Weak Models of Distributed Computing, with Connections to Modal Logic

Lauri Hella · Matti Järvisalo · Antti Kuusisto · Juhana Laurinharju · Tuomo Lempiäinen · Kerkko Luosto · Jukka Suomela · Jonni Virtema

Abstract This work presents a classification of weak models of distributed computing. We focus on deterministic distributed algorithms, and study models of computing that are weaker versions of the widely-studied port-numbering model. In the port-numbering model, a node of degree d receives messages through d input ports and sends messages through d output ports, both numbered with $1, 2, \ldots, d$. In this work, $\mathsf{VV_c}$ is the class of all *graph problems* that can be solved in the standard port-numbering model. We study the following subclasses of $\mathsf{VV_c}$:

- VV: Input port i and output port i are not necessarily connected to the same neighbour.
- MV: Input ports are not numbered; algorithms receive a multiset of messages.
- SV: Input ports are not numbered; algorithms receive a set of messages.
- VB: Output ports are not numbered; algorithms send the same message to all output ports.
- MB: Combination of MV and VB.
- SB: Combination of SV and VB.

Now we have many trivial containment relations, such as $\mathsf{SB} \subseteq \mathsf{MB} \subseteq \mathsf{VB} \subseteq \mathsf{VV} \subseteq \mathsf{VV_c}$, but it is not obvious if, for example, either of $\mathsf{VB} \subseteq \mathsf{SV}$ or $\mathsf{SV} \subseteq \mathsf{VB}$ should hold. Nevertheless, it turns out that we can identify a *linear order* on these classes. We prove that $\mathsf{SB} \subsetneq \mathsf{MB} = \mathsf{VB} \subsetneq \mathsf{SV} = \mathsf{MV} = \mathsf{VV} \subsetneq \mathsf{VV_c}$. The same holds for the constant-time versions of these classes.

We also show that the constant-time variants of these classes can be characterised by a corresponding *modal logic*. Hence the linear order identified in this work has direct implications in the study of the expressibility of modal logic. Conversely, one can use tools from modal logic to study these classes.

L. Hella · K. Luosto · J. Virtema
School of Information Sciences, University of Tampere, Finland

M. Järvisalo · J. Laurinharju · T. Lempiäinen · J. Suomela
Helsinki Institute for Information Technology HIIT, Department of Computer Science, University of Helsinki, Finland

A. Kuusisto
Institute of Computer Science, University of Wrocław, Poland

Keywords Distributed computing · Local algorithms · Modal logic · Models of computation

1 Introduction

We introduce seven complexity classes, $\mathsf{VV_c}$, VV, MV, SV, VB, MB, and SB, each defined as the class of *graph problems* that can be solved with a deterministic distributed algorithm in a certain variant of the widely-studied *port-numbering model*. We present a *complete characterisation* of the containment relations between these classes, as well as their constant-time counterparts, and identify connections between these classes and questions related to *modal logic*.

1.1 State Machines

For our purposes, a distributed algorithm is best understood as a state machine $\mathcal{A}$. In a distributed system, each node is a copy of the same state machine $\mathcal{A}$. Computation proceeds in synchronous steps. In each step, each machine

1. sends messages to its neighbours,
2. receives messages from its neighbours, and
3. updates its state based on the messages that it received.

If the new state is a stopping state, the machine halts.

Let us now formalise the setting studied in this work. We use the notation $[k] = \{1, 2, \ldots, k\}$. For each positive integer Δ, let $\mathcal{F}(\Delta)$ consist of all simple undirected graphs

