

# Bounded-Collision Memory-Mapping Schemes for Data Structures with Applications to Parallel Memories

Gennaro Cordasco, Vittorio Scarano, and Arnold L. Rosenberg, *Fellow, IEEE*

**Abstract**—Techniques are developed for mapping structured data to an ensemble of parallel memory modules in a way that limits the number of *conflicts*, i.e., simultaneous accesses by distinct processors to the same memory module. The techniques determine, for any given *conflict tolerance*  $c$ , the smallest ensemble that allows one to store *any*  $n$ -node data structure “of type  $X$ ” in such a way that no more than  $c$  nodes of a structure are stored on the same module. This goal is achieved by determining the smallest  $c$ -*perfect universal graphs* for data structures “of type  $X$ .” Such a graph is the smallest graph that contains a homomorphic image of each  $n$ -node structure “of type  $X$ ,” with each node of the image holding  $\leq c$  nodes of the structure. In the current paper, “type  $X$ ” refers to *rooted binary trees* and three array-like structures: *chaotic arrays*, *ragged arrays*, and *rectangular arrays*. For each of these families of data structures, the number of memory modules needed to achieve conflict tolerance  $c$  is determined to within constant factors.

**Index Terms**—Parallel memory systems, data structures for parallel systems, bounded-conflict parallel memory access, data mapping, parallel architectures, parallel systems, data structures, graph labeling.

## 1 INTRODUCTION

WE study the efficient mapping of data structures onto a parallel memory system (PMS, for short) composed of several *modules* that can be accessed simultaneously (say, by the processors of a multiprocessor system). Mapping a data structure onto a PMS poses a challenge to an algorithm designer because such systems typically are single-ported: They queue up simultaneous accesses to the same memory module, thereby incurring delay. The effective use of a PMS therefore demands efficient mapping strategies for the data structures that one wishes to access in parallel—strategies that minimize, for each memory access, the delay incurred by this queuing. Consequently, the designer must allocate the data structure in a way that minimizes the number of conflicts for any set of  $n$  items (from the data structure) that will possibly be accessed simultaneously. Obviously, different mapping strategies are needed for different data structures as well as for different ways of accessing the same structure. As two simple examples of the second point: 1) One would map the nodes of a complete binary tree quite differently in order to optimize access to levels of the tree rather than to optimize access to root-to-leaf paths. 2) One would map the nodes of a rectangular array quite differently in order to optimize access to rows or columns of the array rather than to optimize access to block subarrays.

The preceding considerations give rise to the following problem: Given a data structure represented by a graph and a repertoire of subgraph “shapes” (which we call *templates*) that one wants to access easily, we wish to design a memory-mapping strategy for the nodes of the structure, which minimizes the number of simultaneous requests to a single memory module, over all instances of the considered templates. The strategy should be also time-efficient; i.e., the time to evaluate the (address of the) memory module where a node is mapped should be minimized lest it become a bottleneck to the parallel access to instances of the data structure, thereby limiting the overall parallel speedup of the application.

We concentrate on four important families of templates: (*rooted*) *binary trees* and three genres of array-like structures termed *chaotic arrays*, *ragged arrays*, and *rectangular arrays* (cf. [22], [23], [24]). We focus on sets of templates suggested by application areas as diverse as relational databases, sparse-matrix calculations, and computer vision.

**Our contributions.** We study the PMS-mapping problem via a graph-theoretic abstraction that allows us to partition our problem into two logical phases. In phase 1, we find, for each of our families of templates, a single structure/graph that “almost” contains as a subgraph every one of our desired templates. If “almost” were replaced by “exactly” in the preceding sentence, then the graph we find would be *perfect-universal*, in the sense of [8]. The qualifier “almost” refers to our allowing a bounded number of nodes of a subgraph to map to the same node of the universal graph—reflecting our allowing a bounded number of conflicting accesses to each module in our memory-access problem. In phase 2 of our solution, we find efficient storage schemes for our “almost” perfect-universal graphs. Thus, by establishing bounds on the sizes of graphs that are “almost” perfect-universal for our four families of templates, we

- G. Cordasco and V. Scarano are with the Dipartimento di Informatica ed Applicazioni, “R.M. Capocelli,” Università di Salerno, 84084, Fisciano (SA)-Italy. E-mail: {cordasco, vitsca}@dia.unisa.it.
- A.L. Rosenberg is with the Department of Computer Science, University of Massachusetts Amherst, Amherst, MA 01003. E-mail: rsnbrg@cs.umass.edu.

Manuscript received 3 Aug. 2005; revised 2 May 2006; accepted 30 June 2006; published online 9 Jan. 2007.

Recommended for acceptance by D. Trystram.

For information on obtaining reprints of this article, please send e-mail to: [tpds@computer.org](mailto:tpds@computer.org), and reference IEEECS Log Number TPDS-0354-0805. Digital Object Identifier no. 10.1109/TPDS.2007.1024.

TABLE 1  
The Numbers of Nodes in Data Structures That Give Parallel Access to All Templates Having  $\leq n$  Nodes

Data Structure	Template	Lower Bound	Upper Bound
complete binary trees	rooted binary trees	$2^{\lceil \frac{n+1}{c+1} - \frac{1}{3} \rceil}$	$(2 + 2c^{1-1/c})2^{\frac{n+1}{c+1}} + O(n)$
two-dimensional arrays	chaotic arrays	$\frac{1}{c} \left( \lfloor \frac{n}{2} \rfloor \cdot \lfloor \frac{n}{2(c+1)} \rfloor \right)$	$\left( \frac{n+1}{c} \right)^2$
two-dimensional arrays	ragged arrays	$\frac{1}{c} \left( \lfloor \frac{n}{2} \rfloor \cdot \lfloor \frac{n}{2(c+1)} \rfloor \right)$	$\frac{n(n+1)}{c(c+1)} + n$
two-dimensional arrays	rectangular arrays	$\lceil n/c \rceil$	$\lceil n/c \rceil$

obtain bounds on the efficiency of PMS storage mappings for the families that admit a broader range of templates than does earlier work. For instance, we present the first strategy for mapping binary trees onto parallel memories in a way that allows one to access *any* rooted subtree with a bounded number of conflicts. Our upper and lower bounds on the sizes of “almost” perfect-universal graphs are tight, to within constant factors (Section 3), which leads to similarly tight results for the parallel-memory mapping problem (Section 5). Our main results are summarized in Table 1.

**Related work.** Research on the parallel-memory mapping problem for arrays originated with strategies for mapping two-dimensional arrays into parallel memories. In fact, one can view Euler’s “latin square” construction [18] as the very first strategy for mapping a square matrix onto a parallel memory in a way that allows rows, columns, and diagonals to be accessed without conflicts. Research has since moved on to seeking mapping algorithms for arrays that extend the “versatility” of latin squares by allowing conflict-free access for other templates, such as blocks and diagonals [26]. Several schemes have been proposed [5], [6], [13], [19], [21] that offer conflict-free access to several templates simultaneously, including rows, columns, diagonals, and submatrices. While the strategies in these sources provide conflict-free mappings, such was not their primary goal; they were actually designed to accommodate as many templates as possible. Strategies for mapping tree-like structures have a more recent history. Early research [12], [13], [15], [29] mainly considered conflict-free access for one *elementary* template—either complete subtrees or root-to-leaf paths or levels—at a time. The current research emulates the cited array-related sources in seeking “maximally versatile” mapping strategies. Several sources [1], [2], [3] offer strategies for accessing any instance of the three elementary templates or combinations thereof. These sources also offer strategies that scale with the number of memory modules so that the number of available modules changes with the problem instance. The only study that approaches the generality of our result—access to *any* subtree—is the C-template (“C” standing for “composite”) of [2], whose instances are combinations of different numbers of distinct elementary templates. The mapping strategy presented for C-template instances of size  $K$ , with  $M$  memory modules, achieves  $O(K/M + c)$  conflicts. In [14], [17], [16], the hypercube data structure has been investigated with regard to different templates. Similar objectives to ours are found in the work on parallel *secondary* memories (i.e., disks) in [27], [28]; however, their work differs from ours in the nature of the devices (which offer block-access rather than cell-access) as well as their

emphasis on efficient algorithms rather than generic, efficiently accessed data structures. Finally, one finds in [20], [25] techniques for achieving storage-conservative, bounded-conflict access to a variety of data structures; their storage-conservativeness is analogous to our small universal graphs and their bounded-conflict access is exactly analogous to ours; however, their focus is on element-wise access to data structures, rather than template-wise.

## 2 PRELIMINARIES

### 2.1 Basic Definitions

#### 2.1.1 Four Families of Templates

For any data structure  $\mathcal{D}$ , let  $\mathcal{G}_{\mathcal{D}}$  be the graph whose adjacencies describe those of  $\mathcal{D}$ . For any graph  $G = (V^G, E^G)$ : Size  $G = |V^G|$ , the number of nodes of  $G$ .

