not a vertex cover. □

VERTEX COVER is an NP-complete problem [17], but there is a well-known polynomial algorithm that approximates the optimal vertex cover within a factor of 2. Take

any maximal matching of G. A maximal matching is a set of vertex-disjoint edges in a graph such that no additional edges can be added to it that are vertex-disjoint. Let $V' = \{\text{all endpoints in the maximal matching}\}$. V' is a vertex cover, since if any edge had neither endpoint in V, we could add it to the matching, implying that the matching was not maximal. It approximates the optimal vertex cover within a factor of 2, since at least one endpoint of every edge in the maximal matching must be present in the vertex cover. Unfortunately, the upper bound on the cardinality of V' is n, which fails to reduce the storage blow-up. We can use this idea to give upper bounds on the maximum cardinality subsets of BALL COVER(r).

LEMMA 15. *There is a solution V' to BALL COVER(2) such that $|V'| \leq \lfloor n/2 \rfloor$ whenever $n \geq 2$.*

PROOF. Find a maximal matching of G and let V' consist of one endpoint of every edge in the matching. If the matching is empty, the graph must consist of a single point. If the matching is nonempty, any vertex that is a member of some matching is within a distance 1 of some $v' \in V'$. Likewise, any vertex that is adjacent to a vertex in the matching is within a distance 2 of some $v' \in V'$. If there is any vertex v that is not within a distance 2 of some $v' \in V'$, it must have a distance at least 2 from any vertex in the matching. However, then there is a vertex u adjacent to v that has a distance 1 from any matching, so (v, u) could be added to the matching, contradicting our assumption that the matching was maximal. Hence, V' solves BALL COVER(2). The matching can use at most n vertices, of which only one half are in V', so $|V'| \leq \lfloor n/2 \rfloor$. □

In finding a solution to BALL COVER(1), we packed as many graphs of the form ●—● as we could, where a filled circle denotes a vertex that we include in V'. When searching for a solution to BALL COVER(2), we packed subgraphs of the form ●—○. What happens if we pack subgraphs of the form ○—●—○?

LEMMA 16. *There is a solution V' to BALL COVER(3) such that $|V'| \leq \lfloor n/3 \rfloor$ as long as $n \geq 3$.*

PROOF. Find a maximal packing of graphs of the form ○—●—○, where we include the middle vertex in V'. If the packing is empty, the graph has at most two vertices in it (either of which solves BALL COVER(3)). Assume that V' is nonempty. To show that V' solves BALL COVER(3), consider the fact that any vertex within one of the packed subgraphs must have a distance at most 1 from a vertex in V'. Thus, if there were a vertex v that had a distance of 4 from any vertex in V', then it must have a distance at least 3 from any vertex in the packing. However, if that were the case, then it would be adjacent to a vertex u of distance 2 which is adjacent to a vertex w of distance 1 from any vertex in the packing. Then the path (v, u, w) could have been added to the packing (and u added to V'), contradicting the assumption that the packing was maximal. □

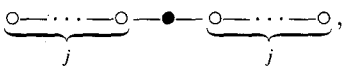
We can summarize the results above in the following table:

r	$\lvert\text{BALL COVER}(r)\rvert$
1	n
2	$\left\lfloor \frac{n}{2} \right\rfloor$
3	$\left\lfloor \frac{n}{3} \right\rfloor$

It is tempting to extrapolate the table to say that $|\text{BALL COVER}(r)| \leq \lfloor n/r \rfloor$. Unfortunately, the above construction does not generalize to $n = 4$. However, we can extend the method to show the following theorem.

THEOREM 3. *Let $j \in \mathbf{Z}^+$. Then there is a solution V' to BALL COVER$(3j)$ such that $|V'| \leq \lfloor n/(2j+1) \rfloor$ whenever $n \geq 2j+1$.*

PROOF. Consider a maximal packing of subgraphs of the form



choosing only the centermost point to be in V'. If the packing is empty, then there are no two points in the graph that are separated by a distance of more than $2j+1 \leq 3j$, so any single vertex solves BALL COVER$(3j)$; the cardinality, 1, is no more than $n/(2j+1)$ whenever there are at least $2j+1$ vertices in the graph. Now consider the case where the packing is not empty. (By definition, there are at least $2j+1$ vertices in the graph.) We claim that V' solves BALL COVER$(3j)$. Assume that there is some vertex v that has a distance (without loss of generality) of $3j+1$ from any vertex in V'. Every vertex in one of the packed subgraphs is within a distance j of some point in V', so vertex v must have a distance that is at least $2j+1$ from any vertex in the packing. However, then there is a path of length $2j+1$ that does not include vertices of any matching, contradicting the assumption that the packing was maximal. Thus, V' is a BALL COVER$(3j)$. It is evident by inspection that $|V'| \leq \lfloor n/(2j+1) \rfloor$. □

What about BALL COVER$(2j+1)$ and BALL COVER$(2j+2)$? We can extend Theorem 3 by the simple observation that any solution to BALL COVER(r) also solves BALL COVER$(r+1)$.

COROLLARY 2. *There is a solution V' to BALL COVER(r) such that $|V'| \leq n/(2\lfloor r/3 \rfloor + 1)$.*

PROOF. Choose $j = \lfloor r/3 \rfloor$ and apply Theorem 3. Every vertex is within a distance $3\lfloor r/3 \rfloor \leq r$ of some vertex of V' and the cardinality $|V'|$ is exactly the given expression. □

We are now ready to reduce the storage blow-up needed to attain a blocking speed-up of $cr^-(B)$, for some constant c.

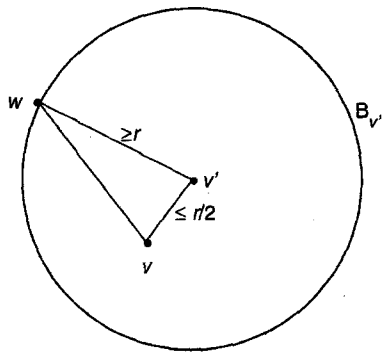


Fig. 3. The distance between vertex v and vertex w must be at least $\lceil r/2 \rceil$.