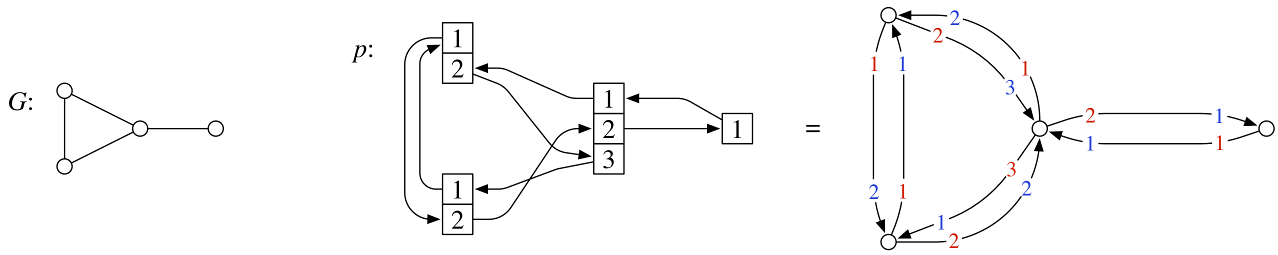


Fig. 1 A port numbering p of graph G. Here we present p using two different notations; in the illustration on the left, the ports are explicitly shown, while in the illustration on the right, the ports are given as the labels of the edges.

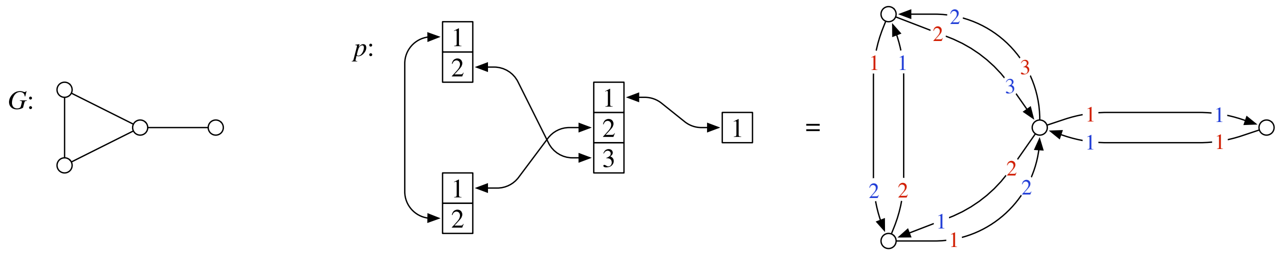


Fig. 2 A consistent port numbering.

of maximum degree at most Δ. A *distributed state machine* for $\mathcal{F}(\Delta)$ is a tuple $\mathcal{A} = (Y, Z, z_0, M, m_0, \mu, \delta)$, where

- Y is a finite set of stopping states,
- Z is a (possibly infinite) set of intermediate states such that $Y \cap Z = \emptyset$,
- $z_0 \colon \{0, 1, \dots, \Delta\} \to Y \cup Z$ defines the initial state depending on the degree of the node,
- M is a (possibly infinite) set of messages,
- $m_0 \in M$ is a special symbol for "no message",
- $\mu \colon Z \times [\Delta] \to M$ is a function that constructs the outgoing messages,
- $\delta \colon Z \times M^{\Delta} \to Y \cup Z$ defines the state transitions.

To simplify the notation, we extend the domains of μ and δ to cover the stopping states: for all $y \in Y$, we define $\mu(y, i) = m_0$ for any $i \in [\Delta]$, and $\delta(y, \mathbf{m}) = y$ for any $\mathbf{m} \in M^{\Delta}$. In other words, a node that has stopped does not send any messages and does not change its state any more.

1.2 Port Numbering

Now consider a graph $G = (V, E) \in \mathcal{F}(\Delta)$. We write $\deg(v)$ for the degree of node $v \in V$. A *port* of G is a pair (v, i) where $v \in V$ and $i \in [\deg(v)]$. Let $P(G)$ be the set of all ports of G. Let $p \colon P(G) \to P(G)$ be a bijection. Define

$$A(p) = \big\{(u, v) : u \in V,\, v \in V, \text{ and } \\ p((u, i)) = (v, j) \text{ for some } i \text{ and } j\big\},$$

$$A(G) = \big\{(u, v) : \{u, v\} \in E\big\}.$$

We say that p is a *port numbering* of G if $A(p) = A(G)$; see Fig. 1 for an example. The intuition here is that a node $v \in V$ has $\deg(v)$ communication ports; if it sends a message to its port (v, i), and $p((v, i)) = (u, j)$, the message will be received by its neighbour u from port (u, j).

