Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Computer and System Sciences 70 (2005) 53–72
JOURNAL OF COMPUTER AND SYSTEM SCIENCES
www.elsevier.com/locate/jcss

Wire length as a circuit complexity measure☆

Robert A. Legenstein*, Wolfgang Maass

Institute for Theoretical Computer Science, Technische Universität Graz, A-8010 Graz, Austria

Received 11 October 2001; received in revised form 7 June 2004

Available online 17 August 2004

Abstract

We introduce *wire length* as a salient complexity measure for analyzing the circuit complexity of sensory processing in biological neural systems. This new complexity measure is applied in this paper to two basic computational problems that arise in translation- and scale-invariant pattern recognition, and hence appear to be useful as benchmark problems for sensory processing. We present new circuit design strategies for these benchmark problems that can be implemented within realistic complexity bounds, in particular with linear or almost linear wire length. Finally, we derive some general bounds which provide information about the relationship between new complexity measure wire length and traditional circuit complexity measures.

Keywords: Circuit complexity; Wire length; Sensory processing; VLSI-area

1. Introduction

Carver Mead had suggested that "economizing on wire is the single most important priority for both nerves and chips" [14].

We show in this article that a rather simple and analytically tractable complexity measure can be defined that approximates the wire length of circuits, at least for 3-dimensional circuits as found in many neural

☆ Research for this article was partially supported by the Fonds zur Förderung der wissenschaftlichen Forschung (FWF), Austria, Project P15386, and PASCAL Project # IST2002-506778 of the European Union.
* Corresponding author.
E-mail addresses: legi@igi.tu-graz.ac.at (R.A. Legenstein), maass@igi.tugraz.at (W. Maass).
URLs: http://www.igi.tugraz.at/legi, http://www.igi.tugraz.at/maass.

0022-0000/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcss.2004.06.001

systems (e.g. the cortex) where crossings of dendrites and axons (i.e., "wires") that connect the neurons in a circuit present less of a problem than in 2-dimensional circuits where all wires need to be routed within a few numbers of parallel layers without crossing or touching other wires in the same layer (except if the circuit design calls for such connections). We demonstrate that the principle of minimizing this complexity measure gives rise to interesting circuit designs for basic computational problems that arise in typical sensory processing tasks. These circuits differ significantly from circuits that arise when traditional circuit complexity measures such as the number of gates or the depth of the circuit are minimized. The principle of minimizing wire length has already previously been used in a non-computational setting as an interesting heuristics for understanding details of cortical circuitry if the axons and dendrites of neurons are interpreted as "wires" [5–7,16]. The concepts and constructions presented in this article suggest new ideas for understanding details of cortical circuitry also in a computational setting. In addition it turns out that minimizing the wire length of circuits also yields circuit designs that consume little area in VLSI, based on standard models for VLSI-area. The required wire length for some 1-dimensional pattern matching problems is investigated for threshold circuits in [12].

After introducing our computational model in Section 2, we begin in Section 3 the investigation of circuits for basic computational tasks that can be implemented within biologically realistic bounds with regard to their number of gates and their wire length. We show in Section 4 that two basic pattern recognition tasks can be solved under these severe complexity constraints, one of them even with a number of gates and a wire length that are both linear in the number n of inputs. In Section 5 we derive general bounds for the wire length of a circuit in terms of the number of gates and in terms of the VLSI area required by the circuit.

2. The computational model

The most frequently considered complexity measures in traditional circuit complexity theory are the number (and types) of gates, as well as the depth of a circuit. We will follow traditional circuit complexity theory in assuming that the underlying graph of each circuit is a directed graph without cycles. The depth of a circuit is defined as the length of the longest directed path in the underlying graph, and can also be interpreted as the computation time of the circuit. Most research has focused on the classification of functions that can be computed by circuits whose number of gates is bounded by a polynomial in the number n of input variables. This implicitly also provides a polynomial—although typically quite large—bound on the number of "wires" (defined as the edges in the underlying graph of the circuit), but no bound on the total length of these wires.

We propose the following model for estimating the wire length of an abstract circuit design (which is formally defined as a directed graph with nodes labeled by specific types of gates, or by input- or output-variables):

Definition 1. Gates, input- and output-ports of a circuit C are placed on different nodes of a 2-dimensional grid (with unit distance 1 between adjacent grid nodes). Connections between them are represented by (unidirectional) wires that run through the grid-plane in any way that the designer wants; in particular wires may cross with or without contact and need not run rectilinearly (wires are thought of as running in the 3-dimensional space above the plane, without charge for vertical wire segments). Wires from a gate or input port may branch and provide input to several other gates. We define the minimal value of the

sum of all wire lengths that can be achieved by any such arrangement as the *wire length* $WL(C)$ of the circuit C.

Special computational units that compute functions of k inputs, for some $k > 2$, may be modeled as subcircuits in the following way. Such subcircuits take one unit of time for their computation like all the other gates, but occupy each a set of k intersection points of the grid that are all connected by an undirected wire (whose length contributes to the wire length) in some arbitrary fashion. Any one of these k nodes may be used to provide one of the k inputs or to extract one of the outputs of the function.

Note that the arrangement of the input variables on the grid will in general leave many nodes empty, which can be occupied by gates of the circuit. The last paragraph of Definition 1 is introduced to make this model also applicable to cases where some special functions of k inputs such as the function computed by a threshold gate [1] are computed by neural microcircuits or in analog VLSI by efficient subcircuits that employ a number of transistors, wire length and area that are all linear in k, with a setting time that is independent of k (see [11]).

The model allows that a wire from a gate or input port may branch and provide input to several other gates. For reasonable bounds on the maximal fan-out (10^4 in the case of neural circuits) this is realistic both for neural circuits and for VLSI.

The attractiveness of this model lies in its mathematical simplicity. Nevertheless it provides a useful criterion for judging whether some abstract circuit can potentially be implemented in hardware or "wetware" (i.e., biological neural circuits). We refer to Appendix 6 for an analysis of our model from the biological point of view. This analysis suggests that only those circuit architectures can possibly be implemented in neural circuits that can be implemented according to Definition 1 with a number of gates that is almost linear in the number n of inputs, and a wire length that is quadratic or subquadratic in n—with the additional requirement that the constant factor in front of the asymptotic complexity bound needs to have a value not much larger than 1. Since most practically arising asymptotic bounds involve larger constant factors, one should focus on circuit architectures that can be implemented in our model with clearly subquadratic bounds for their wire length.

Our model for estimating the wire length is easy to handle since one does not have to worry about how exactly the wires need to be routed in order to avoid interference. This laxness may be justified for modeling cortical circuits—since their 2 mm vertical dimension leaves a lot of room to route axons whose thickness lies in the μm range. But it is not a priori justified for estimating the wire length required by a VLSI-implementation of the same circuit, since currently available VLSI-technologies allow just a small number (typically less than 10) of horizontal layers in which wires can be routed. However it turns out that those circuit designs that emerge from minimizing wire length for the computational problems considered in this article are also very area-efficient in a common model for VLSI-area (see the next section).

2.1. The VLSI model

We refer to Section 12.2 in [17] for the precise definition of the abstract model for VLSI-area to which the theorems in this article refer. It is assumed there that gates, input- and output-ports and wires cover rectilinear areas with a width and separation of at least λ. Areas occupied by different gates, input- and

[1] A threshold gate computes a Boolean function $T : \{0,1\}^k \to \{0,1\}$ of the form $T(x_1, \ldots, x_k) = 1 \Leftrightarrow \sum_{i=1}^{k} w_i x_i \geqslant w_0$.

output-ports are not allowed to intersect with one another. Areas occupied by wires may intersect with areas occupied by gates, input- and output-ports and also with other wires, but there is a constant bound μ on the number of wire areas to which a point of the plane may belong. The complexity measure induced by this model is the *area* of the smallest rectangle that encloses the circuit.

Since, we also consider in this article circuits that involve gates with a large number of inputs such as threshold gates, we extend the model for VLSI-area by assuming that a threshold gate with k inputs can be implemented by $k+1$ gates (k of them for multiplying a binary input with a weight, one for comparing the weighted sum with the threshold) that are linearly connected by a wire. We follow [17] in assuming that in the VLSI-model one unit of time is needed to transmit a bit along a wire (of any length), and also for each gate switching. However in contrast to [17] we always assume that all inputs are presented in parallel.