A **template**  $t$  for a data structure  $\mathcal{D}$  is any subgraph of  $\mathcal{G}_{\mathcal{D}}$ . Each subgraph of  $\mathcal{G}_{\mathcal{D}}$  that is isomorphic to  $t$  is an *instance* of template  $t$ . Informally, a template describes a pattern for accessing the nodes of  $\mathcal{D}$ . The templates that we consider here are more “ambitious” than those found in most earlier sources. Specifically, when we consider memory mappings for trees, we allow all rooted binary trees as templates; when we consider memory mappings for any of our three genres of array, we allow all “origin-connected” chaotic arrays as templates.

Our study concentrates on the four families of templates depicted in Fig. 1.

A **rooted binary tree**  $T = (V^T, E^T)$  is a graph whose node-set  $V^T$  is comprised of a finite *prefix-closed* set of binary strings. Let  $x$  be an arbitrary (possibly null) binary string and  $\beta$  an arbitrary bit:  $\beta \in \{0, 1\}$ . By “prefix-closed,” we mean that  $x \in V^T$  whenever  $x\beta \in V^T$ . There is an edge between each node  $x \in V^T$  and node  $x\beta$ ; all edges of  $T$  are of this form.  $T$ ’s *root* is  $\lambda$ , the null string. For each  $v \in V^T$ , we denote by  $l(v)$  the *level* of  $v$ , which is the length of  $v$  as a binary string or, equivalently, the distance between  $v$  and the root. The  $i$ th *level* of  $T$  is the set  $L_i^{(T)} = \{v \in V^T | l(v) = i\}$ .  $\text{Hgt}(T)$  denotes the *height* of  $T$ , which is the number of levels. Nonroot nodes of  $T$  that are incident to only one edge are *leaves*. The tree  $T$  is *complete* if all leaf-to-root paths have the same length or, equivalently, if each level  $l$  of  $T$  has exactly  $2^l$  nodes. We denote by  $\mathbf{T}$  the family of rooted binary trees.

A **(two-dimensional) chaotic array**  $C$  is an undirected graph whose node-set  $V^C$  is a subset of  ${}^1 N \times N$  that is *order-closed*, in the sense that, for each node  $(r, s) \neq (0, 0)$  of  $C$ , the

1.  $N$  denotes the nonnegative integers. For each  $n \in N$ ,  $[n]$  denotes the set  $\{0, 1, \dots, n-1\}$ .

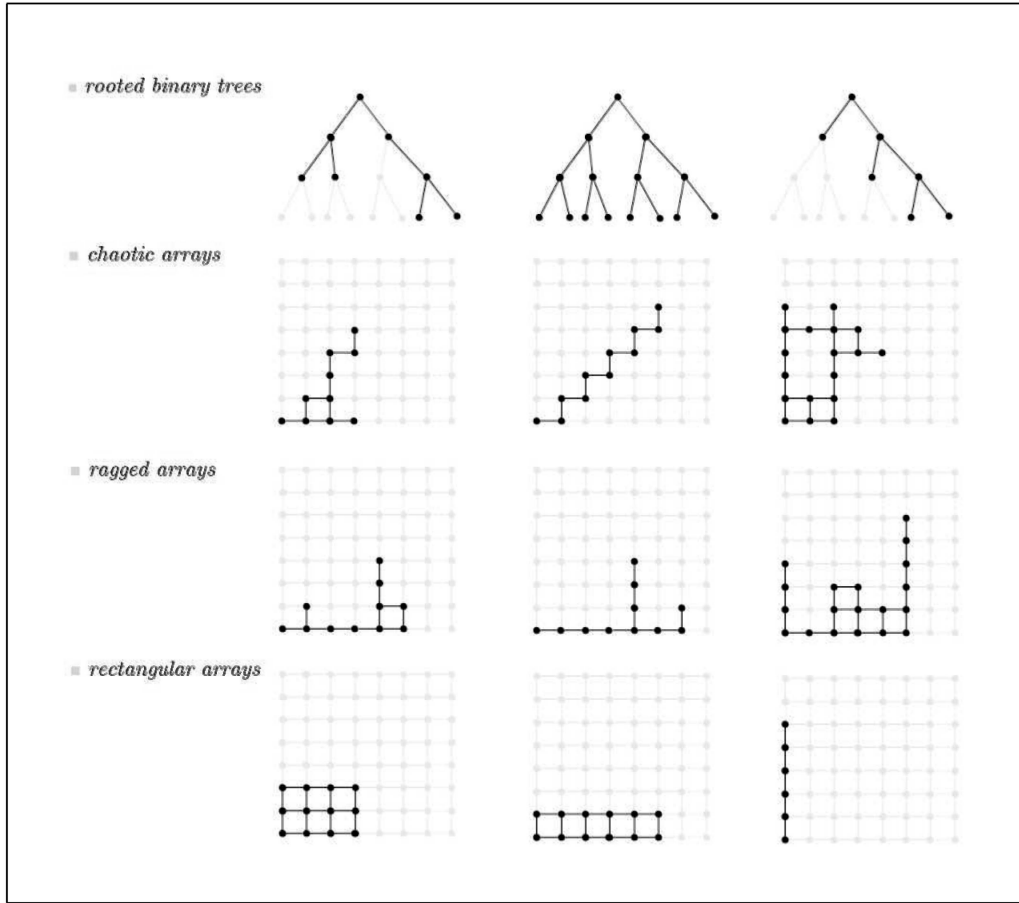


Fig. 1. Our four families of templates.

set  $V^C \cap \{\langle r-1, s \rangle, \langle r, s-1 \rangle\} \neq \emptyset$ . Node  $\langle 0, 0 \rangle$  is the bottom-left grid corner.  $C$ 's edges are comprised of all two-element subsets of  $V^C$  of the form  $\langle v, v + \alpha \rangle$ , where  $\alpha \in \{\langle 0, 1 \rangle, \langle 1, 0 \rangle\}$ . We denote by  $\mathbf{C}$  the family of two-dimensional chaotic arrays.

A **(two-dimensional) ragged array**  $R$  is a chaotic array whose node-set  $V^R$  satisfies the following:

- If  $\langle v_1, v_2 \rangle \in V^R$ , then  $\{v_1\} \times [v_2] \subseteq V^R$ .
- If  $\langle v_1, 0 \rangle \in V^R$ , then  $[v_1] \times \{0\} \subseteq V^R$ .

We denote by  $\mathbf{R}$  the family of two-dimensional ragged arrays.

A **(two-dimensional) rectangular array**  $S$  is a ragged array whose node-set has the form  $[a] \times [b]$  for some  $a, b \in \mathbb{N}$ . We denote by  $\mathbf{S}$  the family of two-dimensional rectangular arrays.

For  $v = \langle v_1, v_2 \rangle \in \mathbb{N} \times \mathbb{N}$ , we define  $\Sigma(v) = v_1 + v_2$ . The  $i$ th diagonal of an array<sup>2</sup>  $A = (V^A, E^A)$  is the set  $D_i^{(A)} \stackrel{\text{def}}{=} \{v \in V^A \mid \Sigma(v) = i\}$ .

For any binary tree  $T$ ,  $\text{Size}(T, i) = |L_i^{(T)}| \leq 2^i$ ; for any genre of array  $A$ ,  $\text{Size}(A, i) = |D_i^{(A)}| \leq i + 1$ .

### 2.1.2 Templates and Their Contractions

Letting  $\mathbf{X}$  ambiguously denote  $\mathbf{T}$ ,  $\mathbf{C}$ ,  $\mathbf{R}$ , or  $\mathbf{S}$ , we denote by  $\mathbf{X}_n$  the family of templates of size  $\leq n$ , i.e.,  $\text{Size}(\mathbf{X}_n) \leq n$ .

2.  $A$  can be a chaotic or ragged or rectangular array.

For any integer  $c > 0$ , a  $c$ -contraction of a template  $X$  is a graph  $H$  that is obtained from  $X$  as follows (see Fig. 2):

1. Rename  $X$  as  $H^{(0)}$ . Set  $k = 0$ .
2. Pick a set  $Y$  of  $\leq c$  nodes of  $H^{(k)}$  that were nodes of  $X$ . Replace these nodes by a single node  $v_Y$ ; replace all edges of  $H^{(k)}$  that are incident to nodes of  $Y$  by edges that are incident to  $v_Y$ . The graph so obtained is  $H^{(k+1)}$ .
3. Iterate Step 2 some number of times,  $\ell$ ;  $H$  is the final  $H^{(\ell)}$ .

We denote by  $H^*(X)$  the set of all the  $c$ -contractions of a generic template  $X$ . We seek an algorithm that produces a *conflict-free* memory-mapping for a graph  $H \in H^*(X)$ . This mapping for  $H$  easily yields a  $c$ -*conflict-free* memory-mapping for  $X$ .