THEOREM 4. *There is a blocking and an on-line paging algorithm that achieves a speed-up* $\sigma \geq \lceil r^-(B)/2 \rceil$ *with asymptotically* $s = 3B/r^-(B)$.

PROOF. For compactness of notation, let $r = r^-(B)$. Construct a set V' to solve BALL COVER($\lfloor r/2 \rfloor$), and choose as our blocking a representative of $\mathcal{N}_v^*(B)$ for every $v \in V'$. These representatives can be constructed by breadth-first search. Recall that a compact B-neighborhood has radius at least r. When a page fault occurs at some vertex v, we know that there is some $v' \in V'$ such that $d(v, v') \leq r/2$. Bring in the block associated with v', $B_{v'}$, replacing whatever else is in the memory. The block contains every vertex within a distance $r_{v'}(B) - 1 \geq r - 1$ of v'. Consider any vertex w not within the block $B_{v'}$, as illustrated in Figure 3. By the triangle inequality, we know that

$$d(v', w) \leq d(v, w) + d(v, v')$$

or

$$d(v, w) \geq d(v', w) - d(v, v').$$

Furthermore, $d(v', w) \geq r$ and $d(v, v') \leq \lfloor r/2 \rfloor$. Consequently it takes at least $\lceil r^-(B)/2 \rceil$ steps to cause an additional page fault from v. The memory needs clearly never exceed B.

To show the storage blow-up, notice that as long as the graph has at least $\lfloor r/2 \rfloor$ vertices in it, the number of vertices in V' is

$$|V'| \leq \frac{n}{2\lfloor r/6 \rfloor + 1}.$$

Since each block contains B vertices, the average number of times each vertex is represented is

$$s \leq \frac{B}{n} \frac{n}{2\lfloor r/6 \rfloor + 1}.$$ □

Notice that the lower bounds give us tight upper and lower bounds on the blocking speed-up in uniform classes of graphs, since the upper bound we have achieved is

proportional to $r^+(B)$ and the lower bound is proportional to $r^-(B)$, which are within a constant factor of each other for uniform classes of graphs. We can give a different solution to the BALL COVER problem that works out better for grid graphs in terms of storage blow-up, but not quite as well for complete trees. First, we define something that is as close as we can come to the inverse functions for $r^+(k)$ and $r^-(k)$.

DEFINITION 7. Let $k_v(r) = |K_v(r)|$, the cardinality of the ball of radius r around vertex v. We call $k_v(r)$ the *volume* of $K_v(r)$. Then we define

$$k^+(r) = \max_{v \in V} k_v(r),$$
$$k^-(r) = \min_{v \in V} k_v(r).$$

Now we can show our other bound for BALL COVER.

THEOREM 5. *Let $G = (V, E)$ and radius r be given. Then there exists a solution V' to BALL COVER(r) with cardinality*

$$|V'| \leq \frac{n}{k^-(\lfloor r/2 \rfloor)}.$$

PROOF. Consider a maximal packing of G with balls of radius $\lfloor r/2 \rfloor$ and include in V' the center point of all these balls. We claim that every vertex v is within a distance r of some $v' \in V'$, i.e., that V' solves BALL COVER(r). Assume by way of contradiction that there is a point v that is more than a distance r from all the vertices, so that $d(v, v') \geq r+1$ for all $v' \in V'$. Then each of the neighbors $w_{1,i}$ of v has $d(w_{1,i}, v') \geq r$ for all $v' \in V'$. Similarly, those vertices of distance 2 from v have distance $d(w_{2,i}, v') \geq r - 1$, and so on down to $d(w_{\lfloor r/w \rfloor, i}, v') \geq \lceil r/2 \rceil + 1 > \lfloor r/2 \rfloor$. However, then we could include $K_v(\lfloor r/2 \rfloor)$ in the packing, contradicting the assumption that the packing was maximal. Hence, V' solves BALL COVER(r). To show the cardinality, simply notice that each of the balls of radius $\lfloor r/2 \rfloor$ removes at least $k^-(\lfloor r/2 \rfloor)$ vertices from further consideration in the packing; at most

$$\frac{n}{k^-(\lfloor r/2 \rfloor)}$$

such balls could be removed. □

Now we can do a new version of Theorem 4.

THEOREM 6. *There is a blocking and an on-line paging algorithm that achieves a speed-up of $\lceil r^-(B)/2 \rceil$ with $s \leq B/k^-(\lfloor r^-(B)/4 \rfloor)$.*

PROOF. As in the proof to Theorem 4, construct V' to solve BALL COVER($\lfloor r^-(B)/2 \rfloor$) and pick a compact B-neighborhood around each $v \in V'$. The same argument shows

that the speed-up is at least $\lceil r^-(B)/2 \rceil$. By Theorem 5, the cardinality of V' need be no more than

$$|V'| \leq \frac{n}{k^-(\lfloor r^-(B)/4 \rfloor)}. \qquad \square$$

For d-dimensional grid graphs, this theorem results in a storage blow-up of 4^d. It may be possible to reduce the base of the exponent using a different argument, but the fact that we need to consider neighborhoods of radius $r/2$ to get a ball cover of radius r means that the result is exponential.

We do not have any lower-bound results for general graphs that have constant storage blow-up. We hope to get something with $s =$ constant that is proportional to $r^-(B)$. It is likely that when we disallow storage blow-up then we cannot obtain such good results; a general proof technique for showing upper bounds in the absence of storage blow-up is not known.

In the next two sections we consider special-case arguments for complete trees and grid graphs.

5. Searching in Complete Trees. First we consider upper bounds on the blocking speed-up in complete d-ary trees. After showing the upper bounds, we give algorithms to achieve them within a constant factor with a small constant storage blow-up.

5.1. *Upper Bounds.* If we allow the arity of the tree to be arbitrarily large, we immediately run into problems where we cannot use blocking efficiently. Consider, for example, a graph in which some vertex v is connected to M other vertices. It is possible to cause (at least) one page fault by extending the path to v and then going to any of the other vertices that are not in memory. There must be at least one such vertex. Thus, we can generate a page fault with every other vertex for a worst-case blocking speed-up $\sigma(B) \leq 2$ independently of B.