3. Useful design tools

A basic structure for area efficient computing is the well-known H-tree (see, e.g. [15,17]). An H-tree makes optimal use of area and wire length if the n inputs are allowed to be arranged as an $\sqrt{n} \times \sqrt{n}$ array on the plane. Fig. 1a shows the H-tree H_1 with 4 darkly shaded leaves (inputs) and lightly shaded inner nodes of the binary tree. H_{k+1} can be constructed by replacing the leaves of H_k with H-trees H_1. Since H_1 is a tree with four leaves, H_k has 4^k leaves. In Fig. 1b, each leaf of H_1 was replaced by an H-tree H_1. The depth of a node v in an H-tree is the length of the shortest path from v to a leaf. Note that a recursive step in the construction of an H-tree adds depth 2 to the graph. Hence, it will be more convenient to talk about *levels* rather than depth, where a node v is on level i if v is in depth $2i-1$ or in depth $2i$. So, the nodes in depth 1 and 2 are on level 1 (these are the nodes of the last recursive step in the construction of the H-tree), and the root of an H-tree H_k is on level k.

The idea of the H-tree was extended by Leiserson in the context of parallel computing (see [13]). His fat-trees are tree-like structures where each edge of the underlying tree consists of several communication wires. We use a similar structure which we will call an *extended H-tree*. Our layout will differ from the H-tree layout in a crucial point. Internal nodes of the H-tree are replaced by groups of several gates, and the connections between these groups consist of "busses" rather than of single wires. More precisely,

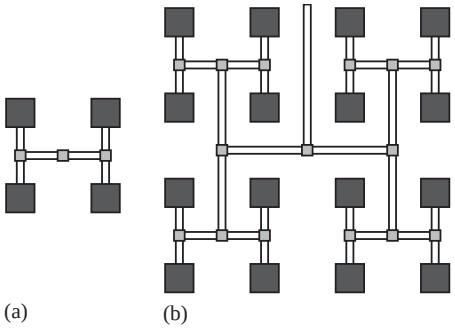

Fig. 1. The H-tree layout. Dark rectangles are leaves, and light rectangles are inner nodes: (a) H_1 is a tree layout for 4 leaves; (b) H_2. H_{k+1} is constructed recursively by replacing the leaves of H_k with H-trees H_1 (Figure taken from [17]).

each "node" on level i of an H-tree is a circuit with $O(i)$ gates and $O(i^2)$ wire length and area with side length $O(i)$. Instead of a single edge in an H-tree one has a "bus" consisting of $O(i)$ wires if the bus connects a node on level i with a node on level i or $i+1$.

One has to be careful in talking about levels and nodes in an extended H-tree, since the circuit in a "node" might consist of several gates and might even have non-constant depth. However each extended H-tree has an underlying H-tree and the levels are counted with regard to this underlying H-tree.

Lemma 1. *Let T be an extended H-tree on n leaves with $O(i)$ wires in a bus on level i, $O(i)$ gates at a node on level i and $O(i^2)$ wire length at a node on level i. Then T consists of $O(n)$ gates, and T can be implemented with $O(n)$ wire length. If the side length of a node on level i is $O(i)$ in the VLSI-model, the layout uses $O(n)$ area in the VLSI-model.*

Proof. We will not only derive asymptotic bounds, but also pay attention to the size of constant factors. To achieve this, we will use the recursive construction of the extended H-tree to derive recursive formulas on size, side-length and wire length of the layout. The nodes on level 1 play a special role in the circuit. There are $\frac{n}{4}$ extended H-trees H_1 on level 1 that compute, in parallel, the basic values for the subsequent "conquer steps". Let $S(H_1)$, $C(H_1)$ and $WL(H_1)$ denote the side-length, size and wire length of one such H_1-circuit.

We start the proof by deriving an upper bound on the side-length $S(H_k)$ of the extended H-tree H_k. Since the number of gates in a node on level i is $O(i)$, we can assume that the side-length of a node on level i is bounded by ci for some suitable constant c. The side-length of H_k is the sum of the side-lengths of two H-trees H_{k-1} and the side length of a node on level k (see Fig. 2). Hence, the following recurrence holds:

$$S(H_k) \leqslant 2S(H_{k-1}) + ck. \tag{1}$$

The solution of Eq. (1) yields the bound $S(H_k) \leqslant 2^{k-1}S(H_1) + \frac{3c}{2}2^k$. Since $n = 4^k$, we have $S(H_k) \leqslant \sqrt{n}(\frac{S(H_1)}{2} + \frac{3c}{2}) = O(\sqrt{n})$. The area of the layout in the VLSI-model if the sidelength of nodes on level i is bounded by ci is therefore

$$area(H_k) = S^2(H_k) \leqslant \left(\frac{S(H_1)}{2} + \frac{3c}{2}\right)^2 n = O(n).$$

A similar recurrence holds for the number of gates $C(H_k)$ in the circuit for the H-tree H_k: Let the number of gates at a node on level i of the extended H-tree be bounded by si for a suitable constant s (recall that a recursive step in the H-tree layout adds 3 inner nodes). We get a recursive formula which we iterate $k-1$ times:

$$\begin{aligned} C(H_k) &\leqslant 4C(H_{k-1}) + 3 \cdot s \cdot k \\ &\leqslant 4^{k-1}C(H_1) + 3 \cdot s \sum_{j=0}^{k-2} 4^j (k-j). \end{aligned} \tag{2}$$

Since $\sum_{j=0}^{k-2} 4^j(k-j) \leqslant \frac{77}{36}4^k$ the solution of Eq. (2) is

$$C(H_k) \leqslant n\left(\frac{1}{4}C(H_1) + \frac{77}{12}s\right). \tag{3}$$

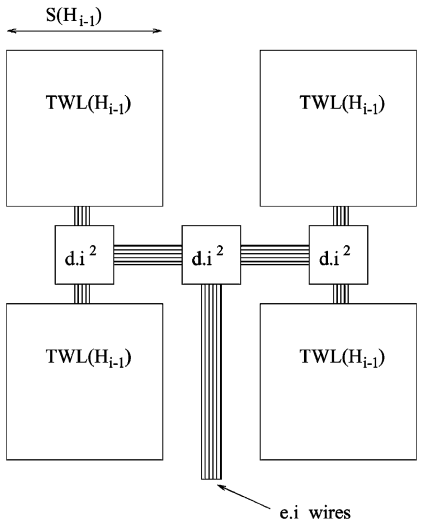

Fig. 2. The H-tree H_i has wires from four H-trees H_{i-1}, the wires of three inner nodes, and the wires of the busses. An inner node has a wire length of $d \cdot i^2$, and a bus consists of $e \cdot i$ wires.

Now we use a similar argument to estimate the wire length. The wire length of the layout consists of the wire lengths at the inner nodes and the wire lengths of the "busses". Let d be a constant such that the wire length of a node on level i is bounded by $d \cdot i^2$. Also, let the number of wires of a "bus" from a node on level i to a node on level i or $i+1$ be bounded by $e \cdot i$. The basis for the recursive calculation of the wire length for an extended H-tree H_i is illustrated in Fig. 2. We get a recursive formula which we iterate $k-1$ times:

$$\begin{aligned} WL(H_k) &\leqslant 4WL(H_{k-1}) + 2k \cdot e \cdot S(H_{k-1}) + 3d \cdot k^2 \qquad (4)\\ &\leqslant 4WL(H_{k-1}) + k \cdot e(S(H_1) + 3c)2^{k-1} + 3d \cdot k^2, \end{aligned}$$

$$\begin{aligned} WL(H_k) &\leqslant 4^{k-1}WL(H_1) + 2^{k-1}e(S(H_1) + 3c) \sum_{j=0}^{k-2} 2^j(k-j)\\ &\quad +3d \sum_{j=0}^{k-2} 4^j(k-j)^2. \qquad (5) \end{aligned}$$