## 2.2 c-Perfect-Universal Graphs and Our Results

### 2.2.1 c-Perfect-Universality

A graph  $G_c = (V_c, E_c)$  is  $c$ -**perfect-universal** for the family  $\mathbf{X}_n$  if, for each  $X = (V^X, E^X)$  in the family, there exists a  $c$ -contraction  $I_X = (V, E) \in H^*(X)$  that is a *labeled subgraph* of  $G_c$ . By this we mean that, for any template  $X \in \mathbf{X}_n$ , the fact that  $I_X$  is a subgraph of  $G_c$  is witnessed by a mapping  $f : V \rightarrow V_c$  for which each  $v \in V$  is a subset of  $V^X$ ; cf. Fig. 3.

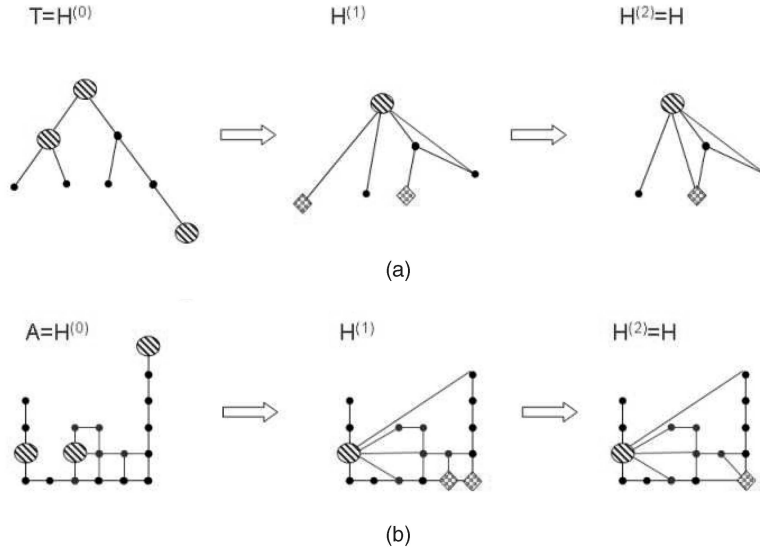


Fig. 2. Illustrating the notion of contraction.

We leave to the reader the simple specialization of the notion of  $c$ -perfect-universality to the families  $\mathbf{T}_n$ ,  $\mathbf{C}_n$ ,  $\mathbf{R}_n$ , and  $\mathbf{S}_n$ .

The notion of (1-)perfect-universality (of more general graph families) was invented in [8] to unite the well-studied areas of universal graphs [4], [7] and perfect hashing [11], toward the end of studying efficient storage mappings for important families of data structures. A perfect-universal graph for  $\mathbf{T}_n$  (respectively, for  $\mathbf{C}_n$ ,  $\mathbf{R}_n$ , or  $\mathbf{S}_n$ ) can be viewed as a way to map each node of a binary tree (respectively, of a chaotic array, ragged array, or rectangular array)  $X \in \mathbf{T}_n$  (respectively,  $\in \mathbf{C}_n$ ,  $\mathbf{R}_n$ , or  $\mathbf{S}_n$ ) to a distinct memory location in such a way that: No two graph-nodes share the same location; each path in  $X$  translates to a sequence of memory locations. A  $c$ -perfect-universal graph for  $\mathbf{T}_n$  (respectively, for  $\mathbf{C}_n$ ,  $\mathbf{R}_n$ , or  $\mathbf{S}_n$ ), where  $c > 1$ , plays a similar role, with  $\leq c$  nodes of  $X$  allowed to share the same memory location.

**Remark.** Intuitively, the nodes of a  $c$ -perfect-universal graph represent the modules of a parallel memory. The preservation of node labels ensures simple address calculations.

### 2.2.2 Bounds for Perfect-Universality and Bounded Perfect-Universality

The simplest perfect-universal graph for the family  $\mathbf{T}_n$  is the height- $n$  complete binary tree,  $\mathcal{T}_n$ . Similarly, the simplest perfect-universal graph for the family  $\mathbf{A}_n$  is the ragged array  $\mathcal{R}_n$  whose node-set is  $\{v \in N \times N \mid \Sigma(v) < n\}$ . The perfect-universality of  $\mathcal{T}_n$  for trees and of  $\mathcal{R}_n$  for arrays

is witnessed by the *identity map* of the nodes of any  $n$ -node tree to the nodes of  $\mathcal{T}_n$  and of the nodes of any  $n$ -node chaotic, ragged, or rectangular array to the nodes of  $\mathcal{R}_n$ . (For this reason, we call these simplest perfect-universal graphs *naive*.) Of course,  $\mathcal{T}_n$  and  $\mathcal{R}_n$  are quite “inefficient” in that they have, respectively,  $2^n - 1$  and  $n(n + 1)/2$  nodes, whereas each  $X \in \mathbf{X}_n$  has only  $n$  nodes. It is natural to wonder how much smaller a perfect-universal graph for  $\mathbf{T}_n$ ,  $\mathbf{C}_n$ ,  $\mathbf{R}_n$ , and  $\mathbf{S}_n$  can be. The *perfection number* of  $\mathbf{T}_n$  (respectively, of  $\mathbf{C}_n$ ,  $\mathbf{R}_n$ , and  $\mathbf{S}_n$ ), denoted  $\text{Perf}(\mathbf{T}_n)$  (respectively,  $\text{Perf}(\mathbf{C}_n)$ ,  $\text{Perf}(\mathbf{R}_n)$ , and  $\text{Perf}(\mathbf{S}_n)$ ), is the size of the smallest such graph for the indicated family. These numbers are determined exactly in [8].

**Theorem 1 [8].** For each integer  $n > 0$ ,

$$\begin{aligned} \text{Perf}(\mathbf{T}_n) &= (3 - (n \bmod 2))2^{\lfloor (n-1)/2 \rfloor} - 1, \\ \text{Perf}(\mathbf{C}_n) &= \lceil n/2 \rceil (\lfloor n/2 \rfloor + 1), \\ \text{Perf}(\mathbf{R}_n) &= \frac{1}{2} (\lfloor n/3 \rfloor + 1)(3\lceil 2n/3 \rceil - n), \\ \text{Perf}(\mathbf{S}_n) &= n. \end{aligned}$$

In this paper, we generalize the study of “perfect” memory mappings from [8] by allowing boundedly many *collisions* in such mappings. We thus relax the “one-to-one” demands of perfect-universality to “(boundedly many)-to-one.” For each integer  $c > 0$ , let  $\text{Perf}_c(\mathbf{X}_n)$  denote the  $c$ -collision analogue of  $\text{Perf}(\mathbf{X}_n)$ , so that  $\text{Perf}_c(\mathbf{X}_n)$  is the size of the smallest  $c$ -perfect-universal graph for  $\mathbf{X}_n$ . (Clearly,  $\text{Perf}_1(\mathbf{X}_n) = \text{Perf}(\mathbf{X}_n)$ .) Our study attempts to generalize Theorem 1 to  $c$ -perfect-universality. We present close upper and lower bounds on  $\text{Perf}_c(\mathbf{X}_n)$  for each of the four families of interest. Our upper bounds are supported by the first strategies for mapping structured data onto a parallel memory in such a way that any  $n$ -node binary tree, chaotic array, ragged array, or rectangular array can be accessed with at most  $c$  conflicts. We obtain the following bounds<sup>3</sup> which are verified in subsequent sections.

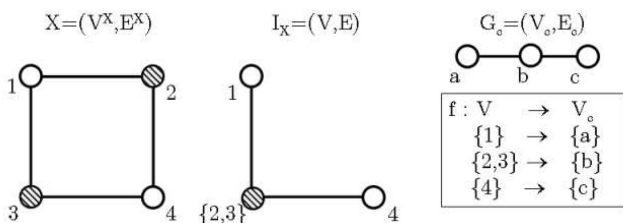


Fig. 3. An example of mapping.

3. To enhance the legibility of powers of 2 with complicated exponents, we often write  $\text{exp}2(X)$  for  $2^X$ .

**Theorem 2.** For all positive integers  $c$  and  $n$ :

$$\begin{aligned} \exp 2 \left( \left\lfloor \frac{n+1}{c+1} - \frac{11}{3} \right\rfloor \right) &< \text{Perf}_c(\mathbf{T}_n) \\ &< \left( 2 + 2c^{1-1/c} \right) \exp 2 \left( \frac{n+1}{c+1} \right) + O(n) \\ \frac{1}{c} \left( \left\lfloor \frac{n}{2} \right\rfloor \cdot \left\lfloor \frac{n}{2(c+1)} \right\rfloor \right) &\leq \text{Perf}_c(\mathbf{C}_n) \leq \left( \left\lfloor \frac{n+1}{c} \right\rfloor \right)^2 \\ \frac{1}{c} \left( \left\lfloor \frac{n}{2} \right\rfloor \cdot \left\lfloor \frac{n}{2(c+1)} \right\rfloor \right) &\leq \text{Perf}_c(\mathbf{R}_n) < \frac{n \cdot (n+1)}{c \cdot (c+1)} + n = \frac{n^2}{c^2} + \text{l.o.t.} \\ \text{Perf}_c(\mathbf{S}_n) &= \left\lfloor \frac{n}{c} \right\rfloor. \end{aligned}$$

### 2.2.3 An Overview of Our Proof Techniques

We extract the common structure of our proofs of the various components of Theorem 2 in an attempt to illustrate how our study can be extrapolated to other families of templates. Focus on a particular family  $\mathbf{X}_n$  of  $n$ -node graphs. In the present context,  $\mathbf{X}$  stands, in turn, for  $\mathbf{T}$ ,  $\mathbf{C}$ ,  $\mathbf{R}$ , or  $\mathbf{S}$ .