We can generalize this argument to complete d-ary trees to give a result that is about a factor of 8 stronger than the general $8r^+(B)$ result of Lemma 12.

THEOREM 7. *In a complete d-ary tree of height h, the maximum worst-case speed-up that could be guaranteed is*

$$\sigma(B) \leq \frac{2h}{(h/\log_d B) - \log_d M}. \tag{2}$$

PROOF. Consider an adversary who is trying to cause as many page faults as possible. Assume that the pathfront is at the root, with some M vertices in the memory. Thereafter, the adversary walks down the tree, taking at each level the branch that goes to the closest vertex that is not in the memory. Since at most $(d^{r+1} - 1)/(d - 1)$ vertices can be within a distance r of the pathfront, it is guaranteed that a page fault is generated at least every $\log_d B$ steps in regions that were previously not in the memory. The fact that there were M items in the memory means that the path could traverse up to $\log_d M$ of these, for a total of at most $\log_d M$ page faults not caused. Once the adversary has reached a leaf, he extends the path by going directly back to the root. If we assume that the tree has a

total height of h, and that no page faults are caused on the way back to the root, then the adversary can cause a minimum of $(h/\log_d B) - \log_d M$ page faults in $2h$ steps. Since this process leaves the same state as was assumed initially, the process can continue indefinitely. Taking the ratio of steps to page faults gives the result. □

COROLLARY 3. *As the height of the tree grows without bound, the maximum worst-case speed-up is* $\sigma(B) \leq 2 \lg B / \lg d$.

PROOF. This result follows by taking the limit of (2) as $h \to \infty$. □

For upper bounds on other kinds of trees, see Section 4, which gives bounds for general graphs.

5.2. *Lower Bounds.* So far, there is no known optimal algorithm that does not have storage blow-up greater than 1. The obvious blocking where we put nonoverlapping trees of height $k = \lfloor \log_d B \rfloor$ into blocks allows a simple adversarial algorithm to produce a path for which the blocking speed-up is only $\sigma(B) \sim 2$, regardless of how much memory is allowed. However, we can give a simple blocking with $s = 2$ that achieves optimal performance.

LEMMA 17. *There is a blocking in a d-ary tree with* $s = 2$ *and an on-line paging algorithm such that the worst-case speed-up* σ *is at least* $\lg B / 2 \lg d$.

PROOF. Consider a blocking where we put complete trees of height $k = \lg B / \lg d + o(\lg B)$ into each block. Each of the children of the leaves of a block is the root of a new block. Assume that the root of the tree is the root of the top block. We call this blocking $\mathcal{B}_1$. We overlap a second such blocking $\mathcal{B}_2$ offset by a height of $k/2$, where $k = \lg B / \lg d + o(\lg B)$, as shown in Figure 4. Clearly, this can be done with a storage blow-up of 2, since vertices are represented at most once in each of $\mathcal{B}_1$ and $\mathcal{B}_2$. Now consider a step where the pathfront causes a page fault. There are two cases to consider:

(1) The pathfront stepped off the top (i.e., toward the root) of a block that was in memory.
(2) The pathfront stepped off the bottom (i.e., toward the leaves) of a block that was in memory.

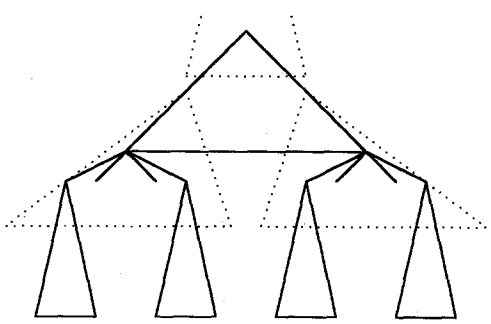


Fig. 4. A blocking of a d-ary tree using $s = 2$.

Without loss of generality assume that the block being stepped out of belongs to $\mathcal{B}_1$. In either case we bring in the block in $\mathcal{B}_2$ containing the pathfront into the memory (getting rid of anything else in the memory to make room for the block). By construction, it requires at least $k/2$ additional steps to cause another page fault, proving the result. □

The lower bound when we allow a constant degree of storage blow-up is within a factor of 4 of the upper bound. This result contrasts with the general case, where potentially large (nonconstant) storage blow-up is required to come within a constant factor of the optimal blocking speed-up. It is an open question what is possible in the absence of storage blow-up.

6. Searching in Grid Graphs and Diagonal Grid Graphs. By a grid graph in d dimensions, we mean a graph, where the vertex set is $\mathbf{Z}^d$ and the edge set is

$$E = \left\{ ((a_0, \ldots, a_d), (b_0, \ldots, b_d)) \left| \sum_i |a_i - b_i| = 1 \right. \right\}.$$

For example, in one dimension, the grid graph consists of the integer points on a number line. In this section we consider mainly infinite grid graphs. We also consider diagonal grid graphs, which have the same vertex set as grid graphs, but have edge set

$$E = \{((a_0, \ldots, a_d), (b_0, \ldots, b_d)) \mid |a_i - b_i| \leq 1 \text{ for all } 1 \leq i \leq d\}.$$

Figure 5 shows an example of a diagonal grid graph in two dimensions.

6.1. *One-Dimensional Grid Graphs.* We start with the results for one-dimensional grid graphs, where we have tight upper and lower bounds for the speed-up. The bounds, in particular the upper bound, may seem trivial, but understanding the argument in one dimension is the key to understanding the higher-dimensional analogues. (The upper bound is also better in some ways than any we have been able to prove for general graphs.) Note that in one dimension, a diagonal grid graph is the same as a grid graph.

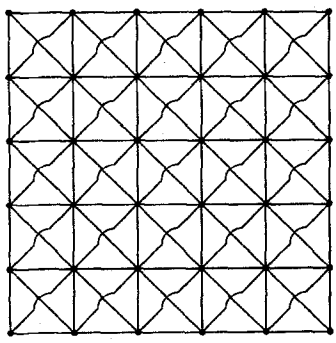


Fig. 5. A diagonal grid graph in two dimensions.

6.1.1. *Upper Bounds.*

LEMMA 18. *The fastest worst-case speed-up that could be achieved in a one-dimensional grid graph is* $\sigma \leq B$.