Since $\sum_{j=0}^{k-2} 2^j(k-j) \leqslant \frac{3}{2}2^k$ and $\sum_{j=0}^{k-2} 4^j(k-j)^2 \leqslant \frac{77}{108}4^k$ we get:

$$\begin{aligned} WL(H_k) &\leqslant 4^{k-1}WL(H_1) + \frac{3}{2}2^{2k-1}e(S(H_1) + 3c) + \frac{77}{36}d4^k\\ &\leqslant \frac{1}{4}n \cdot WL(H_1) + \frac{3}{4}n \cdot e(S(H_1) + 3c) + \frac{77}{36}d \cdot n\\ WL(H_k) &\leqslant n\left(\frac{1}{4}WL(H_1) + \frac{3}{4}e(S(H_1) + 3c) + \frac{77}{36}d\right) = O(n). \qquad \square \qquad (6) \end{aligned}$$

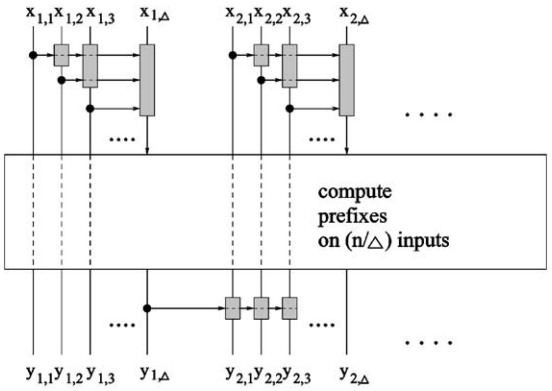

Fig. 3. Layout of an efficient prefix circuit with fan-in Δ. Dark shaded boxes are gates, and the light shaded box is the recursive application of the circuit.

Another widely used strategy in parallel computation is the computation of prefixes in parallel prefix circuits. We will use parallel prefix circuits in Section 4. Consider a set X of elements with an associative binary operation. We denote the binary operation by juxtaposition of the elements in X. Suppose we have functional gates such that each gate with inputs $x_1, \ldots, x_k$ computes the product $x_1 x_2 \ldots x_k$, for some fan-in k.

There exist efficient circuits for such computations (see e.g. [17]). We show how a parallel prefix circuit can be implemented in our model and the VLSI-model. Lemma 2 gives upper bounds on a circuit of such gates with maximal fan-in Δ that computes the prefixes $x_1, x_1x_2, \ldots, x_1x_2 \ldots x_n$. For simplicity, we assume that n is a power of Δ.

Lemma 2. *If n inputs $x_1, \ldots, x_n$ are arranged on a row of a grid, then the prefixes $x_1, x_1x_2, \ldots, x_1x_2 \ldots x_n$ can be computed by a circuit with maximum fan-in $\Delta \in \{2, \ldots, n\}$, $size \leqslant 2n$ in $depth = 2\frac{\log n}{\log \Delta}$. In our model the circuit uses only a constant number of rows and the wire length is $O(\frac{\log n}{\log \Delta} n\Delta)$. In the VLSI-model the circuit uses an area $\leqslant n\Delta \frac{\log n}{\log \Delta}$.*

Proof. We divide the inputs $x_1, \ldots, x_n$ into $\frac{n}{\Delta}$ consecutive subintervals and rename the inputs to $x_{1,1}, \ldots, x_{1,\Delta}, x_{2,1}, \ldots, x_{2,\Delta}, \ldots, x_{\frac{n}{\Delta},1}, \ldots, x_{\frac{n}{\Delta},\Delta}$. We denote the outputs of the circuit as $y_1, \ldots, y_n$ such that $y_i = x_1 \ldots x_i$. It will be convenient to divide the outputs into consecutive subintervals in the same manner as the inputs. Then, the outputs of the circuit can be written as $y_{1,1}, \ldots, y_{1,\Delta}, y_{2,1}, \ldots, y_{2,\Delta}, \ldots, y_{\frac{n}{\Delta},1}, \ldots, y_{\frac{n}{\Delta},\Delta}$ where $y_{i,j} = y_{(i-1)\Delta+j}$. These intervals for inputs and outputs are illustrated in Fig. 3.

In the first step, we compute the prefixes for each group of inputs $x_{i,1}, \ldots, x_{i,\Delta}$, i.e. we compute $x'_{i,j} = x_{i,1}x_{i,2} \ldots x_{i,j}$ for $i = 1, \ldots, \frac{n}{\Delta}$ and $j = 1, \ldots, \Delta$. In the second step, we recursively apply the prefix computation on $x'_{1,\Delta}, x'_{2,\Delta}, \ldots, x'_{\frac{n}{\Delta},\Delta}$, gaining the prefixes $y_{i,\Delta} = x_1x_2 \ldots x_{i\cdot\Delta}$. In the third step, we finally fill up the gaps between those prefixes with $y_{i,j} = y_{(i-1),\Delta}x'_{i,j}$ for $i = 2, \ldots, \frac{n}{\Delta}$ and $j = 1, \ldots, \Delta - 1$. The layout and structure of the circuit is shown in Fig. 3. Fig. 4 shows the whole circuit for $\Delta = 2, n = 8$.

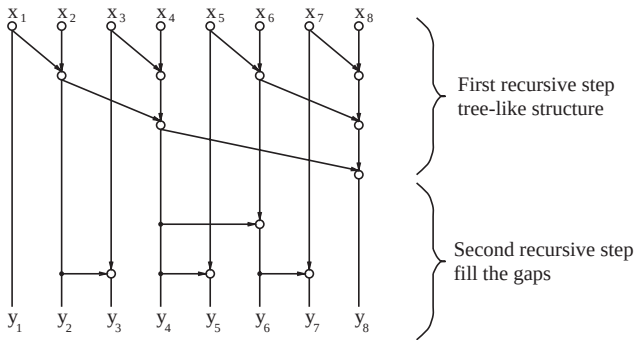

Fig. 4. The prefix circuit for $\Delta = 2$ and $n = 5$. It can be decomposed into two parts. The upper part is a tree. (Based on Fig. 2.13 in [17].)

Since the construction of the circuit and layout is recursive, we can give recursive formulas for size, depth, area and wire length of the circuit. Let $PREF^n$ denote such a layout with n inputs. The first computational step and the third computational step each consist of $\frac{n}{\Delta}(\Delta - 1)$ gates. The recursive step computes the prefixes on $\frac{n}{\Delta}$ inputs:

$$\begin{aligned} C(PREF^n) &\leqslant \frac{n}{\Delta}(\Delta - 1) + C(PREF^{\frac{n}{\Delta}}) + \frac{n}{\Delta}(\Delta - 1) \\ &= 2n - 2\frac{n}{\Delta} + C(PREF^{\frac{n}{\Delta}}). \end{aligned}$$

The solution to this recurrence is $C(PREF^n) \leqslant 2n$, since $C(PREF^1) = 0$. Each recursive step adds depth 2 to the circuit depth:

$$depth(PREF^n) = 2 + depth(PREF^{\frac{n}{\Delta}}).$$

The solution to this recurrence is $depth(PREF^n) = 2\frac{\log n}{\log \Delta}$, since $depth(1) = 0$. To bound the occupied area in the VLSI-model, we compute the vertical side length $S(PREF^n)$ of the layout. Let $area(L)$ denote the area used by a layout L.

$$S(PREF^1) = 0$$

$$S(PREF^n) \leqslant (\Delta - 1) + S(PREF^{\frac{n}{\Delta}}) + 1 = \Delta\frac{\log n}{\log \Delta}, \tag{7}$$

$$area(PREF^n) \leqslant nS(PREF^n) = n\Delta\frac{\log n}{\log \Delta}. \tag{8}$$

Note that this area bound is derived for the VLSI-model. In our model, there is a better layout since we do not need space for wires. Nevertheless, Fig. 3 gives an idea of a recursive formula for the wire length of *horizontal* wires:

$$\begin{aligned} WL(PREF^1) &= 0 \\ WL(PREF^n) &\leqslant \frac{n}{\Delta}\Delta^2 + \Delta WL(PREF^{\frac{n}{\Delta}}) + n \\ &= n\Delta + n + \Delta WL(PREF^{\frac{n}{\Delta}}) = n(\Delta + 1)\frac{\log n}{\log \Delta}. \end{aligned}$$

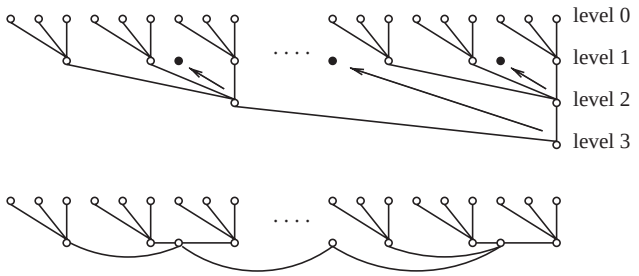

Fig. 5. Tree layout on a grid for $\Delta = 3$. The upper layout shows a leveled arrangement of inner nodes. The arrows and filled circles indicate the rearrangement of inner nodes. We gain a layout which uses just two rows (lower layout).