**Upper bounds.** The proofs of the upper bounds in Theorem 2 employ the following abstraction to construct explicit  $c$ -perfect-universal graphs for the subject family  $\mathbf{X}_n$ : We start with a naive, huge, perfect-universal graph  $\Xi_n$  for  $\mathbf{X}_n$  and then “contract”  $\Xi_n$  by identifying certain collections of its nodes. The mechanism for effecting the contraction will be an algorithm  $\mathcal{A}_c$  that colors the nodes of  $\Xi_n$  in such a way that the natural node-label-preserving embedding of any  $n$ -node structure  $X$  into  $\Xi_n$  (which witnesses  $X$  being a subgraph of  $\Xi_n$ ) never uses more than  $c$  nodes of any given color as homes for  $X$ 's nodes. When we identify (or, merge) each like-colored set of nodes of  $\Xi_n$ —i.e., contract each set to a single node in the obvious way—we obtain a  $c$ -perfect-universal graph  $G_c$ , whose size is (obviously) an upper bound on  $\text{Perf}_c(\mathbf{X}_n)$ . We craft the algorithm  $\mathcal{A}_c$  to implement this strategy via the following two steps:

1. We partition  $\Xi_n$  into blocks  $\Xi_{n,1}, \Xi_{n,2}, \dots, \Xi_{n,k}$  (usually via levels for trees and via diagonals for arrays).
2. We color the nodes of each block  $\Xi_{n,i}$ , using a unique set of colors,  $C_i$ , in a round-robin fashion.

Since  $\text{Size}(G_c) = \sum_i |C_i|$ , a crucial step in crafting  $\mathcal{A}_c$  is to make the sets  $C_i$  as small as possible while ensuring that  $G_c$  is  $c$ -perfect-universal for  $\mathbf{X}_n$ .

It is worth emphasizing that the technique allows an efficient assignment of colors to nodes since the coloring can be done in  $O(1)$  time.

**Lower bounds.** The proofs of the lower bounds in Theorem 2 employ the same abstraction as just described. In fact, one can see that any mapping strategy is equivalent to a coloring of the *naive* perfect-universal graph which, in turn, is equivalent to the contraction of the nodes with the same color. Then, imagine that one has devised an optimally small  $c$ -perfect-universal graph,  $G_n$ , for  $\mathbf{X}_n$  by suitably coloring the nodes of the naive perfect-universal  $\Xi_n$  and merging like-colored nodes. We bound the size of  $G_n$  as follows:

- We identify a subgraph  $R$  of  $\Xi_n$ .
- We show that, for each  $(c+1)$ -node subgraph  $U$  of  $R$ , there exists a particular template that 1) contains all the nodes in  $U$  and 2) has size  $\leq n$ .

The existence of  $R$  demonstrates that the only way to avoid incurring  $> c$  conflicts in our mappings is to avoid using any color more than  $c$  times while coloring the nodes in  $R$ . It follows that one must use  $\geq \lceil |R|/c \rceil$  colors, hence,  $\text{Perf}_c(\mathbf{X}_n) \geq \lceil |R|/c \rceil$ .

## 3 C-PERFECT-UNIVERSAL GRAPHS FOR TREES

We begin our study of  $c$ -perfect-universality with binary trees.

### 3.1 An Upper Bound on $\text{Perf}_c(\mathbf{T}_n)$

**Theorem 3.** For all integers  $n$  and  $c > 1$ ,

$$\text{Perf}_c(\mathbf{T}_n) < \left( 2 + 2c^{1-1/c} \right) 2^{(n+1)/(c+1)} + O(n). \quad (1)$$

We construct our  $c$ -perfect-universal graph  $G_c$  for  $\mathbf{T}_n$  via an algorithm  $\mathcal{A}_c$  that colors the nodes of  $\mathbf{T}_n$  (the naive perfect-universal graph for  $\mathbf{T}_n$ ) according to the algorithm described in Section 2.2.3.

Our first step in constructing  $\mathcal{A}_c$  is straightforward since we can use the levels of  $\mathbf{T}_n$  as the blocks of its partition. The remainder of the section is devoted to the second step, i.e., estimating how big the sets of colors  $C_i$  must be in order for  $G_c$  to be  $c$ -perfect-universal for  $\mathbf{T}_n$ . The desired bound will easily follow by summing the cardinalities of the  $C_i$ .

**Auxiliary results.** We begin by identifying some special subtrees of  $\mathbf{T}_n$ . Let  $m$  and  $i$  be integers such that  $4 \lceil \log m \rceil \leq i \leq n$  and let  $x$  be a node at level  $i - \lceil \log m \rceil$  of  $\mathbf{T}_n$ . Consider the subtree  $T_{(i,m)}(x)$  of  $\mathbf{T}_n$  constructed as follows:

1. Generate the length- $(i - \lceil \log m \rceil - 1)$  path from the root of  $\mathbf{T}_n$  to the parent of node  $x$ .
2. Generate the smallest complete subtree rooted at  $x$  that has  $\geq m$  leaves; easily, this subtree—call it  $T(x)$ —has height  $\lceil \log m \rceil + 1$ .
3. Finally, prune the tree so constructed, removing all nodes and edges other than those needed to incorporate (or, “capture”) the leftmost  $m$  leaves.

Easily, every tree  $T_{(i,m)}$  (cf. Fig. 4a) has exactly  $m$  leaves which reside at level  $i$  of  $\mathbf{T}_n$ .

**Lemma 1.** The trees  $T_{(i,m)}$  are the smallest rooted subtrees of  $\mathbf{T}_n$  that have  $m$  leaves at level  $i$ .

**Proof.** For any level  $j$  of any binary tree  $T$ , we have  $\text{Size}(T, j-1) \geq \lceil \frac{1}{2} \text{Size}(T, j) \rceil$ . One verifies easily from our construction that the trees  $T_{(i,m)}$  achieve this bound with equality. In detail, for each level  $j \leq i$ , we have  $\text{Size}(T_{(i,m)}, j-1) = \lceil \frac{1}{2} \text{Size}(T_{(i,m)}, j) \rceil$ . Thus, even level by level, they are the smallest trees satisfying the premise of the lemma.  $\square$

By analyzing the structure of the trees  $T_{(i,m)}$  in more detail, we obtain close bounds on their sizes.

**Lemma 2.** For all  $i$  and  $m$ ,

$$2m + i - \lceil \log m \rceil - 1 \leq \text{Size}(T_{(i,m)}) \leq 2m + i - 1. \quad (2)$$

**Proof.** By construction, the  $m$  leaves of each tree  $T_{(i,m)}(x)$  are the *leftmost* leaves of  $T(x)$ . Consequently, these leaves belong to a succession of complete subtrees of  $T(x)$  of the following heights:  $h_0 = \lceil \log m \rceil$  (which accounts for  $2^{h_0}$  of

4. All logarithms are to the base 2.

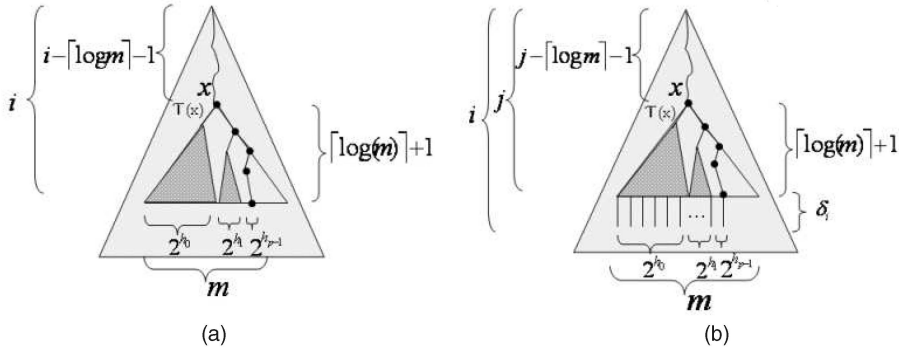


Fig. 4. (a)  $T_{(i,m)}(x)$ . (b)  $T_{(j,m)}(x)$  with the tentacles, where  $j = i - \delta_i$ .