We say that a port numbering is *consistent* if p is an involution, that is,

$$p\big(p((v, i))\big) = (v, i) \text{ for all } (v, i) \in P(G).$$

See Fig. 2 for an example.

1.3 Execution of a State Machine

For a fixed distributed state machine $\mathcal{A}$, a graph G, and a port numbering p, we can define the *execution* of $\mathcal{A}$ in (G, p) recursively as follows.

The state of the system at time $t = 0, 1, \dots$ is represented as a state vector $x_t \colon V \to Y \cup Z$. At time 0, we have

$$x_0(u) = z_0(\deg(u))$$

for each $u \in V$.

Now assume that we have defined the state x_t at time t. Let $(u, i) \in P(G)$ and $(v, j) = p^{-1}((u, i))$. Define

$$a_{t+1}(u, i) = \mu(x_t(v), j).$$

In words, $a_{t+1}(u, i)$ is the message received by node u from port (u, i) in round $t + 1$, or equivalently the message sent by node v to port (v, j). For each $u \in V$ we define a vector $\mathbf{a}_{t+1}(u)$ of length Δ as follows:

$$\mathbf{a}_{t+1}(u) = \big(a_{t+1}(u, 1), a_{t+1}(u, 2), \dots, a_{t+1}(u, \deg(u)), \\ m_0, m_0, \dots, m_0\big).$$

In other words, we simply take all messages received by u, in the order of increasing port number; the padding with the dummy messages m_0 is just for notational convenience so that $\mathbf{a}_{t+1}(u) \in M^\Delta$. Finally, we define the new state of a node $u \in V$ as follows:

$$x_{t+1}(u) = \delta(x_t(u), \mathbf{a}_{t+1}(u)).$$

We say that $\mathcal{A}$ *stops in time* T in (G,p) if $x_T(u) \in Y$ for all $u \in V$. If $\mathcal{A}$ stops in time T in (G,p), we say that $S = x_T$ is the *output* of $\mathcal{A}$ in (G,p). Here $S(u) = x_T(u)$ is the *local output* of $u \in V$.

1.4 Graph Problems

A *graph problem* is a function Π that associates with each undirected graph $G = (V,E)$ a set $\Pi(G)$ of *solutions*. Each solution $S \in \Pi(G)$ is a mapping $S\colon V \to Y$; here Y is a finite set that does not depend on G.

We emphasise that this definition is by no means universal; however, it is convenient for our purposes and covers a wide range of classical graph problems:

- Finding a *subset of vertices*. A typical example is the task of finding a *maximal independent set*: $Y = \{0,1\}$, and each solution S is the indicator function of a maximal independent set.
- Finding a *partition of vertices*. A typical example is the task of finding a *vertex* 3*-colouring*: $Y = \{1,2,3\}$, and each solution S is a valid 3-colouring of the graph.
- Deciding *graph properties*. A typical example is deciding if a graph is *Eulerian*: Here $Y = \{0,1\}$. If G is Eulerian, there is only one solution S with $S(v) = 1$ for all $v \in V$. If G is not Eulerian, valid solutions are mappings S such that $S(v) = 0$ for at least one $v \in V$. Put otherwise, all nodes must accept a yes-instance, and at least one node must reject a no-instance.

The idea is that a distributed state machine $\mathcal{A}$ *solves* a graph problem Π if, for any graph G and for any port numbering of G, the output of $\mathcal{A}$ is a valid solution $S \in \Pi(G)$. However, the fact that we study graphs of bounded degree requires some care; hence the following somewhat technical definition.