Since the circuit has logarithmic depth, *vertical* wires have a summed length of $O(n\Delta\frac{\log n}{\log \Delta})$ and the upper bound for wire length is:

$$WL(PREF^n) = O\left(n\Delta\,\frac{\log n}{\log \Delta}\right). \tag{9}$$

The advantage of this circuit in our model is that one can implement it in area $O(n)$ without increasing the wire length. As shown in Fig. 4 for $\Delta = 2$, the circuit implements a Δ-ary tree to compute larger and larger prefixes (first recursive step) and then fills up gaps in the computed prefixes (second recursive step). We show that the tree can be implemented by using two rows.

The area-efficient implementation of a tree is illustrated in Fig. 5. As shown in Fig. 5, we label the nodes of the tree such that leaves are in level 0, inner nodes which are incident to leaves are in level 1 and so on. More formally, a node v is in level i if the shortest path from v to some leaf has i edges. For a node v on level i, denote the subgraph consisting of all nodes on levels $0, \ldots, i$ and all edges between them as the *subtree of* v. We start with a layout L for a Δ-ary tree where inner nodes are placed beneath the rightmost root of their predecessors' subtrees (for an inner node on level $i \geqslant 2$, these are the subtrees of adjacent nodes on level $i-1$). This is shown in the upper graph of Fig. 5. Note that L is the layout of the tree in the prefix circuit (confirm Fig. 4). Furthermore, L has wire length $O(n \log n)$. We show how to rearrange the inner nodes of L to achieve an area-efficient layout L'. For L', we place each inner node beneath the leftmost leaf of its rightmost subtree (indicated by arrows and filled circles in Fig. 5). The layout L' is shown in the lower graph of Fig. 5. It is easy to show that this location is not occupied by another node, and that the wire length of L' is bounded from above by the wire length of the previous layout L. But it uses just two rows on the grid.

In order to use L' in the layout of a prefix circuit, we note that L' differs from L since inner nodes are horizontally displaced as compared to L. Since the outputs of these nodes are needed for further computation in the prefix circuit, we need to check if this displacement results in a significant increase in wire length. However, if the summed displacement over all inner nodes is small, one could simply introduce additional wires that map the outputs of inner nodes back to the horizontal positions given by L. Then, the horizontal displacement can be ignored. This is shown in the following. There are $\frac{n}{\Delta^i}$ nodes in level i. The horizontal displacement of a node in level i is Δ^{i-1} and nodes in level 1 are not displaced. Hence, the summed displacement of nodes is: $\sum_{i=2}^{\log_\Delta n} \Delta^{i-1}\frac{n}{\Delta^i} \leqslant \frac{n}{\Delta}\frac{\log n}{\log \Delta}$.

It remains to be shown that the computations in the second recursion step (see Fig. 4) can be implemented in one row. Just observe that whenever there is a gate in this second recursive step, it computes an output of the circuit. Hence in the second recursive step, each gate can be paced above one output in a single row. This concludes the layout of the prefix circuit. □

4. Global pattern detection in 2-dimensional maps

Several of the most successful computer vision methods for generic object recognition [19,8,2] represent objects by 2-dimensional spatial relationships between local features or "interest points" (e.g. corners, i.e., points where two edges meet at a certain angle, or centric interest points such as eyes in a face); see Fig. 6. Since the visual system of most organisms transforms light from the outside world in a topographic manner into a corresponding 2-dimensional activation pattern of neurons in primary visual cortex, it is not impossible that brains apply a related method for generic object recognition.

We formalize such 2-dimensional global pattern detection problems by assuming that the input consists of 2-dimensional matrices A, B, etc. of binary variables a_{ij}, b_{ij}. Each index pair $\langle i, j\rangle$ of an input variable can be thought of as representing a location in a 2-dimensional image. We assume that $a_{ij} = 1$ if and only if feature a is detected at location $\langle i, j\rangle$ and that $b_{ij} = 1$ if and only if feature b is detected at location $\langle i, j\rangle$.

We can present such input in our model by reserving a sub-square (i.e. adjacent nodes of the grid within a square) within the 2-dimensional grid for each index pair $\langle i, j\rangle$, where the input variables a_{ij}, b_{ij}, etc. are given on adjacent nodes of this grid. To avoid cumbersome notation, we will in the following skip double indexing of input variables and represent the input by arrays $\underline{a} = \langle a_1, \ldots, a_n\rangle, \underline{b} = \langle b_1, \ldots, b_n\rangle$, etc. of binary variables that are arranged on a 2-dimensional square grid. To make this more precise we assume that indices i and j represent pairs $\langle i_1, i_2\rangle, \langle j_1, j_2\rangle$ of coordinates. Then "input location j is above and to the right of input location i" means: $i_1 < j_1$ and $i_2 < j_2$. For functions considered in this article, the circuit complexity is not altered if one or both of the "$<$" is replaced by "$\leqslant$" in the notion above. Since, we assume that this spatial arrangement of input variables reflects spatial relations in the outside world, many salient examples for global pattern detection problems require the computation of functions such as

$$P_D^n(\underline{a}, \underline{b}) = \begin{cases} 1 & \text{if there exist } i \text{ and } j \text{ so that } a_i = b_j = 1 \text{ and input location } j \\ & \text{is above and to the right of input location } i, \\ 0 & \text{else.} \end{cases}$$

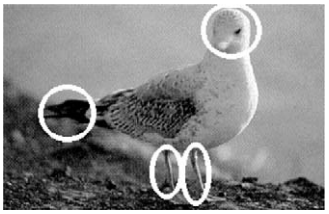

Fig. 6. Examples of some local features (marked), whose spatial arrangement is essential for recognizing an object.

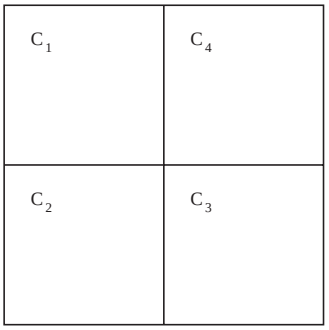

Fig. 7. The input area C is divided into four sub-squares C_k, which are numbered in a counterclockwise fashion.

If $P_D^n(\underline{a}, \underline{b}) = 1$ then we refer to indices i and j for which the first clause in this definition is satisfied as "witnesses" (for the fact that $P_D^n(\underline{a}, \underline{b}) = 1$).

Theorem 1. *The function P_D^n can be computed and witnesses i and j with $a_i = b_j = 1$ can be exhibited if they exist by a circuit with wire length $O(n)$, consisting of $O(n)$ Boolean gates of fan-in 2 and fan-out 2 in depth $O(\log n \cdot \log\log n)$.*

The depth of the circuit can be reduced to $O(\log n)$ if one employs threshold gates with fan-in $\log n$. This can also be done with wire length $O(n)$. In the VLSI-model, this circuit uses $O(n)$ area.

Proof. This circuit design is based on a divide-and-conquer approach. On first sight it appears that such an approach is bound to fail for computing P_D^n, since there may exist for example just a single pair of witnesses i and j with the desired properties, but the chosen subdivision of the input area happens to assign i and j to *different* components of the subdivision. Hence the evaluation of P_D for each of the components is of little help for the evaluation of P_D^n for the full input area.