the  $m$  leaves),  $h_1 = \lfloor \log(m - 2^{h_0}) \rfloor$  (which accounts for an additional  $2^{h_1}$  of the  $m$  leaves),  $h_2 = \lfloor \log(m - (2^{h_0} + 2^{h_1})) \rfloor$ , and so on; the successive  $h_j$  are the positions of the 1s in the binary representation of  $m$ ; say that there are  $p_m$  such positions. Since each complete subtree of height  $h_j$  is a copy of  $T_{h_j}$ , hence, it has precisely  $2^{h_j+1} - 1$  nodes, it is easy to evaluate the size of  $T_{(i,m)}(x)$ : We merely sum the sizes of the  $T_{h_j}$  plus the size of the various paths needed to “capture” them. Thus,

$$\begin{aligned}
 \text{Size}(T_{(i,m)}(x)) &= (i - h_0 + 1) + \text{Size}(T_{h_0}) + (h_0 - h_1) + \text{Size}(T_{h_1}) \\
 &\quad + (h_1 - h_2) + \text{Size}(T_{h_2}) + \cdots + (h_{p_m-2} - h_{p_m-1}) \\
 &\quad + \text{Size}(T_{h_{p_m-1}}) \\
 &= (i - h_{p_m-1} + 1) + \sum_{j=0}^{p_m-1} \text{Size}(T_{h_j}) \\
 &= (i - h_{p_m-1} + 1) + 2m - p_m.
 \end{aligned} \tag{3}$$

We obtain the bound of the lemma from the exact, but nonperspicuous, expression (3), coupled with the inequality,  $1 \leq h_{p_m-1} \leq \lfloor \log m \rfloor + 1 - (p_m - 1)$ , which follows from the fact that there is at least one 1 in the binary representation of  $m$  and that  $p_m \geq 1$ .  $\square$

Let us now number the nodes at each level of  $T_n$  from left to right and say that the *distance* between any two such nodes is the magnitude of the difference of their numbers.

**Lemma 3.** For any integers  $0 \leq i \leq n$  and  $0 \leq \delta_i < i$ , the size of the smallest rooted subtree of  $T_n$  that has  $m$  leaves at level  $i$  of  $T_n$ , each at distance  $\geq 2^{\delta_i}$  from the others, is  $\geq \text{Size}(T_{(i,m)}) + (m - 1)\delta_i$ .

**Proof.** If two nodes on level  $i$  are at distance  $\geq 2^{\delta_i}$  apart, then their least common ancestor in  $T_n$  must be at some level  $\leq i - (\delta_i + 1)$ . It follows that the smallest rooted subtree  $T$  of  $T_n$  that satisfies the premises of the lemma must consist of a copy of some  $T_{(i-\delta_i,m)}$ , with  $m$  leaves at level  $i - \delta_i$ , plus  $m$  node-disjoint paths (like “tentacles”) from that level down through each of the  $\delta_i$  levels  $i - \delta_i + 1, i - \delta_i + 2, \dots, i$  of  $T_n$ ; cf. Fig. 4b. The chain (3) therefore yields:

$$\begin{aligned}
 \text{Size}(T) &\geq m\delta_i + \text{Size}(T_{(i-\delta_i,m)}) \\
 &= m(\delta_i + 2) + (i - \delta_i - h_{p_m-1} - p_m + 1) \\
 &= (m - 1)\delta_i + \text{Size}(T_{(i,m)}).
 \end{aligned}$$

$\square$

**The proof of Theorem 1.** We now have the tools to proceed with our upper-bound proof. We propose, in the next two lemmas, two coloring schemes for  $\mathcal{A}_c$ , each inefficient on its own (in the sense of yielding an upper bound on  $\text{Perf}_c(T_n)$  that comes close to matching our lower bound), but which combine to yield an efficient coloring scheme.

**Lemma 4.** Let  $T_n$  be colored by Algorithm  $\mathcal{A}_c$  using  $2^{\delta_i}$  colors at each level  $i$ . If each

$$2^{\delta_i} \geq \kappa_i \stackrel{\text{def}}{=} \lceil 2^i / c \rceil, \tag{4}$$

then any rooted  $n$ -node subtree of  $T_n$  engenders at most  $c$  collisions.

We leave to the reader the simple proof of Lemma 4, in addition to the proof that the lemma’s coloring scheme yields a  $c$ -perfect-universal graph for  $T_n$  that is roughly one power of  $c$  bigger than our lower bound suggests is necessary.

**Lemma 5.** Let  $T_n$  be colored by Algorithm  $\mathcal{A}_c$  using  $2^{\delta_i}$  colors at each level  $i$ . If each

$$2^{\delta_i} \geq \lambda_i \stackrel{\text{def}}{=} \frac{1}{4} \exp 2 \left( \left\lceil \frac{n + \log(c + 1) - i}{c} \right\rceil \right), \tag{5}$$

then any rooted  $n$ -node subtree of  $T_n$  engenders at most  $c$  collisions.

**Proof.** We show that, if a subtree  $T$  of  $T_n$  engenders more than  $c$  collisions, then  $\text{Size}(T) > n$ . Note that each collision that occurs must involve nodes from the same level of  $T_n$  because  $\mathcal{A}_c$  uses a distinct set of colors at each level. Let us, therefore, consider a shallowest tree  $T$  that engenders  $c + 1$  collisions at level  $i$  of  $T_n$ . Easily, the offending tree  $T$  has  $c + 1$  leaves from level  $i$  of  $T_n$  and, by the design of Algorithm  $\mathcal{A}_c$ , these leaves must be at distance  $2^{\delta_i}$  from one another. We can therefore combine Lemma 3, the lower bound of (2) (both with  $m = c + 1$ ), and (5) to bound from below the size of the offending tree  $T$ .

$$\begin{aligned}
 \text{Size}(T) &\geq \text{Size}(T_{(i,m)c}) + c\delta_i \\
 &\geq 2(c + 1) + i - \lfloor \log(c + 1) \rfloor - 1 \\
 &\quad + c \left\lceil \frac{n - 2c + \log(c + 1) - i}{c} \right\rceil \geq n + 1.
 \end{aligned}$$

Since we are interested only in  $n$ -node subtrees of  $T_n$ , the lemma follows.  $\square$

A bit of analysis verifies that the respective strengths and weaknesses of the coloring schemes of Lemmas 4 and 5 are mutually complementary. The  $\kappa_i$  of the former lemma work well on the small, lower-numbered levels of  $\mathcal{T}_n$ , but are too profligate with colors on the large, higher-numbered levels; the  $\lambda_i$  of the latter lemma work well on the higher-numbered levels of  $\mathcal{T}_n$ , but not on the lower-numbered ones. This complementarity suggests the ploy of using the  $\kappa_i$ -scheme to color the “top” of  $\mathcal{T}_n$  and the  $\lambda_i$ -scheme to color the “bottom.” A natural place to divide the “top” of  $\mathcal{T}_n$  from the “bottom” would be at a level  $i$ , where  $\kappa_i \approx \lambda_i$ . Using this intuition, we choose level

$$i^* \stackrel{\text{def}}{=} \left\lceil \frac{n-c+1}{c+1} + \log c \right\rceil \quad (6)$$

to be the first level of the “bottom” of  $\mathcal{T}_n$ . (Since  $c \leq n$ , trivial calculations show that  $n > i^*$ .) Using our hybrid coloring scheme, we end up with a  $c$ -perfect-universal graph  $G_c$  such that  $\text{Size}(G_c) = \sum_{j=0}^{i^*-1} \kappa_j + \sum_{k=i^*}^{n-1} \lambda_k$ . Evaluating the two summations in turn, we find the following, under the assumption that  $c > 1$  (since the case  $c = 1$  is dealt with definitively in [8]; see Theorem 1):

$$\begin{aligned} \sum_{j=0}^{i^*-1} \kappa_j &= \sum_{j=0}^{i^*-1} \lceil 2^j/c \rceil \leq \sum_{j=0}^{i^*-1} (2^j/c + 1) = \frac{1}{c} 2^{i^*} + i^* - \frac{1}{c} \\ &\leq \frac{1}{c} \exp 2 \left\lceil \frac{n-c+1}{c+1} + \log c \right\rceil + O(n) \\ &< 2 \cdot 2^{(n+1)/(c+1)} + O(n). \end{aligned} \quad (7)$$

$$\begin{aligned} \sum_{k=i^*}^{n-1} \lambda_k &= \sum_{k=i^*}^{n-1} \exp 2 \left( \left\lceil \frac{n + \log(c+1) - k}{c} \right\rceil - 2 \right) \\ &< \frac{(c+1)^{1/c}}{2} \cdot \sum_{k=i^*}^{n-1} 2^{(n-k)/c} < 2 \sum_{k=i^*}^{i^*+c-1} 2^{(n-k)/c} \\ &\leq 2c \cdot \exp 2 \left( \frac{n}{c} - \left( \frac{n-c+1}{c(c+1)} + \frac{\log c}{c} \right) \right) \\ &< 2c^{1-1/c} \cdot 2^{(n+1)/(c+1)}. \end{aligned} \quad (8)$$

$$(9)$$

The only subtlety in the preceding estimation of  $\sum_k \lambda_k$  is the step leading to estimate (8); that step is justified as follows:

$$\begin{aligned} &\sum_{k=i^*}^{n-1} 2^{(n-k)/c} \\ &= \left[ \exp 2 \left( \frac{n-i^*}{c} \right) + \cdots + \exp 2 \left( \frac{n-(i^*+c-1)}{c} \right) \right] \\ &\quad + \left[ \exp 2 \left( \frac{n-i^*}{c} - 1 \right) + \cdots + \exp 2 \left( \frac{n-(i^*+c-1)}{c} - 1 \right) \right] \\ &\quad + \left[ \exp 2 \left( \frac{n-i^*}{c} - 2 \right) + \cdots + \exp 2 \left( \frac{n-(i^*+c-1)}{c} - 2 \right) \right] \\ &\quad + \cdots + \left[ 2 + \cdots + 2^{\frac{1}{c}} \right] \\ &< 2 \cdot \left[ \exp 2 \left( \frac{n-i^*}{c} \right) + \cdots + \exp 2 \left( \frac{n-(i^*+c-1)}{c} \right) \right] \\ &= 2 \cdot \sum_{k=i^*}^{i^*+c-1} 2^{(n-k)/c}. \end{aligned}$$