PROOF. We adopt an adversarial position and demonstrate that under any blocking, we can force a page fault every B steps in the graph. We start anywhere to the right of all the covered vectors (we arbitrarily call it vertex 0) and always walk to the right. We keep a potential function $\varphi(t)$ which increases by B every time we cause a page fault, and decreases by 1 for every step to the right we take, with $\varphi(0) = 0$. Let $p(t)$ be the number of page faults caused after we have taken t steps, and let $\varphi(t)$ be the value of the potential function after t steps. We want to show that, at all times,

$$p(t) \geq \frac{t}{B}, \tag{3}$$

or in other words that

$$\sigma = \frac{t}{p(t)} \leq B.$$

By definition,

$$\varphi(t) = Bp(t) - t, \tag{4}$$

so rearranging (4), we get

$$p(t) = \frac{t}{B} + \frac{\varphi(t)}{B}. \tag{5}$$

Comparing (3) and (5), we see that we can satisfy (3) by showing that, for all t, $\varphi(t) \geq 0$. However, $\varphi(t) - 1$ is an upper bound on how many covered cells (i.e., cells that are in memory) are to the right of t, since it increases by $B - 1$ every time a page fault is caused and decreases by 1 every time a covered cell is reached. Since the number of covered cells to the right is nonnegative and $\varphi(t)$ is at least that large, it follows that $\varphi(t) \geq 0$. □

We can also give a simple upper-bound proof in the case where we have a finite one-dimensional grid graph.

LEMMA 19. *The fastest worst-case blocking speed-up that could be achieved in a one-dimensional grid graph containing* ρM *vertices,* $\rho > 1$, *is*

$$\sigma \leq \frac{\rho}{\rho - 1} B - \frac{B}{(\rho - 1)M}.$$

PROOF. Assume that we are at one end of the grid graph and that there are at most M vertices in memory. We follow a path that traverses to the other end of the grid graph. There are $(\rho - 1)M$ vertices not in memory, which means that we cause at least $(\rho - 1)M/B$ page faults in $\rho M - 1$ steps. Taking the ratio of steps to page faults gives the result. □

Notice that when $\rho \to \infty$, the upper bound of Lemma 19 reduces to that of Lemma 18.

6.1.2. *Lower Bounds.* It is possible to achieve the upper bound using the obvious strategy of putting B vertices in one block, the next B vertices in the next block, and so on.

LEMMA 20. *There is a blocking of the one-dimensional grid graph with $s = 1$ that achieves a worst-case speed-up of $\sigma \geq B$, as long as $M \geq 2B$.*

PROOF. Let $B_i = \{k \mid Bi \leq k < B(i+1)\}$ and consider the obvious blocking $\mathcal{B} = \{B_i \mid i \in \mathbf{Z}\}$. (See Figure 7(a).) The blocks are nonoverlapping sequences of B consecutive vertices. We show that in the steady state, we can service at least B steps in the graph every time a page fault is generated. Assume that we cause a page fault by stepping to vertex k. Without loss of generality, assume that we have stepped from vertex $k-1$; the same argument applies in the other direction. Then $k = Bi$ for some i and we can service the request by bringing in block B_i. Since $k-1$ was in memory, we know that block B_{i-1} had been read in at some previous time. We retain block B_{i-1}. Now, in order to cause another page fault, we must move from vertex k to either vertex $k-B-1$ or vertex $k+B$ since all the intervening nodes are in memory. Thus, it requires a minimum of B steps to cause another page fault. □

It is also possible to alleviate the memory requirement to $M \geq B$ if we allow the storage blow-up s to be 2 and let the speed-up decline to $B/2$ by overlapping two sets of the blockings used in the proof of Lemma 20 offsetting one by $B/2$ with respect to the other.

6.2. *Two-Dimensional Grid Graphs.* We now turn to two-dimensional grid graphs, where the results are not trivial.

6.2.1. *Upper Bound.*

LEMMA 21. *In a two-dimensional grid graph the fastest worst-case speed-up that could be achieved is $\sigma \leq 2\sqrt{B}$.*

PROOF. This proof proceeds in a similar fashion to that of Lemma 18. This time we consider an infinitely long, $\sqrt{B}$-wide corridor extending to the right. We divide t, the number of steps taken, into two components, t_x and t_y. We let t_x be an index of how many columns we have traversed in the corridor (again, we always move to the right), and let t_y be the total number of steps we have taken in the y direction. As before, we want to show that

$$p(t) \geq \frac{t}{2\sqrt{B}}. \tag{6}$$

We show that

$$t_x \leq \sqrt{B}\,p(t), \tag{7}$$

$$t_y \leq \sqrt{B}\,p(t), \tag{8}$$

so that $t = t_x + t_y \leq 2\sqrt{B}p(t)$, implying (6). Our scheme is actually quite simple: we show that there is an uncovered vertex in the corridor within the next (amortized) $\sqrt{B}$ columns (including the current column) and then an additional $\sqrt{B} - 1$ moves in the y direction suffice to step to any uncovered vertex in that column. As before, we define a potential function $\varphi(t)$ that increases by B for every page fault we cause. This time, however, we decrement only when t_x increases, and we decrement by $\sqrt{B}$. Thus,

$$\varphi(t) = Bp(t) - \sqrt{B}t_x. \tag{9}$$

$\varphi(t)$ is an upper bound on the number of vertices covered in the channel to the right of (and including) the current column, since at most B vertices to the right can be newly covered when we cause a page fault, and by definition each step to the right leaves behind $\sqrt{B}$ covered cells. Rearranging (9), we get that

$$t_x = \sqrt{B}p(t) - \frac{\varphi(t)}{\sqrt{B}}. \tag{10}$$

Comparing (7) and (10), we see that we only need to show that $\varphi(t) \geq 0$ to establish (7). However, the interpretation of $\varphi(t)$ ensures this, since the number of covered columns is nonnegative.

We do not need to use an amortized argument to show (8), since a page fault is guaranteed to occur at least every $\sqrt{B} - 1$ steps in the y direction. □

6.2.2. *Lower Bounds.* We can find a blocking that comes close to the above limit, if we allow a storage blow-up $s = 2$. First we need a definition.