In order to make the divide-and-conquer approach feasible it is essential that one computes for each component of the subdivision more than just whether P_D holds for this component. If one divides iteratively each square into 4 sub-squares C_1, C_2, C_3, C_4, (see Fig. 7) then it suffices to compute for each sub-square C_k the following data:

$$\begin{aligned}
\mathit{left}(C_k) &:= \text{the x-coordinate of the leftmost location } i \text{ in } C_k \text{ with } a_i = 1\\
\mathit{right}(C_k) &:= \text{the x-coordinate of the rightmost location } j \text{ in } C_k \text{ with } b_j = 1\\
\mathit{down}(C_k) &:= \text{the y-coordinate of the lowest location } i \text{ in } C_k \text{ with } a_i = 1\\
\mathit{up}(C_k) &:= \text{the y-coordinate of the highest location } j \text{ in } C_k \text{ with } b_j = 1\\
\mathit{found}(C_k) &:= \begin{cases} 1 & \text{if } P_D \text{ applied to } C_k \text{ outputs 1,}\\ 0 & \text{else.}\end{cases}
\end{aligned}$$

We assume that each of the first four functions assumes the value 0 on C_k if and only if there exists no location i or j in C_k with the desired property. Thus all coordinates are assumed to be numbers $\geqslant 1$.

The essential property of these 5 functions is that $\mathit{left}(C)$, $\mathit{right}(C)$, $\mathit{down}(C)$, $\mathit{up}(C)$ and $\mathit{found}(C)$ can be computed from the values of these 5 functions for the 4 sub-squares C_1, C_2, C_3, C_4. This is obvious for $\mathit{left}(C)$, $\mathit{right}(C)$, $\mathit{down}(C)$, $\mathit{up}(C)$, requiring just comparisons of pairs of $(b+1)$-bit natural numbers

if each C_k is responsible for a sub-square of the input-array of size $2^b \times 2^b$. The value of $found(C)$ can be computed in the following fashion, assuming that the components C_k that make up C are numbered in a counterclockwise fashion, starting with C_1 in the upper left quadrangle (see Fig. 7):

$$
\begin{aligned}
&found(C) = 1 \\
&\Leftrightarrow \bigvee_{k=1}^{4} found(C_k) = 1 \vee \\
&\quad 0 < down(C_1) < up(C_4) \vee \\
&\quad 0 < down(C_2) < up(C_3) \vee \\
&\quad ((0 < down(C_2)) \wedge (0 < up(C_4))) \vee \\
&\quad 0 < left(C_2) < right(C_1) \vee \\
&\quad 0 < left(C_3) < right(C_4).
\end{aligned}
$$

Obviously this algorithm makes use of the fact that the area is not subdivided in an *arbitrary* fashion into components, but in a way which is consistent with the map, i.e. with the spatial relationship of locations in the outside world. Or, with a variation of a well-known design philosophy of Carver Mead, one could say that *space represents itself* in this algorithm design.

The layout of a circuit for P_D^n with small wire length is based on the extended H-tree discussed in Section 3. We now show how the extended H-tree can be used as a layout strategy for a circuit that implements the previously developed algorithm for solving P_D^n. The extended H-tree layout implements the structure of the algorithm by recursively dividing the input-area into four axis-parallel sub-squares. The computations needed in a node on level i of the H-tree can be carried out by a circuit of size $O(i)$ and $O(i^2)$ wire length and area, which is placed at that node. The depth of a circuit at a node is $O(1)$ if threshold gates of fan-in $O(\log n)$ are used and $O(\log i)$ if Boolean gates of fan-in 2 are used. Lemma 1 shows that the extended H-tree stays within the claimed complexity bounds. The depth of an H-tree is $O(\log n)$, hence if the circuits at the nodes have depth $O(1)$, the extended H-tree has depth $O(\log n)$. If the circuits at the nodes have depth $O(\log i)$, the depth of the extended H-tree is $O(\log n \cdot \log\log n)$.

An extension to the circuit that reports a pair of witnesses is straight forward. □

The linear wire length of this circuit is *optimal* up to a constant factor for any circuit whose output depends on all of its n inputs. Note that most connections in this circuit are local, just like in a biological neural circuit. Thus, we see that minimizing wire length tends to generate biology-like circuit structures.

However, the tree-like circuit structure results in considerable circuit-depth for large input-size. In biological neural systems, neural gates of large fan-in are used to implement shallow circuits, whereas the circuit design above is based on gates of fan-in 2 or $\log(n)$ which is comparatively small.

The next theorem shows that one can compute P_D^n faster (i.e. by a circuit with smaller depth) if one can afford a somewhat larger wire length. This circuit construction, which is based on AND/OR gates of limited fan-in Δ, has the additional advantage that it can exhibit *all* j that can be used as witness together with some i. This property allows us to "chain" the global pattern detection problem formalized through the function P_D^n, and to decide within the same complexity bound whether for any fixed number k of input vectors $\underline{a}^{(1)}, \ldots, \underline{a}^{(k)}$ from $\{0,1\}^n$ there exist locations $i^{(1)}, \ldots, i^{(k)}$ so that $a_{i^{(m)}}^{(m)} = 1$ for $m = 1, \ldots, k$ and location $i^{(m+1)}$ lies to the right and above location $i^{(m)}$ for $m = 1, \ldots, k-1$. In fact, one can also

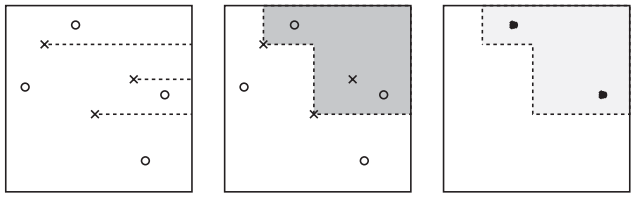

Fig. 8. Computing P_D with prefix circuits. Crosses mark locations where a feature a occurs, open circles mark locations where a features b is present: (a) all locations that are in the same row and to the right of some feature $a = 1$ are marked as dotted lines; (b) all locations that are to the right of and above some feature $a = 1$ are shaded; (c) all locations i with $b_i = 1$ in this area are marked with filled circles.

compute a k-tuple of witnesses $i^{(1)}, \ldots, i^{(k)}$ within the same complexity bounds, provided it exists. This circuit design is based on an efficient layout for prefix computations.

Theorem 2. *For any given n and $\Delta \in \{2, \ldots, \sqrt{n}\}$ one can compute the function P_D^n in depth $O(\frac{\log n}{\log \Delta})$ by a circuit consisting of $O(n)$ AND/OR gates of fan-in $\leqslant \Delta$, with wire length $O(n \cdot \Delta \cdot \frac{\log n}{\log \Delta})$.*

In the VLSI-model, the circuit uses $O(n \cdot (\Delta \cdot \frac{\log n}{\log \Delta})^2)$ area.

Proof. The main idea in the construction of the circuit is illustrated in Fig. 8. In Fig. 8a, a two dimensional input-assignment for P_D^n is shown. Crosses mark locations where a feature a is present and open circles mark locations where a feature b occurs. Every feature b that is located in the shaded region in Fig. 8b is located to the right of and above some present feature a. Hence, if there is some location j that is in the shaded region of Fig. 8b and $b_j = 1$, then the value of $P_D^n(\underline{a}, \underline{b})$ is 1. We introduce indicator variables a'_j $(j = 1, \ldots, n)$, where $a'_j = 1$ if the location j is to the right and above to some location i with $a_i = 1$, and $a'_j = 0$ otherwise. (In Fig. 8b, $a'_j = 1$, if j is a location in the shaded region). It follows that P_D^n has value 1 if there exists some location j such that $a'_j \wedge b_j = 1$.