The bounds (7) and (9) yield the claimed upper bound (1) on  $\text{Perf}_c(\mathbf{T}_n)$ .  $\square$

### 3.2 A Lower Bound on $\text{Perf}_c(\mathbf{T}_n)$

**Theorem 4.** For all  $c > 1$  and all  $n$ ,

$$\text{Perf}_c(\mathbf{T}_n) > \exp 2 \left( \left\lceil \frac{n+1}{c+1} - \frac{11}{3} \right\rceil \right). \quad (10)$$

**Proof.** Following the proof scheme of Section 2.2.3, we identify a region  $R$  of  $\mathcal{T}_n$  and construct, for each  $(c+1)$ -node subset  $U \in R$ , a tree with  $\leq n$  nodes, that contains all the nodes in  $U$ . The desired region  $R$  is the leftmost  $(c+1)2^{\kappa^*}$  nodes on level  $i^*$  (see (6)) of  $\mathcal{T}_n$ , where  $\kappa^* \stackrel{\text{def}}{=} \left\lceil \frac{n+1}{c+1} - \frac{11}{3} \right\rceil$ . For any subset  $(c+1)$ -node  $U \in R$ , we construct a binary tree  $\hat{T}_U$  that has  $\leq n$  nodes and that covers  $U$ . Easily,  $i^* - \kappa^* - \lceil \log(c+1) \rceil > 0$ .

Let  $r$  be the leftmost node on level  $i^* - \kappa^* - \lceil \log(c+1) \rceil$  of  $\mathcal{T}_n$ . To build  $\hat{T}_U$ , we start with the “base” tree  $T_{(i^*-\kappa^*, c+1)}(r)$ —which is the instance of  $T_{(i,m)}$  whose leaves are the first  $c+1$  nodes on level  $i^* - \kappa^*$  of  $\mathcal{T}_n$ ; we then add all the paths (call them “tentacles”) that connect the “base” tree’s leaves to all the  $c+1$  nodes of  $U$ . (Depending on the nodes in  $U$ , the “tentacles” may not be paths; but, importantly, they collectively have  $\leq (c+1)\kappa^*$  nodes). We see that  $\text{Size}(\hat{T}_U) \leq n$  as follows:

$$\begin{aligned} \text{Size} \hat{T}_U &\leq \text{Size}(T_{(i^*-\kappa^*, c+1)}(r)) + (c+1)\kappa^* \leq 2c+1 + i^* + c\kappa^* \\ &= 2c+1 + \left\lceil \frac{n-c+1}{c+1} + \log c \right\rceil + c \left\lceil \frac{n+1}{c+1} - \frac{11}{3} \right\rceil \\ &\leq n + \log c + \frac{1}{c+1} + 2 - \frac{5}{3}c. \end{aligned} \quad (11)$$

The quantity (11) is  $\leq n$  because  $5c \geq 3 \log c + 1/(c+1) + 2$  whenever  $c > 1$ .

As described in Section 2.2.3, we need  $\geq \lceil |R|/c \rceil$  colors to color the nodes of  $R$ , hence

$$\begin{aligned} \text{Perf}_c(\mathbf{T}_n) &> \left\lceil \frac{(c+1) \exp 2 \left( \left\lceil \frac{n+1}{c+1} - \frac{11}{3} \right\rceil \right)}{c} \right\rceil \\ &> \exp 2 \left( \left\lceil \frac{n+1}{c+1} - \frac{11}{3} \right\rceil \right). \end{aligned}$$

$\square$

## 4 c-PERFECT-UNIVERSAL GRAPHS FOR ARRAY-LIKE STRUCTURES

We turn now to our array-like structures.

### 4.1 c-Perfect-Universal Graphs for $C_n$ and $R_n$

#### 4.1.1 An Upper Bound on $\text{Perf}_c(C_n)$

**Theorem 5.** For all integers  $n$  and  $c$ ,

$$\text{Perf}_c(C_n) \leq \left\lceil \frac{n+1}{c} \right\rceil^2. \quad (12)$$

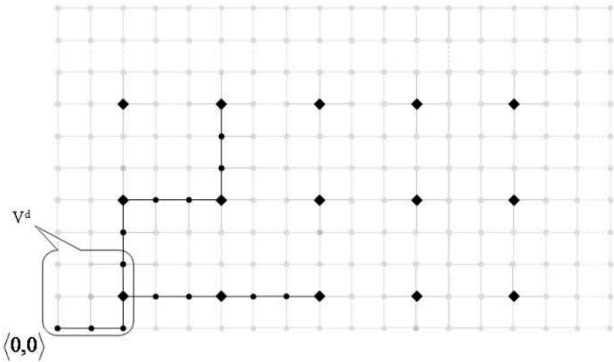


Fig. 5. A sample  $C \in \mathbf{C}_n$  with  $\leq 5$  collisions. Note that all nodes of  $V_d$  have distinct colors.

We derive our upper bound on  $\text{Perf}_c(\mathbf{C}_n)$  by constructing a graph  $G_c$  that is  $c$ -perfect-universal for  $\mathbf{C}_n$ . As described in Section 2.2.3, our construction builds on an algorithm  $\mathcal{A}_c$  that colors the nodes of the naive perfect graph  $\mathcal{R}_n$  for  $\mathbf{C}$ . Note that, following the first step in Section 2.2.3,  $\mathcal{R}_n$  is used as a trivial (one-block) partition for itself. For each node  $v \in \mathcal{R}_n$ , we denote by  $\text{color}(v)$  the color assigned to  $v$  by  $\mathcal{A}_c$ . Let  $d = \lceil (n+1)/c \rceil$  and let  $V_d = \{\langle v_1, v_2 \rangle \in \mathcal{R}_n \mid v_1 < d \text{ and } v_2 < d\}$ . Algorithm  $\mathcal{A}_c$  assigns a color to each  $\langle v_1, v_2 \rangle \in \mathcal{R}_n$  as follows:

- if  $v = \langle v_1, v_2 \rangle \in V_d$ , then  $\text{color}(v) = v_1 \cdot d + v_2$ ;  
 if  $v = \langle v_1, v_2 \rangle \notin V_d$ , then  $\text{color}(v) = \text{color}(\langle v_1 \bmod d, v_2 \bmod d \rangle)$ .

Simple counting and reasoning yield the following observations:

**Observation 1.** Algorithm  $\mathcal{A}_c$  uses exactly  $d^2 = \lceil (n+1)/c \rceil^2$  colors to color  $\mathcal{R}_n$ .

Let  $\delta(u, v)$  denote the (Manhattan) distance between nodes  $u = \langle u_1, u_2 \rangle$  and  $v = \langle v_1, v_2 \rangle$  of  $N \times N$ , i.e.,  $\delta(u, v) = |v_1 - u_1| + |v_2 - u_2|$ .

**Observation 2.** If  $\mathcal{A}_c$  assigns the same color to nodes  $u, v \in \mathcal{R}_n$ —i.e., if  $\text{color}(u) = \text{color}(v)$ —then  $\delta(u, v) \geq d$ .