DEFINITION 8. By a *tessellation*, we mean a collection of regions such that the union of all the regions is the entire space and the intersection of any pair of regions never includes a point in the interior of either.

For simplicial tessellations, it is often required that any intersection between two simplices be a simplex of each; we make no such requirement in our definition.

LEMMA 22. *There exists a blocking in a two-dimensional grid graph with $s = 2$ and an on-line paging algorithm that has a worst-case speed-up $\sigma \geq \sqrt{B}/4$, assuming that $M \geq 2B$.*

PROOF. Consider a tessellation of the plane by blocks of size $\sqrt{B} \times \sqrt{B}$. Let the blocking be two such tessellations, offset by $\sqrt{B}/2$. Figure 6 demonstrates the blocking, where one tessellation is shown with solid lines and the other with dashed lines. Formally, we consider the blocking $\mathcal{B} = \{B_{ij} : i, j \in \mathbf{Z}\}$, where

$$B_{ij} = \left\{(k, \ell): \left\lfloor \frac{i\sqrt{B}}{2} \right\rfloor \leq k < \left\lfloor \frac{i\sqrt{B}}{2} \right\rfloor + \sqrt{B}, \left\lfloor \frac{j\sqrt{B}}{2} \right\rfloor \leq \ell < \left\lfloor \frac{j\sqrt{B}}{2} \right\rfloor + \sqrt{B}\right\}.$$

Consider a step that causes a page fault. Without loss of generality, let the step be to the right out of the shaded block in Figure 6. By leaving the shaded block in memory and

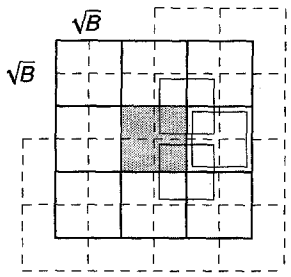


Fig. 6. A blocking of vertices in a two-dimensional grid with storage blow-up $s = 2$.

bringing in the appropriate boxed block, we can make sure that it takes at least $\sqrt{B}/4$ additional steps to produce another page fault. □

It is also possible to prove a lower bound on the speed-up even if there is no storage blow-up, but it is not quite as good. We see in the next subsection that there is a good reason for this. Note also that the memory requirement is a little more stringent, requiring $M \geq 3B$ instead of $M \geq 2B$.

LEMMA 23. *There exists a blocking in a two-dimensional grid graph with* $s = 1$ *and an on-line paging algorithm that has a worst-case speed-up* $\sigma \geq \sqrt{B}/6$, *assuming that* $M \geq 3B$.

PROOF. Consider the blocking of Figure 7(b), where each square has dimensions $\sqrt{B} \times \sqrt{B}$. The slowest speed-up occurs when the pathfront emerges at the place where three of the squares meet; in two steps it is possible to cause two page faults. However, the next page fault after those two requires at least $\sqrt{B}/2$ additional steps, so that we take $\sqrt{B}/2 + 2$ steps to cause three page faults, for a speed-up of $\sqrt{B}/6 + \frac{2}{3}$. □

6.3. *d-Dimensional Grid Graphs and Diagonal Grid Graphs.* We can also show upper and lower bounds on higher-dimensional grid graphs and diagonal grid graphs. The one- and two-dimensional results are special cases of the result for d dimensions, but it was useful to start there to develop our intuitions. However, unexpected results come from generalizing to multiple dimensions, like the fact that our upper and lower bounds are no longer within a constant factor if we block using isothetic (all sides parallel to the coordinate axes) hypercubes, and that going from $s = 1$ to $s = 2$ gives provably more than a constant factor improvement in σ in any blocking using isothetic hypercubes. Furthermore, it does not appear possible to create an optimal algorithm for higher-dimensional grid graphs with any reasonable storage blow-up (see Section 4); on the other hand, there is an optimal algorithm for higher-dimensional diagonal grid graphs with a storage blow-up of 2.

6.3.1. *Upper Bounds.* We give a special-case argument for d-dimensional grid graphs that improves upon the general $8r^+(B)$ upper bound by a constant factor of $4/e \approx 1.47$.

LEMMA 24. *In a d-dimensional grid graph, the fastest worst-case speed-up that could be attained is* $dB^{1/d}$.

PROOF. The argument for this proof is just like that of Lemma 21. This time we have an infinite corridor to the positive first dimension with size

$$\underbrace{B^{1/d} \times \cdots \times B^{1/d}}_{d-1}.$$

We again divide t into d components, $t_1, \ldots, t_d$. We amortize in dimension 1, but not in the others, to show independently that

$$t_i \leq B^{1/d} p(t), \qquad 1 \leq i \leq d. \tag{11}$$

so that

$$t = \sum_{1 \leq i \leq d} t_i \leq dB^{1/d} p(t),$$

giving the desired result

$$p(t) \geq \frac{t}{dB^{1/d}}.$$

Again, we step to an uncovered location in the corridor such that t_1 is increased the minimum amount. This time we define

$$\varphi = Bp(t) - B^{(d-1)/d} t_1, \tag{12}$$

which again is an upper bound on the number of vertices in the memory that have their first coordinate at least t_1. Rearranging (12), we get

$$t_1 = B^{1/d} p(t) - \frac{\varphi(t)}{B^{(d-1)/d}},$$

showing that the first coordinate of (11) is satisfied as long as $\varphi(t) \geq 0$, which it is by its interpretation. The other dimensions all need to take fewer than $B^{1/d}$ steps per page fault to move to the empty space in the corridor. □

On the other hand, we can prove a tighter upper bound for diagonal grid graphs.

LEMMA 25. *In a d-dimensional diagonal grid graph, the fastest worst-case speed-up that could be attained is* $\sigma(B) \geq 2B^{1/d}$.