Hence, the problem is reduced to the problem of computing the values of a'_j for all locations $j = 1, \ldots, n$. A straight-forward implementation would lead either to large depth or to large wire length. In a 1-dimensional scenario, the problem would be equivalent to the following one. Suppose one has a one dimensional array of pixels $x_1, \ldots, x_n$. Then the equivalent problem to computing a'_j would be to compute the values of $x'_1, \ldots, x'_n$ where $x'_j = 1$ if and only if there is a $x_i = 1$ that is to the left of x_j. This is the problem of computing the *prefixes*: $x'_1 = x_1, x'_2 = x_1 \vee x_2, x'_3 = x_1 \vee x_2 \vee x_3, \ldots, x'_n = x_1 \vee x_2 \vee x_3 \vee \ldots \vee x_n$. Such a computation is called a *prefix computation*, see also Section 3. In the 2-dimensional case, we just need to apply prefix computations on all rows and subsequent prefix computations on the columns of the outcomes. A similar serial algorithm for multidimensional prefix computation is given in [9]. By applying the prefix computation on rows of $\underline{a}$, one can determine the locations in the input plane that are in the same row as some feature $a_i = 1$ and located to the right of a_i. This is illustrated in Fig. 8a. Here, the horizontal lines in the input space represent locations where indicator variables have value 1 after that step.

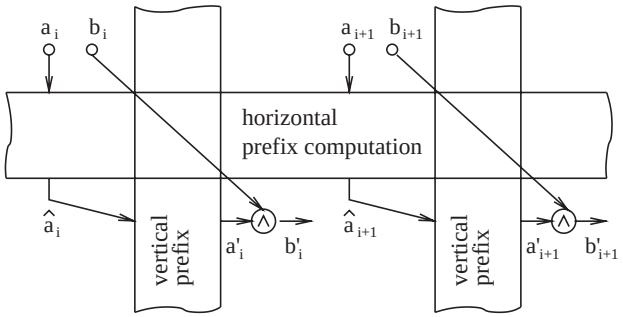

Fig. 9. Detail of the circuit layout for computing P_D^n using prefix circuits. The horizontal prefix circuit computes prefixes $\hat{a}$ on features a in the same row with feature a_i. Vertical prefix circuits are applied to the results of the horizontal prefix circuits to compute a_i'. Another AND gate computes $b_i' = b_i \wedge a_i'$.

Let us call the outputs of the horizontal prefix circuits $\hat{a}_j$, where $j = 1, \ldots, n$ denotes locations in the same manner as the inputs are indexed. Then, a location j is in the right spatial relation to some feature $a_i = 1$ at location i, if it is above of some location k with $\hat{a}_k = 1$. Hence, we can apply the same prefix-operation on columns of these intermediate variables $\hat{a}_1, \ldots, \hat{a}_n$ to compute the correct value of all indicator variables (see Fig. 8b). Now, $b_i' = a_i' \wedge b_i$ has value 1 if location i is in the right spatial relation with some present feature a and $b_i = 1$. (This is not exactly what we want, since this would also mark b-features that lie in the same row or column with some a-feature. However, we can also AND the b-feature with the marking-bit that is one pixel to the left and below it.) In Fig. 8c, the locations l with $b_l' = 1$ are marked with filled circles. Finally, an OR over all b_i''s outputs $P_D^n(\underline{a}, \underline{b})$ for all inputs $\underline{a}, \underline{b} \in \{0,1\}^n$.

The circuit consists of prefix computations for every row of feature a ($\sqrt{n}$ many rows) that compute $\hat{a}_1, \ldots, \hat{a}_n$. Then, prefix circuits are applied on columns of the outputs of these circuits ($\sqrt{n}$ many columns) to compute $a_1', \ldots, a_n'$. Furthermore, n AND gates are used for the computation of $b_i' = a_i' \wedge b_i$. A detail of the circuit layout is shown in Fig. 9. Finally, there is one OR with inputs $b_1', \ldots, b_n'$. This OR can be implemented by a tree of OR gates with fan-in Δ in order to reduce the wire length. This adds depth $O(\frac{\log n}{\log \Delta})$ to the circuit.

Let $C(PREF^n)$, $depth(PREF^n)$ and $WL(PREF^n)$ denote the size, depth and wire length of a prefix circuit with n inputs. Hence, the circuit has size $O(\sqrt{n}\, C(PREF^{\sqrt{n}}) + n)$ and depth $O(depth(PREF^{\sqrt{n}}) + \frac{\log n}{\log \Delta})$. By Lemma 2, this results in size $O(n)$ and depth $O(\frac{\log n}{\log \Delta})$.

In the following, we give upper bounds on wire length and area for this circuit. Lemma 2 gives upper bounds on wire length and area for an efficient prefix circuit consisting of gates with maximal fan-in Δ ($\Delta \in \{2, \ldots, \sqrt{n}\}$). There is a prefix computation of $\sqrt{n}$ inputs for each row of $\underline{a}$ in the input plane. We can place this prefix circuit in between the rows of inputs. Note that if these circuits would need too many rows, we would have to place the input rows far away from each other. This would influence the wire length of the subsequent prefix circuits. But since by Lemma 2 each prefix circuit uses a constant number of rows in our model, the computations for rows and columns do not affect each other. Therefore the wire length used for this part of the computation is $O(\sqrt{n}\sqrt{n}\Delta\frac{\log n}{\log \Delta}) = O(n\Delta\frac{\log n}{\log \Delta})$. The AND gates that compute $b_i' = a_i' \wedge b_i$ need $O(n)$ wire length all together. We implement the OR of $b_1', \ldots, b_n'$ as a 2-dimensional tree of fan-in Δ. This influences the size and the depth of the circuit only by a constant

factor. It can be shown that a 2-dimensional tree of fan-in Δ has wire length $O(n\sqrt{\Delta})$. Hence, the circuit has $WL = O\left(n \cdot \Delta \cdot \frac{\log n}{\log \Delta}\right)$, $depth = O\left(\frac{\log n}{\log \Delta}\right)$, and $size = O(n)$.

The situation is different in the VLSI-model. The crucial part of the layout is the prefix circuits. In the VLSI-model, these circuits have side-lengths $O(\Delta \frac{\log n}{\log \Delta})$ and $O(\sqrt{n})$ each (see proof of Lemma 2). Nevertheless, we layout these circuits in the same manner as above. Since we need one prefix circuit for every row and every column, the side length of the layout for the prefix circuits is $O(\sqrt{n}\Delta \frac{\log n}{\log \Delta})$. Hence, the circuit for P_D^n can be implemented within an area of $O\left(n \cdot \left(\Delta \cdot \frac{\log n}{\log \Delta}\right)^2\right)$. □

An advantage of this approach is that we computed *all* the witnesses in $\underline{b}$ for P_D. Hence we can use this information to compare these witnesses with some feature $\underline{c}$. In other words, we can compute if there is some feature a beneath and to the left of some feature b which is beneath and to the left of some feature c and so on. Denote this function with $P_D^{n,k}$ for some $k \geqslant 2$. Consider $k \geqslant 2$ different feature types $\underline{a}^{(1)}, \ldots, \underline{a}^{(k)}$. $P_D^{n,k}$ is defined as

$$P_D^{n,k}(\underline{a}^{(1)}, \ldots, \underline{a}^{(k)}) = \begin{cases} 1 & \text{if there exist } i^{(1)}, \ldots, i^{(k)} \text{ so that} \\ & a_{i^{(l)}}^{(l)} = 1 \text{ for all } l \in \{1, k\} \text{ and input} \\ & \text{location } i^{(l+1)} \text{ is to the right and} \\ & \text{above of input location } i^{(l)} \\ & \text{for all } l \in \{1, k-1\}, \\ 0 & \text{else.} \end{cases}$$

Consider a multidimensional function $W^{n,k} : \{0,1\}^{k \cdot n} \to \{0,1\}^n$ that reports all locations $i^{(k)}$ (i.e. positions in the last feature map $\underline{a}^{(k)}$) that, together with some locations $i^{(1)}, \ldots, i^{(k)}$, satisfy the upper clause in the definition above. For $k = 2$, this function can be written as

$$W^{n,2}(\underline{a}, \underline{b}) = (w_1, \ldots, w_n), \quad \text{where}$$

$$w_j = \begin{cases} 1 & \text{if } b_j = 1 \text{ and there exist } i \text{ so that} \\ & a_i = 1 \text{ and input location } j \\ & \text{is above and to the right of input} \\ & \text{location } i, \\ 0 & \text{else.} \end{cases}$$

For $k > 2$, one can define this function recursively as

$$W^{n,k}(\underline{a}^{(1)}, \ldots, \underline{a}^{(k)}) = W^{n,2}(W^{n,k-1}(\underline{a}^{(1)}, \ldots, \underline{a}^{(k-1)}), \underline{a}^{(k)}).$$