**Proof of Theorem 5.** We claim that the coloring of  $\mathcal{R}_n$  by Algorithm  $\mathcal{A}_c$  illustrates how to embed any  $C \in \mathbf{C}_n$  into  $\mathcal{R}_n$  with at most  $c$  collisions. Indeed, if a chaotic array  $C \in \mathbf{C}_n$  cannot be so embedded, then its size must exceed  $n$ . An easy generalization of Observation 2 shows that any connected set of  $c+1$  like-colored nodes created by Algorithm  $\mathcal{A}_c$  must have at least  $c \cdot \lceil (n+1)/c \rceil > n$  nodes; cf. Fig. 5. An appeal to Observation 1 now establishes the claimed bound on  $\text{Perf}_c(\mathbf{C}_n)$ .  $\square$

#### 4.1.2 An Upper Bound on $\text{Perf}_c(\mathbf{R}_n)$

**Theorem 6.** For all integers  $n$  and  $c$ ,

$$\text{Perf}_c(\mathbf{R}_n) < \frac{n \cdot (n+1)}{c \cdot (c+1)} + n = \frac{n^2}{c^2} + \text{l.o.t.} \quad (13)$$

**Proof.** We derive our upper bound on  $\text{Perf}_c(\mathbf{R}_n)$  by constructing a graph  $G_c$  that is  $c$ -perfect-universal for  $\mathbf{R}_n$ . As described in Section 2.2.3, we construct  $G_c$  via an algorithm  $\mathcal{A}_c$  that colors the nodes of  $\mathcal{R}_n$  (the naive

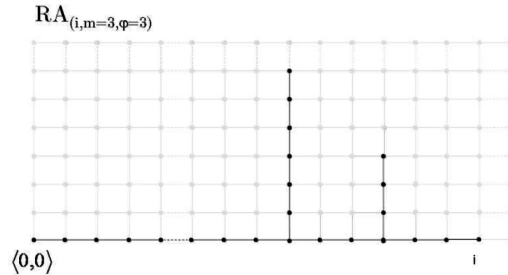


Fig. 6.  $RA_{(i,m=3,\varphi=3)}$ .

perfect-universal graph for  $\mathbf{R}_n$ ). In this case,  $\mathcal{A}_c$  partitions  $\mathcal{R}_n$  along diagonals.  $\square$

The remainder of the section is devoted to estimating how big the sets  $C_i$  must be in order for  $G_c$  to be  $c$ -perfect-universal for  $\mathbf{R}_n$ .

We number the nodes of each diagonal of  $\mathcal{R}_n$  from the northwest to the southeast; say that the *distance* between any two such nodes is the magnitude of the difference of their numbers.

We begin by considering the following ragged sub-arrays of  $\mathcal{R}_n$ . For positive integers  $m, i$ , and  $\varphi$ , where  $(m-1) \cdot \varphi \leq i \leq n$ , the ragged array  $RA_{(i,m,\varphi)}$  consists of:

- the  $i+1$  nodes that connect node  $\langle 0,0 \rangle$  with node  $\langle i,0 \rangle$ ;
- for  $1 \leq j \leq m-1$ , the  $j\varphi$  nodes that connect nodes  $\langle i-j\varphi, 1 \rangle$  and  $\langle i-j\varphi, j\varphi \rangle$ .

Note (cf. Fig. 6) that  $RA_{(i,m,\varphi)}$  has  $m$  nodes on diagonal  $D_i^{(\mathcal{R}_n)}$ , each at distance  $\varphi$  from the others and none on diagonals  $D_j^{(\mathcal{R}_n)}$  for  $j > i$ . Easily,  $RA_{(i,m,\varphi)}$  has

$$i+1 + \sum_{i=1}^{m-1} i\varphi = \binom{m}{2} \cdot \varphi + i+1$$

nodes. The following source of our interest in the ragged arrays  $RA_{(i,m,\varphi)}$  is readily verified:

**Lemma 6.** For all integers  $i, m, \varphi > 0$ , where

$$(m-1) \cdot \varphi \leq i \leq n,$$

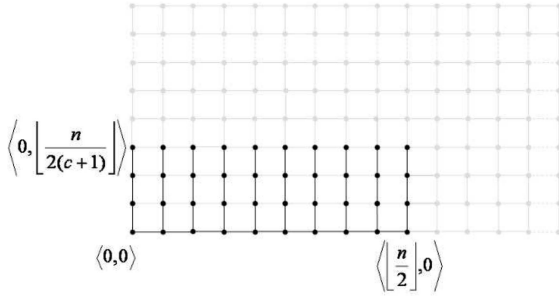
$RA_{(i,m,\varphi)}$  is the smallest ragged array  $R \in \mathbf{R}_n$  that has  $m$  nodes on diagonal  $D_i^{(\mathcal{R}_n)}$ , each at distance  $\varphi$  from the others.

**Lemma 7.** Let  $\mathcal{R}_n$  be colored by Algorithm  $\mathcal{A}_c$  using  $\mu_i$  colors on each diagonal  $i$ . If each

$$\mu_i \geq \nu_i \stackrel{\text{def}}{=} \left\lceil \frac{2(n-i)}{c \cdot (c+1)} \right\rceil, \quad (14)$$

then any  $R \in \mathbf{R}_n$  can be embedded into  $\mathcal{R}_n$  with at most  $c$  collisions.

**Proof.** We show that, if a ragged array  $R \in \mathbf{R}_n$  engenders more than  $c$  collisions, then  $\text{Size}(R) > n$ . Note that each collision that occurs must involve nodes from the same diagonal of  $\mathcal{R}_n$  because  $\mathcal{A}_c$  uses a distinct set of colors at each diagonal. Let us therefore consider a shallowest (in terms of number of diagonals touched) ragged array  $R$  that engenders  $c+1$  collisions at diagonal  $i$  of  $\mathcal{R}_n$ . Easily, the offending ragged array  $R$  has  $c+1$  nodes from

Fig. 7. The set of nodes of  $R$ .

diagonal  $i$  of  $\mathcal{R}_n$  and, by the design of Algorithm  $\mathcal{A}_c$  (see Section 2.2.3), these nodes must be at distance  $\mu_i$  from one another. We can therefore combine Lemma 6 (with  $m = c + 1$ ) and the bound (14) to bound from below the size of the offending ragged array  $R$ .

$$\begin{aligned} \text{Size}(R) &\geq \text{Size}(RA_{(i,c+1,\nu_i)}) = \nu_i \cdot \binom{c+1}{2} + i + 1 \\ &= \left\lfloor \frac{2(n-i)}{c \cdot (c+1)} \right\rfloor \cdot \left( \frac{c \cdot (c+1)}{2} \right) + i + 1 \\ &\geq n - i + i + 1. \end{aligned}$$

Since we are interested only in ragged arrays having  $\leq n$  nodes, the lemma follows.

Combining Lemmas 6 and 7, we thus have the following bound on the size of our  $c$ -perfect-universal graph  $G_c$ :

$$\begin{aligned} \text{Size}(G_c) &= \sum_{i=0}^{n-1} \nu_i = \sum_{i=0}^{n-1} \left\lfloor \frac{2(n-i)}{c \cdot (c+1)} \right\rfloor \\ &< n + \frac{2}{c \cdot (c+1)} \cdot \sum_{i=0}^{n-1} (n-i) \\ &= \left( \frac{n \cdot (n+1)}{c \cdot (c+1)} \right) + n. \end{aligned}$$

The claimed upper bound (13) follows.  $\square$

#### 4.1.3 A Lower Bound on $\text{Perf}_c(\mathbf{C}_n)$ and $\text{Perf}_c(\mathbf{R}_n)$

**Theorem 7.** For all positive integers  $c$  and  $n$ ,

$$\text{Perf}_c(\mathbf{C}_n) \geq \text{Perf}_c(\mathbf{R}_n) \geq \frac{1}{c} \left( \left\lfloor \frac{n}{2} \right\rfloor \cdot \left\lfloor \frac{n}{2(c+1)} \right\rfloor \right).$$

**Proof.** Since every ragged array is a chaotic array, we need concentrate only on the second inequality. Following the proof scheme of Section 2.2.3, we identify a region  $R$  of  $\mathbf{R}_n$  and construct, for each  $(c+1)$ -node subset  $U \in R$ , a template  $t_U \in \mathbf{R}_n$  that contains all the nodes in  $U$ . In this case, the desired region  $R$  is the set of all nodes  $\langle v_1, v_2 \rangle \in V^{\mathbf{R}_n}$  such that  $v_1 \leq \lfloor n/2 \rfloor$  and  $v_2 \leq \lfloor \frac{n}{2(c+1)} \rfloor$ ; cf. Fig. 7.

We describe a ragged (hence, chaotic) array  $\widehat{C}_U$ , with  $\text{Size}(\widehat{C}_U) \leq n$ , that contains each  $(c+1)$ -node subset  $U \in R$ .