PROOF. The argument for this proof combines ideas from the proofs of Lemmas 21 and 24. We consider an infinite corridor to the positive first dimension with size

$$\underbrace{B^{1/d} \times \cdots \times B^{1/d}}_{d-1}.$$

This time, we divide t into two components, t_1 and t_2. We amortize in dimension 1, but not in dimension 2, to show independently that

$$t_i \leq B^{1/d} p(t), \qquad 1 \leq i \leq 2, \tag{13}$$

so that

$$t = t_1 + t_2 \leq 2B^{1/d} p(t),$$

giving the desired result

$$p(t) \geq \frac{t}{2B^{1/d}}.$$

Again, we step to an uncovered location in the corridor such that t_1 is increased the minimum amount. This time we define

$$\varphi = Bp(t) - B^{(d-1)/d} t_1, \tag{14}$$

which again is an upper bound on the number of vertices in the memory that have their first coordinate at least t_1. Rearranging (14), we get

$$t_1 = B^{1/d} p(t) - \frac{\varphi(t)}{B^{(d-1)/d}},$$

showing that the first coordinate of (13) is satisfied as long as $\varphi(t) \geq 0$, which it is by its interpretation. Unlike the situation with d-dimensional grid graphs, however, the diagonals ensure that all the other dimensions can reach the empty space in the corridor within $B^{1/d}$ steps per page fault, since all the other dimensions can be moving one step closer to the empty space simultaneously at each step. □

6.3.2. *Lower Bounds.* In a similar fashion we can get lower bounds on d-dimensional grid graphs and diagonal grid graphs.

LEMMA 26. *There exists a blocking of a d-dimensional grid graph or a d-dimensional diagonal grid graph with $s = 2$ and an on-line paging algorithm that produces a worst-case speed-up of $\sigma \geq \frac{1}{4}B^{1/d}$, for $M \geq 2B$.*

PROOF. This proof proceeds similarly to that of Lemma 22. We let our blocks be of size $B^{1/d}$ in each dimension and consider two overlapping grids where the corners of one grid correspond to the centers of the other grid. Exactly the same argument shows that when a page fault occurs, there is always some block that can be brought in so that the next page fault takes at least time $\frac{1}{4}B^{1/d}$ to occur. □

This algorithm is optimal within a factor of 8 for diagonal grid graphs. However, for grid graphs there is a discrepancy of $4d$ between the upper bound of Lemma 24 and this lower bound of Lemma 26. As we see in the next lemma, it is the upper bound that is correct, and if we permit the storage blow-up to be as large as B, we can attain a blocking speed-up of at least $(1/2e)\, dB^{1/d}$, which is off from the upper bound by a factor of at most $2e$.

LEMMA 27. *There is a blocking with $s = B$ that gives a worst-case speed-up of $\sigma(B) \geq (1/2e)\, dB^{1/d}$ in a d-dimensional grid graph.*

PROOF. Lemma 13 gives a blocking with $s = B$ that achieves a blocking speed-up of $r^-(B)$. From (1), $r^-(B) = (1/2e)\, dB^{1/d}$. □

Thus, the blocking given in the proof of Lemma 13 is better than the best known one with a constant storage blow-up ($s = 2$). The storage blow-up needed for this blocking is quite large, however, and it is not possible to describe the blocking simply. Using the storage reduction techniques of Theorems 4 and 6, we can reduce the storage blow-up to

$$s \leq \min\left\{\frac{6e}{d} B^{(d-1)/d}, 4^d\right\}$$

while giving up at most a factor of 2 in the blocking speed-up.

6.3.3. *Blocking with Isothetic Hypercubes and $s = 1$.* Intuitively, the problem with using isothetic hypercubes as blocks to tessellate the space, as in Lemma 26, is that the blocks do not correspond well with the neighborhoods of points that are within a certain distance r of some central point. In two dimensions, for example, the neighborhoods are diamond-shaped. In three dimensions they are octahedra. In general, the shape of the neighborhoods is the geometric dual of the isothetic hypercubes: take the centerpoint of each face of the d-dimensional hypercube and take the convex hull of the resulting points. The unfortunate property of using the geometric duals as neighborhoods is that when $d \geq 3$, the neighborhoods do not tessellate neatly.

We do not get even as good a result as Lemma 26 with isothetic hypercubes when we are not allowed to use storage blow-up greater than 1. The following lemma gives a lower bound on the speed-up in grid graphs when no storage blow-up is permitted.

LEMMA 28. *There is a blocking with $s = 1$ in a d-dimensional grid graph or a d-dimensional diagonal grid graph and an on-line paging algorithm that asymptotically achieves a worst-case speed-up of $\sigma(B) \geq B^{1/d}/2d^2$ as long as $M \geq (d+1)B$.*

PROOF. This proof follows along the same lines as that of Lemma 23. Let p be the smallest prime such that $p \geq d$. By a theorem of Chebychev, $p < 2d$. We block using regular isothetic hypercubes that have sides equal to $B^{1/d}$. We tessellate d-dimensional space by stacking up tessellations of $(d-1)$-dimensional space extruded along the dth dimension. We use the same $(d-1)$-dimensional tessellation in each layer, but we offset adjacent layers by $1/p$ in the first dimension, $2/p$ in the second dimension, etc., to $(d-1)/p$ in the $(d-1)$st dimension, to arrange the tessellation so that not too many blocks meet at any given point. Figure 7 shows the tessellation for $1 \leq d \leq 3$. This stacking has been set up so that at no point do more than $d+1$ hypercubes meet; the construction ensures that a point where d hypercubes meet in $d-1$ dimensions is always in the middle of a face along the dth dimension. It similarly controls that places where $d-1$ hypercubes meet in $d-1$ dimensions never stack onto a place that two hypercubes meet, and so on. Thus, we can cause at most d page faults in consecutive moves, as long

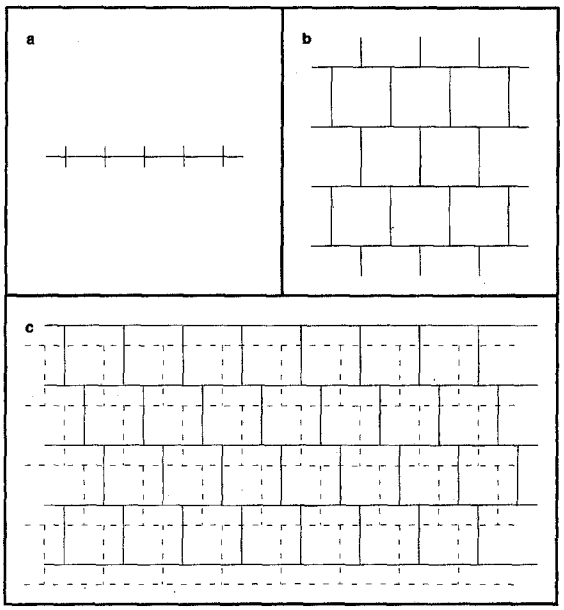


Fig. 7. A blocking of vertices in grid graphs with storage blow-up $s = 1$, in (a) one, (b) two, and (c) three dimensions. The sides of each block have length $B^{1/d}$.