Recall that we computed $w_1, \ldots, w_n$ in the circuit for P_D^n and called these values $b'_1, \ldots, b'_n$ in the proof of Theorem 2. Hence, by the recursive definition of $W^{n,k}$, one just has to apply this circuit $k-1$ times to compute $W^{n,k}$. $P_D^{n,k}$ is 1 if and only if a witness exists, i.e. at least one component in $W^{n,k}(\underline{a}^{(1)}, \ldots, \underline{a}^{(k)})$ is 1. Therefore, Corollary 1 holds:

Corollary 1. *For any given n, $k \geqslant 2$ and $\Delta \in \{2, \ldots, \sqrt{n}\}$ one can compute the function $P_D^{n,k}$ in depth $O(k \frac{\log n}{\log \Delta})$ by a circuit consisting of $O(k \cdot n)$ AND/OR gates of fan-in $\leqslant \Delta$, with wire length $O(k \cdot n \cdot \Delta \cdot \frac{\log n}{\log \Delta})$ and area $O(n \cdot (k \cdot \Delta \cdot \frac{\log n}{\log \Delta})^2)$.*

Another essential ingredient of translation- and scale-invariant global pattern recognition is the capability to detect whether a local feature c occurs in between locations i and j where the local features a and b occur. This global pattern detection problem is formalized through the following function $P_I^n : \{0,1\}^{3n} \to \{0,1\}$:

If $\sum \underline{a} = \sum \underline{b} = 1$ *then* $P_I^n(\underline{a}, \underline{b}, \underline{c}) = 1$, *if and only if there exist* i, j, k *so that input location* k *lies on the middle of the line between locations* i *and* j, *and* $a_i = b_j = c_k = 1$.

This function P_I^n can be computed very fast by circuits with the least possible wire length (up to a constant factor), using threshold gates of fan-in up to $\sqrt{n}$:

Theorem 3. *The function* P_I^n *can be computed—and witnesses can be exhibited—by a circuit with wire length and area* $O(n)$, *consisting of* $O(n)$ *Boolean gates of fan-in* 2 *and* $O(\sqrt{n})$ *threshold gates of fan-in* $\sqrt{n}$ *in depth* 7.

Proof. Omitted. □

5. Relationship between wire length and other circuit complexity measures

The most common complexity measure in traditional circuit complexity theory is the circuit size $C(f)$ of a Boolean function $f : \{0,1\}^n \to \{0,1\}$. $C(f)$ is the smallest number of gates in any feedforward circuit for f over some basis Ω. The basis Ω is normally indicated by writing $C_\Omega(f)$. We omit this index and assume that gates of the optimal circuits for $C(f)$ and $WL(f)$ are drawn from the same basis Ω. We assume that f depends on each of its n variables. The relationship between the wire length and the circuit size of a function is given by the following lemma.

Theorem 4. *The wire length* $WL(f)$ *of a function* $f : \{0,1\}^n \to \{0,1\}$ *relates to its circuit size* $C(f)$ *in the following manner*:

$$C(f) + n - 1 \leqslant WL(f) \leqslant \frac{1}{2}C(f)(C(f)-1) + n\max\{n, C(f)\}.$$

Proof. To show the first inequality, note that each input to the circuit as well as each gate of the circuit contributes to the output. Hence there is at least one edge from each input port to some gate and each gate except the output gate has fan-out at least one. Since gates and input ports are separated by unit distance, each such connection has at least unit length. The first inequality follows.

To show the second inequality, we construct a layout for some circuit C with circuit size $C(f)$. Since the circuit is feedforward, we can label the gates of C by $G_1, \ldots, G_{C(f)}$ such that G_i does not get input from gate G_j for all $1 \leqslant i < j \leqslant C(f)$. Arrange the gates on a row of the grid such that gate G_i is one unit to the left of G_{i+1} $(1 \leqslant i < C(f))$. In this arrangement all gates that receive input from some gate G_i are to the right of G_i (see Fig. 10). Since outputs may spread, the wire length to connect G_i to all of its successors is at most $C(f) - i$. This results in a wire length of $\frac{1}{2}C(f)(C(f)-1)$ for connections between gates of the circuit. Furthermore, arrange the input ports of the circuit on the row one unit above the gates. In the worst case, each input port is connected to each gate. The wire length needed to connect one of the n input ports with all the gates is bounded by n if $n > C(f)$ and by $C(f)$ if $n < C(f)$. Hence,

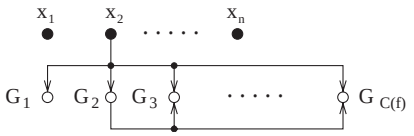

Fig. 10. A layout for an arbitrary circuit whose wire length can easily be estimated in terms of n and $C(f)$. Filled circles $x_1, \ldots, x_n$ are input ports and open circles $G_1, \ldots, G_{C(f)}$ are gates.

the wire length needed to connect input ports to gates is at most $n \max\{n, C(f)\}$. This yields the second summand in the claimed upper bound for $WL(f)$. □

Another interesting question is, how the wire length of a function f relates to the area needed to implement f in VLSI. For the VLSI-model discussed in Section 1 with gates of fan-in 2, we show that the wire length is bounded by the area needed to compute f.

Theorem 5. *If the function $f : \{0,1\}^n \to \{0,1\}$ can be computed in a feedforward manner in VLSI with μ layers, separation λ and area A, then the wire length of f is bounded by*

$$WL(f) = O\left(\frac{\mu}{\lambda^2} A\right).$$

Proof. We construct from a given VLSI-circuit for f a layout in our model for bounding its wire length. We first superimpose a grid of grid-width $\lambda/2$ and area A over the VLSI-layout. Since gates, ports and wires have at least width λ there is a grid-point in any gate, and any two grid-points in connected gates can be connected by a grid-path that runs in the area of the gates and their connecting wire.

In the following we will not distinguish between gates, input ports and output ports. Inputs ports can be treated as gates without inputs and output ports are simply gates without outputs. For each gate G in the VLSI layout, we use a single grid node n_G within the area of G to represent G in our model.

To connect a gate G with its successors $H_1, \ldots, H_m$, build a spanning tree on grid nodes and grid edges in the area of the VLSI-wires and gates from G to its successors that connects n_G with the corresponding input-nodes $n_{H_1}, \ldots, n_{H_m}$. We refer to this spanning tree as the *output-tree* of G. Hence, a gate G is connected to some gate H in the constructed layout, if and only if G is connected to H in the VLSI-circuit.

We will now bound the number of wires in the constructed layout that use a given grid-edge e. Consider an edge e of the grid-graph. Since wires in a layer are separated by at least λ and e has length $\lambda/2$, at most one VLSI-wire per layer intersects e (i.e. e is partly or fully in the area of this wire). Since there are μ VLSI-layers for wires, this shows that there are at most μ VLSI-wires that intersect e. Furthermore, since gates are separated by at least λ, at most one gate fully covers e. By construction, there is at most one output tree for each VLSI-wire and there are at most three output trees for each gate (a gate has its own output tree and the trees of at most two inputs). Since each tree uses e only once, e is used at most $\mu + 3$ times in the whole constructed graph.

We can bound the number of edges in a grid-graph of area A and grid-width $\lambda/2$ by $O(A/\lambda^2)$. Since each grid edge is used at most $\mu + 3$ times and grid edges have length 1 in our model for wire length, the wire length of the constructed layout is $O(\frac{\mu}{\lambda^2} A)$. □

6. Discussion

We have introduced a new complexity measure, wire length, that provides a useful criterion for judging whether a proposed circuit design is realistic from the point of view of a possible physical implementation in hardware or wetware. The relevance of the wire length of cortical circuits has previously been emphasized by numerous neuroscientists, from Cajal (see for example [4, p. 14]–[6]).