One sees as follows that any set

$$U = \{x^{(1)}, x^{(2)}, \dots, x^{(c+1)}\}$$

of  $c+1$  nodes of  $R$  can coexist within the same  $n$ -node ragged array. To wit, if one send out paths to “capture” all of the  $x^{(i)}$ , then one ends up with a ragged array  $\widehat{C}_U$  of size

$$\text{Size}(\widehat{C}_U) \leq \lfloor n/2 \rfloor + \left( \left\lfloor \frac{n}{2(c+1)} \right\rfloor \cdot (c+1) \right) \leq n.$$

We conclude that the number of colors used to color the region  $R$  can be no smaller than  $\lceil |R|/c \rceil$ . Since  $|R| = (\lfloor n/2 \rfloor \cdot \lfloor n/(2(c+1)) \rfloor)$ , we thus have

$$\text{Perf}_c(\mathbf{R}_n) \geq \frac{1}{c} \left( \left\lfloor \frac{n}{2} \right\rfloor \cdot \left\lfloor \frac{n}{2(c+1)} \right\rfloor \right).$$

As noted earlier, this lower bound on  $\text{Perf}_c(\mathbf{R}_n)$  also holds automatically for  $\text{Perf}_c(\mathbf{C}_n)$ .  $\square$

## 4.2 $c$ -Perfect-Universal Graphs for $\mathbf{S}_n$

**Theorem 8.** For all integers  $n$  and  $c$ ,  $\text{Perf}_c(\mathbf{S}_n) = \lceil n/c \rceil$ .

**Proof (sketch).** The rectangular array with nodes  $[n-1] \times \{0\}$  indicates that  $\text{Perf}_c(\mathbf{S}_n) \geq \lceil n/c \rceil$ . Since, by Theorem 1,  $\text{Perf}(\mathbf{S}_n) = n$ , a simple array-“blocking” strategy shows that  $\text{Perf}_c(\mathbf{S}_n) \leq \lceil n/c \rceil$ .  $\square$

## 5 CONCLUSIONS

We have studied a new, highly flexible strategy for mapping complex templates to a parallel memory system, while allowing efficient access to an arbitrarily complex repertoire of access patterns. The strategy derives from extending the work in [8] on “perfect” storage representations to allow bounded numbers of memory collisions. We have illustrated the new strategy via four significant families of templates: binary trees and three genres of array-like structures. The repeating themes in our analysis techniques suggest that our strategy applies to a broad range of classes of data structures.

## ACKNOWLEDGMENTS

Early versions of portions of this work appeared at WDAS '03 [9] and Euro-Par '03 [10]. G. Cordasco and V. Scarano's research was supported in part by Italian FIRB project WebMinds. A.L. Rosenberg's research was supported in part by US National Science Foundation grant CCF-03-42417.

## REFERENCES

- [1] V. Auletta, S.K. Das, A. De Vivo, M.C. Pinotti, and V. Scarano, “Towards Universal Mapping Algorithms for Accessing Trees in Parallel Memory Systems,” *Proc. Int'l Parallel Processing Symp. (IPPS/SPDP)*, pp. 447-459, 1998.
- [2] V. Auletta, S.K. Das, A. De Vivo, M.C. Pinotti, and V. Scarano, “Optimal Tree Access by Elementary and Composite Templates in Parallel Memory Systems,” *IEEE Trans. Parallel and Distributed Systems*, vol. 13, 2002.
- [3] V. Auletta, A. De Vivo, and V. Scarano, “Multiple Template Access of Trees in Parallel Memory Systems,” *J. Parallel and Distributed Computing*, vol. 49, pp. 22-39, 1998.
- [4] S.N. Bhatt, F.R.K. Chung, F.T. Leighton, and A.L. Rosenberg, “Universal Graphs for Bounded-Degree Trees and Planar Graphs,” *SIAM J. Discrete Math.*, vol. 2, pp. 145-155, 1989.

- [5] P. Budnik and D.J. Kuck, "The Organization and Use of Parallel Memories," *IEEE Trans. Computers*, vol. 20, pp. 1566-1569, 1971.
- [6] C.J. Colbourn and K. Heinrich, "Conflict-Free Access to Parallel Memories," *J. Parallel Distributed Computing*, vol. 14, pp. 193-200, 1992.
- [7] F.R.K. Chung and R.L. Graham, "On Universal Graphs for Spanning Trees," *Proc. London Math. Soc.*, vol. 27, pp. 203-211, 1983.
- [8] F.R.K. Chung, A.L. Rosenberg, and L. Snyder, "Perfect Storage Representations for Families of Data Structures," *SIAM J. Algebraic Discrete Methods*, vol. 4, pp. 548-565, 1983.
- [9] G. Cordasco, A. Negro, A.L. Rosenberg, and V. Scarano, "c-Perfect Hashing Schemes for Arrays, with Applications to Parallel Memories," *Proc. Fifth Workshop Distributed Data and Structures*, 2003.
- [10] G. Cordasco, A. Negro, A.L. Rosenberg, and V. Scarano, "c-Perfect Hashing Schemes for Binary Trees, with Applications to Parallel Memories," *Proc. Int'l Conf. Parallel and Distributed Computing*, 2003.
- [11] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms*, second ed. MIT Press 1999.
- [12] R. Creutzburg and L. Andrews, "Recent Results on the Parallel Access to Tree-Like Data Structures—The Isotropic Approach," *Proc. Int'l Conf. Parallel Processing*, vol. 1, 1991, pp. 369-372, 1991.
- [13] S.K. Das and F. Sarkar, "Conflict-Free Data Access of Arrays and Trees in Parallel Memory Systems," *Proc. Sixth IEEE Symp. Parallel and Distributed Processing*, pp. 377-383, 1994.
- [14] S.K. Das and M.C. Pinotti, "Conflict-Free Access to Templates of Trees and Hypercubes in Parallel Memory Systems," *Proc. Int'l Conf. Computing and Combinatorics*, pp. 1-10, 1997.
- [15] S.K. Das, F. Sarkar, and M.C. Pinotti, "Conflict-Free Path Access of Trees in Parallel Memory Systems with Application to Distributed Heap Implementation," *Proc. 24th Int'l Conf. Parallel Processing*, vol. 3, pp. 164-167, 1995.
- [16] S.K. Das, M.C. Pinotti, and F. Sarkar, "Optimal and Load Balanced Mapping of Parallel Priority Queues in Hypercubes," *IEEE Trans. Parallel and Distributed Systems*, vol. 7, pp. 555-564, 1996.
- [17] S.K. Das, I. Finocchi, and R. Petreschi, "Star-Coloring of Graphs for Conflict-Free Access to Parallel Memory Systems," *Proc. 18th Int'l Parallel and Distributed Processing Symp.*, 2004.
- [18] L. Euler, "Recherches sur une Espèce de Carrés Magiques," *Commentationes Arithmeticae Collectae*, vol. II, pp. 302-361, 1849.
- [19] D.H. Lawrie, "Access and Alignment of Data in an Array Processor," *IEEE Trans. Computers*, vol. 24, pp. 1145-1155, 1975.
- [20] R.J. Lipton, A.L. Rosenberg, and A.C. Yao, "External Hashing Schemes for Collections of Data Structures," *J. ACM*, vol. 27, pp. 81-95, 1980.
- [21] K. Kim and V.K. Prasanna, "Latin Squares for Parallel Array Access," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, pp. 361-370, 1993.
- [22] D.L. Milgram and A. Rosenfeld, "Array Automata and Array Grammars," *Information Processing*, vol. 71, pp. 69-74, 1971.
- [23] A.L. Rosenberg, "Computed Access in Ragged Arrays," *Information Processing*, vol. 74, pp. 642-646, 1974.
- [24] A.L. Rosenberg, "On Storing Ragged Arrays by Hashing," *Math. Systems Theory*, vol. 10, pp. 193-210, 1976/1977.
- [25] A.L. Rosenberg and L.J. Stockmeyer, "Hashing Schemes for Extendible Arrays," *J. ACM*, vol. 24, pp. 199-221, 1977.
- [26] V. Scarano, "Versatile Access to Parallel Memory Systems," *Proc. Workshop Distributed Data and Structures*, 1998.
- [27] J.S. Vitter and E.A.M. Shriver, "Algorithms for Parallel Memory, I: Two-Level Memories," *Algorithmica*, vol. 12, pp. 110-147, 1994.
- [28] J.S. Vitter and E.A.M. Shriver, "Algorithms for Parallel Memory II: Hierarchical Multilevel Memories," *Algorithmica*, vol. 12, pp. 148-169, 1994.
- [29] H.A.G. Wijshoff, "Storing Trees Into Parallel Memories," *Parallel Computing*, pp. 253-261, 1986.



interests focus on distributed algorithms, parallel data structures, peer-to-peer systems, and Internet-based computing.



Computer Science at the University of Massachusetts at Amherst, where he did research with Professor Arnold Rosenberg. His research is currently focused on distributed multimedia systems on the World Wide Web, covering several aspects from a theoretical perspective (P2P systems and architectures) to applications (intermediaries, cooperative systems, and multimedia).



positions at Yale University and the University of Toronto; he was a Lady Davis Visiting Professor at the Technion (Israel Institute of Technology) in 1994 and a Fulbright Research Scholar at the University of Paris-South in 2000. Dr. Rosenberg's research focuses on developing algorithmic models and techniques to deal with the new modalities of "collaborative computing" (the endeavor of having several computers cooperate in the solution of a single computational problem) that result from emerging technologies. He is the author or coauthor of more than 150 technical papers on these and other topics in theoretical computer science and discrete mathematics and is the coauthor of the book *Graph Separators, with Applications*. He is a fellow of the ACM, a fellow of the IEEE, and a Golden Core member of the IEEE Computer Society.

► For more information on this or any other computing topic, please visit our Digital Library at [www.computer.org/publications/dlib](http://www.computer.org/publications/dlib).