as we keep all the blocks meeting at a point in memory—this leads to the restriction that $M \geq (d+1)B$; to cause an additional page fault always takes at least $B^{1/d}/p$ additional steps. Thus the speed-up is at least

$$\frac{d + B^{1/d}/p}{d+1},$$

which gives the result since $p \leq 2d$. □

Notice that this bound is off from the general grid graph upper bound by a factor of $2d^3$. In fact, when storage blow-up is not permitted, we can prove a more stringent upper bound on any blocking that tessellates the d-dimensional space using isothetic hypercubes of size $B^{1/d}$ in each dimension; this algorithm is only off by a factor of d from the tighter upper bound. First we need a couple of topological lemmas regarding tessellations of d-space by unit hypercubes.

LEMMA 29. *Any tessellation of d-space using isothetic unit hypercubes can be decomposed into layers of tessellations of $(d-1)$-space extruded along the dth dimension.*

PROOF. Consider the two squares C_1 and C_2 in Figure 8 as projections into two dimensions of two hypercubes whose intersection is a $(d-1)$-dimensional region that is not a complete face of either hypercube. Clearly, the dotted cube C_3 is not an acceptable placement for an additional hypercube that touches C_2, since we know that the hypercubes

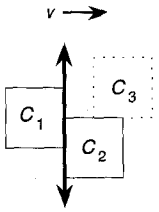


Fig. 8. A tessellation of d-space using isothetic unit hypercubes must consist of a series of shear hyperplanes.

tessellate the space, but no unit hypercube could fit in the niche between cubes C_1 and C_3. Thus, cube C_3 must abut both cubes C_1 and C_2. Continuing this argument shows that the solid hyperplane perpendicular to v must be a shear plane. Likewise, the hyperplanes on both sides of the $(d-1)$-dimensional tessellations must be shear planes, establishing the lemma. □

This lemma enables us to prove the next one, which we apply to get the more stringent upper bound for the blocking where we do not permit storage blow-up. We need a definition first.

DEFINITION 9. A *complex of degree D* in a tessellation is a point that is incident upon D distinct regions of the tessellation.

LEMMA 30. *Any tessellation of d-space using unit hypercubes contains a complex of degree at least* $d+1$.

PROOF. The proof is by induction on the dimension d. In one dimension there clearly must be points where two unit lines meet, forming a complex of degree 2. Assume that the lemma is true for $d-1$ dimensions. We know from Lemma 29 that the d-dimensional tessellation can be decomposed into layers of extruded $(d-1)$-dimensional tessellations, and, by the inductive hypothesis, that there is some complex of degree d in each of these tessellations. When we arrange two adjacent layers together, the best we can do to limit the degree of complexes is to position a complex of degree d in one layer incident upon the $(d-1)$-dimensional face of the next layer. This process, however, produces a complex of degree $d+1$, maintaining the inductive hypothesis. □

Of course, in Lemmas 29 and 30 it does not matter that each hypercube has sides of unit length; the important fact is that all the hypercubes have identical sizes, so we can apply the lemma to the problem of finding a tessellation of a d-dimensional grid graph using hypercubes with sides that are all $B^{1/d}$. We set up our tessellation in such a way that the boundaries of the hypercubes always pass between integer points of the grid graph, so that we can maintain a storage blow-up of 1.

LEMMA 31. *The fastest speed-up that could be achieved by decomposing a d-dimensional grid graph into blocks of size $B^{1/d}$ on a side is*

$$\sigma \leq \frac{B^{1/d} + d}{d + 1}.$$

PROOF. By Lemma 30, there must be a $(d + 1)$-complex in the tessellation underlying our block decomposition. In a $(d + 1)$-complex in a d-dimensional space, it is always possible for a path to visit all $d + 1$ of the blocks that are "incident" in consecutive moves. (This fact can be proved by induction on the number of dimensions.) Thus, if we extend the path to a point that is on a $(d + 1)$-complex where we have not yet brought in any of the incident blocks, we can cause d page faults in d steps. It takes at most $B^{1/d}$ steps to move to another $(d + 1)$-complex where only one block is in memory, to return to the starting situation. This fact means that we cause $d + 1$ page faults every $B^{1/d} + d$ steps. □

This upper bound is a factor of $d/4$ worse than the one attained with isothetic hypercubes by using a storage blow-up $s = 2$.

7. Conclusions. We have demonstrated matching upper and lower bounds for the blocking speed-up in searching complete d-ary trees and d-dimensional grid graphs. We have also shown matching upper and lower bounds for classes of general graphs for which the maximum and minimum B-radii are within a constant factor of one another. Table 1 summarizes these results.

One of the issues we explored was the degree to which storage blow-up was necessary in attaining optimal blocking speed-ups. For complete trees and d-dimensional grid graphs, we were able to show the existence of optimal algorithms with a degree of storage blow-up that was independent of the block size (although there is still a dependence on the dimension for grid graphs). For general graphs, the degree of storage blow-up required to get optimal blocking speed-up by our algorithms is in general dependent upon the block size. In the case of d-dimensional grid graphs blocked using only isothetic hypercubes, we were able to show that even going from a storage blow-up of 1 to 2 results in a provably better performance for the worst-case blocking speed-up, reflected in Table 1 by the fact that the lower bound for $s = 2$ is larger than the upper bound for $s = 1$ as long as $d > 4$. Thus, some degree of storage blow-up is necessary to get good blocking speed-ups. This observation implies that these techniques would not be very useful for applications that have a significant cost for updating the data being stored.