In Section 4, we have analyzed the wire length required for solving two concrete computational problems that are inherent in many global pattern recognition tasks. It turns out that both of these problems can be solved by circuits whose wire length is almost linear. Furthermore, these examples demonstrate that the design of circuits with small wire length yields circuit architectures that differ significantly from those that arise if just the traditional circuit complexity measures (number of gates, depth) are minimized. We expect that in general the construction of circuits with small wire length produces circuit architectures that are less unrealistic from the point of view of physical implementation. In particular, this strategy may help to "guess" circuit design strategies that are implemented in biological neural systems. We also show that the new complexity measure wire length is related to the complexity measure area in abstract VLSI-designs. However in contrast to VLSI-designs, which are necessarily much more detailed, it is in general much easier to estimate the wire length of a circuit architecture in the model that we have proposed in this article. Hence the new circuit complexity measure wire length may represent a useful compromise between practical relevance and mathematical simplicity.

Acknowledgements

We would like to thank two anonymous referees for numerous helpful suggestions regarding the presentation of the results of this article.

Appendix. Biological analysis of the model

In the cortex, neurons do not occupy the nodes of a 2-dimensional grid, but a roughly 2 mm thick 3-dimensional sheet of "gray matter". However, since there exists a strikingly general bound on the order of 10^5 for the number of neurons under any mm^2 of cortical surface, the density of neurons in these circuits remains bounded if the circuits are projected onto a 2-dimensional plane running parallel to the cortical surface (see Fig. 11). This observation provides the justification for the assumption of our abstract model that the neurons are positioned at the nodes of a 2-dimensional grid. It also yields a biologically realistic estimate for the length of an edge between two nodes in this grid: $10^{-5/2}$ mm. Since we are considering just the 2-dimensional projection of a 3-dimensional neural circuit, we can estimate in this way only the contribution of all horizontal components of all connections. However since there exist quite good estimates for the total amount of dendritic and axonal wire under any mm^2 of cortical surface (8 km according to [10]), we know that also the horizontal component of all connections adds up to at most 8 km. If one divides this number by the estimate 10^5 for the number of neurons under any mm^2, one arrives at an average wire length of 80 mm per neuron. Translated into our grid unit measure, this is equivalent to $80 \times 10^{5/2} = 25,300$ grid units. The total bound of $25,300\,j$ grid units for the wire length of cortical circuits with j neurons is likely to be an overestimate, since the preceding argument assumes

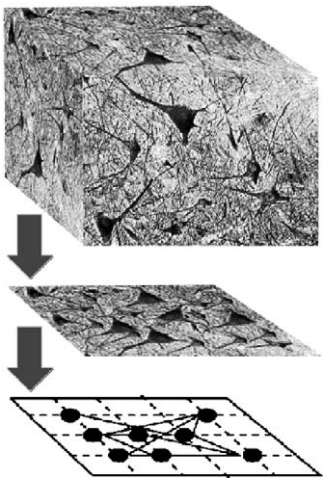

Fig. 11. The relationship between cortical circuitry and a simple mathematical model is illustrated by a projection onto a 2-dimensional plane.

that all of the 8 km of wires under a mm^2 of cortical surface can be used for horizontal connections. In this setup we arrive at a heuristic condition for any abstract circuit design with j neurons to be biologically realistic: it must have an implementation in our 2-dimensional grid model with a wire length of at most $25,300\,j$ grid units.

In circuit complexity theory it is customary to express the total amount of resources used in terms of the number n of circuit inputs. For the sake of simplicity we denote in the formal results of this article the number of pixels by n, and the actual number of circuit inputs is some constant multiple of n. Several empirical studies provide estimates for the order of magnitude of the number n of inputs, the number of neurons, and the length of axons and dendrites ("wires") in biological neural circuits for sensory processing, see [1,10,18,3]. [2]

[2] The number of neurons that transmit information from the retina (via the thalamus) to the cortex is estimated to be around 10^6 (all estimates given are for primates, and they only reflect the order of magnitude). The number of neurons in the primary visual cortex of primates is estimated to be around 10^9, occupying an area of roughly $10^4\,mm^2$ of cortical surface. Since the total length of axonal and dendritic branches below $1\,mm^2$ of cortical surface is estimated to be at most 8 km, this yields an upper bound of 10^{11} mm for the wire length of the primary visual cortex. Thus if one assumes for example that 100 separate circuits are implemented in the primary visual cortex, each of them can use 10^7 neurons and a wire length of 10^9 mm. If one writes these estimates as powers of the number $n = 10^6$ of inputs, this amounts to $10^7 = n^{7/6}$ neurons and a wire length of $10^{11.5} < n^2$ grid units (in the framework of our model).
The whole cortex receives sensory input from about 10^8 neurons. It processes this input with about 10^{10} neurons and less than 10^{12} mm wire length. If one assumes that 10^3 separate circuits process this sensory information in parallel, each of them processing about $1/10$th of the input, one arrives at $n = 10^7$ inputs for each circuit, and an average circuit can use on the order of n neurons and a wire length of $10^{11.5} < n^2$ grid units. The actual resources available for sensory processing are likely to be substantially smaller, since most cortical neurons and circuits are believed to have many other functions (for example related to memory, learning and attention) besides online sensory processing. The purpose of this is to provide some concrete numbers for the amount of resources available in cortical circuits. It is of course impossible to derive support for specific asymptotic complexity measures from such sparse data.

References

[1] M. Abeles, Corticonics: Neural Circuits of the Cerebral Cortex, Cambridge University Press, Cambridge, 1999.
[2] S. Agarwal, D. Roth, Learning a sparse representation for object detection, in: Proceedings of the Seventh European Conference on Computer Vision,2002, pp. 113–130.
[3] V. Braitenberg, A. Schüz, Cortex: Statistics and Geometry of Neuronal Connectivity, 2nd Edition, Springer, Berlin, 1998.
[4] S.R. Cajal, Histology of the Nervous System, vols. 1 and 2, Oxford University Press, New York, 1995.
[5] D.B. Chklovskii, Binocular disparity can explain the orientation of ocular dominance stripes in primate primary visual area (V1), Vis. Res. 40 (13) (2000) 1765–1773.
[6] D.B. Chklovskii, A.A. Koulakov, A wire length minimization approach to ocular dominance patterns in mammalian visual cortex, Phys. A 284 (1–4) (2000) 318–334.
[7] D.B. Chklovskii, C.F. Stevens, Wiring optimization in the brain, Advances in Neural Information Processing Systems, vol. 12, MIT Press, Cambridge, MA, 2000, pp. 103–107.
[8] L. Fei-Fei, R. Fergus, P. Perona, A Bayesian approach to unsupervised one-shot learning of object categories, Proceedings of the Ninth International Conference on Computer Vision, vol. 2, 2003, pp. 1134–1141.
[9] C.-T. Ho, R. Agrawal, N. Megiddo, R. Srikant, Range queries in OLAP data cubes, in: Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data, Tucson, AZ, 1997, pp. 73–88.
[10] C. Koch, Biophysics of Computation, Oxford University Press, Oxford, 1999.
[11] J. Lazzaro, S. Ryckebusch, M.A. Mahowald, C.A. Mead, Winner-take-all networks of $O(n)$ complexity, Advances in Neural Information Processing Systems, vol. 1, Morgan Kaufmann, San Mateo, 1989, pp. 703–711.
[12] R.A. Legenstein, W. Maass, Neural circuits for pattern recognition with small total wire length, Theoret. Comput. Sci. 287 (2002) 239–249.
[13] C.E. Leiserson, Fat-trees: universal networks for hardwareefficient supercomputing, IEEE Trans. Comput. C-34 (1985) 892–901.
[14] C. Mead, Analog VLSI and Neural Systems, Addison-Wesley, Reading, MA, USA, 1989.
[15] C. Mead, M. Rem, Cost and performance of VLSI computing structures, IEEE J. Solid State Circuits SC-14 (1979) 455–462.
[16] G. Mitchison, Axonal trees and cortical architecture, Trends Neurosci. 15 (4) (1992) 22–26.
[17] J.E. Savage, Models of Computation: Exploring the Power of Computing, Addison-Wesley, Reading, MA, USA, 1998.
[18] G.M. Shepherd, The Synaptic Organization of the Brain, 2nd Edition, Oxford University Press, Oxford, 1998.
[19] M. Weber, M. Welling, P. Perona, Unsupervised learning of models for recognition, in: Proceedings of the Sixth European Conference on Computer Vision,2000, pp. 101–108.