There are clearly important questions that are still open.

1. Is it possible to prove that there are optimal lazy algorithms for the strong model in nonuniform classes of graphs? We have demonstrated optimal lazy algorithms that work in the strong model for uniform classes of graphs.
2. What is the best possible speed-up attainable when $s = 1$? Is there an algorithm to achieve the speed-up?
3. Are there upper-bound techniques that take into account the storage blow-up?

Table 1. Summary of blocking results for external graph searching.

Type of graph	Upper bound	Lower bound	s	$\frac{M}{B}$
Complete d-ary tree	$2\frac{\log B}{\log d}$	$\frac{\log B}{2\log d}$	2	≥ 1
One-dimensional grid graphs	B	B	1	≥ 2
		$\frac{B}{2}$	2	≥ 1
Two-dimensional grid graphs	$2\sqrt{B}$	$\frac{1}{6}\sqrt{B}$	1	≥ 3
		$\frac{1}{4}\sqrt{B}$	2	≥ 2
d-Dimensional grid graphs	$dB^{1/d}$	$\frac{1}{2e}dB^{1/d}$	B	≥ 1
		$\frac{1}{4e}dB^{1/d}$	$\frac{6e}{d}B^{(d-1)/d}$	≥ 1
		$\frac{1}{4e}dB^{1/d}$	4^d	≥ 1
d-Dimensional grid graphs (isothetic hypercubes)	$dB^{1/d}$	$\frac{1}{4}B^{1/d}$	2	≥ 2
	$\frac{1}{d}B^{1/d}$	$\frac{1}{2d^2}B^{1/d}$	1	$\geq d+1$
d-Dimensional diagonal grids	$2B^{1/d}$	$\frac{1}{4}B^{1/d}$	2	≥ 2
General graphs with ρM vertices	$r^+(M)$	$r^-(B)$	B	≥ 1
	$2r^-(M)$	$\frac{1}{2}r^-(B)$	$\frac{3B}{r^-(B)}$	≥ 1
	$2\frac{\rho}{\rho-1}B$	$\frac{1}{2}r^-(B)$	$\frac{B}{k^-(\lfloor r^-(B)/4 \rfloor)}$	≥ 1
	$\left(2\frac{M}{B}+3\right)r^+(B)$			
	$8r^+(B)$			

4. Is it possible to reduce the storage blow-up to some constant and still achieve an optimal speed-up?
5. How do the results given here apply to directed graphs, instead of undirected graphs? This extension is particularly important for object-oriented databases.
6. Is it possible to get matching upper and lower bounds for nonuniform classes of graphs? Intuitively, if there are a few vertices with a small B-radius, but they are separated by a large distance, say $r^+(M)$, then the adversary cannot take too much advantage of them if $B \ll M$.
7. Are there algorithms that trade off memory usage for better blocking speed-ups? Most of our algorithms have the ratio $M/B = 1$.
8. What kinds of results can be obtained using competitive analysis?

Acknowledgments. We would like to thank Darren Vengroff for some helpful conversations regarding the BALL COVER problem.

References

[1] D. E. Knuth, *The Art of Computer Programming*, Vol. 3, Addison-Wesley, Reading, MA, 1973.

[2] A. Aggarawal and J. Park, Notes on Searching in Multidimensional Monotone Arrays, *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, White Plains, NY, October 1988, pp. 497–512.

[3] J. D. Ullman, *Principles of Database and Knowledge-Base Systems*, Computer Science Press, Rockville, MD, 1988.

[4] A. Borodin, S. Irani, P. Raghavan, and B. Schieber, Competitive Paging with Locality of Reference, *Proceedings of the 23rd ACM Symposium on Theory of Computing*, New Orleans, LA, May 1991, pp. 249–259.

[5] J. D. Ullman and M. Yannakakis, The Input/Output Complexity of Transitive Closure, *Ann. Math. Artificial Intel.* **3** (1991), 331–360.

[6] A. L. Rosenberg, Preserving Proximity in Arrays, *SIAM J. Comput.* **4** (1975), 443–460.

[7] R. A. DeMillo, S. C. Eisenstat, and R. J. Lipton, Preserving Average Proximity in Arrays, *Comm. ACM* **21** (1978), 228–231.

[8] A. L. Rosenberg, Encoding Data Structures in Trees, *J. Assoc. Comput. Mach.* **26** (1979), 668–689.

[9] A. L. Rosenberg, Data Encodings and Their Costs, *Acta Inform.* **9** (1978), 273–292.

[10] A. L. Rosenberg and L. Snyder, Bounds on the Costs of Data Encodings, *Math. Systems Theory* **12** (1978), 9–39.

[11] F. R. K. Chung, A. L. Rosenberg, and L. Snyder, Perfect Storage Representations for Families of Data Structures, *SIAM J. Algebraic Discrete Methods* **4** (1983), 548–565.

[12] R. Aleliunas and A. L. Rosenberg, On Embedding Rectangular Grids in Square Grids, *IEEE Trans. Comput.* **31** (1982), 907–913.

[13] R. J. Lipton, S. C. Eisenstat, and R. A. DeMillo, Space and Time Hierarchies for Classes of Control Structures and Data Structures, *J. Assoc. Comput. Mach.* **23** (1976), 720–732.

[14] C. Berge, *Graphs and Hypergraphs*, 2nd edn., North-Holland, Amsterdam, 1976.

[15] G. Reich and P. Widmayer, Beyond Steiner's Problem: A VSLI Oriented Generalization, in *Graph-Theoretic Concepts in Computer Science: Proceedings of the 15th International Workshop WG '89* (G. Goos and J. Hartmanis, eds.), Lecture Notes in Computer Science, Vol. 411, Springer-Verlag, Berlin, 1990, pp. 196–210.

[16] E. Ihler, Bounds on the Quality of Approximate Solutions to the Group Steiner Problem, in *Graph-Theoretic Concepts in Computer Science, Proceedings of the 16th International Workshop WG '90* (G. Goos and J. Hartmanis, eds.), Lecture Notes in Computer Science, Vol. 484, Springer-Verlag, Berlin, 1991, pp. 109–118.

[17] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, CA, 1979.