Let Π be a graph problem. Let $T : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. Let $\mathbf{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots)$ be a sequence of distributed state machines. We say that $\mathbf{A}$ *solves* Π *in time* T if the following hold for any $\Delta \in \mathbb{N}$, any graph $G \in \mathcal{F}(\Delta)$, and any port numbering p of G:

1. State machine $\mathcal{A}_\Delta$ stops in time $T(\Delta, |V|)$ in (G,p).
2. The output of $\mathcal{A}_\Delta$ is in $\Pi(G)$.

We say that $\mathbf{A}$ *solves* Π *in time* T *assuming consistency* if the above holds for any consistent port numbering p of G. Note that we do not require that $\mathcal{A}_\Delta$ stops if the port numbering happens to be inconsistent.

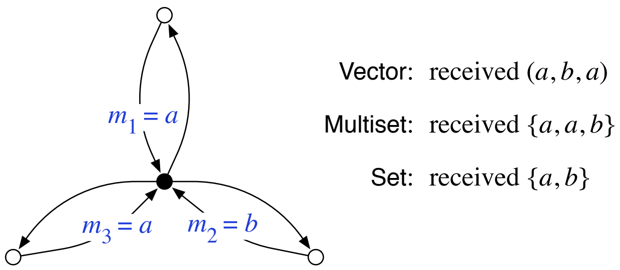


Fig. 3 Comparison of Vector, Multiset, and Set.

We say that $\mathbf{A}$ *solves* Π or $\mathbf{A}$ *is an algorithm for* Π if there is any function T such that $\mathbf{A}$ solves Π in time T. We say that $\mathbf{A}$ *solves* Π *in constant time* or $\mathbf{A}$ *is a local algorithm for* Π if $T(\Delta,n) = T'(\Delta)$ for some $T' : \mathbb{N} \to \mathbb{N}$, independently of n.

Remark 1 We emphasise that the term "constant time" refers to the case of a fixed Δ. We only require that for each given Δ the running time of state machine $\mathcal{A}_\Delta$ on graph family $\mathcal{F}(\Delta)$ is bounded by a constant. That is, "local algorithms" are $O(1)$-time algorithms on any graph family of maximum degree $O(1)$.

1.5 Algorithm Classes

Now we are ready to introduce the concepts studied in this work: variants of the definition of a distributed algorithm.

For a vector $\mathbf{a} = (a_1, a_2, \dots, a_\Delta) \in M^\Delta$ we define

$$\begin{aligned} \mathrm{set}(\mathbf{a}) &= \{a_1, a_2, \dots, a_\Delta\}, \\ \mathrm{multiset}(\mathbf{a}) &= \big\{(m,n) : m \in M,\ n = |\{i \in [\Delta] : m = a_i\}|\big\}. \end{aligned}$$

In other words, multiset($\mathbf{a}$) discards the ordering of the elements of $\mathbf{a}$, and set($\mathbf{a}$) furthermore discards the multiplicities.

Let Vector be the set of all distributed state machines $\mathcal{A}$, as defined in Section 1.1. We define three subclasses of distributed state machines, Set $\subseteq$ Multiset $\subseteq$ Vector, and Broadcast $\subseteq$ Vector:

- $\mathcal{A} \in$ Multiset if multiset($\mathbf{a}$) $=$ multiset($\mathbf{b}$) implies that $\delta(x,\mathbf{a}) = \delta(x,\mathbf{b})$ holds for all $x \in Z$.
- $\mathcal{A} \in$ Set if set($\mathbf{a}$) $=$ set($\mathbf{b}$) implies that $\delta(x,\mathbf{a}) = \delta(x,\mathbf{b})$ holds for all $x \in Z$.
- $\mathcal{A} \in$ Broadcast if $\mu(x,i) = \mu(x,j)$ holds for all $x \in Z$ and $i,j \in [\Delta]$.

Classes Multiset and Set are related to *incoming* messages; see Fig. 3 for an example. Intuitively, a state machine in class Vector considers a *vector* of incoming messages, while a state machine in Multiset considers a *multiset* of incoming messages, and a state machine in Set considers a *set* of incoming messages. In particular, state machines in Multiset and Set do not have any access to the numbering of incoming ports.

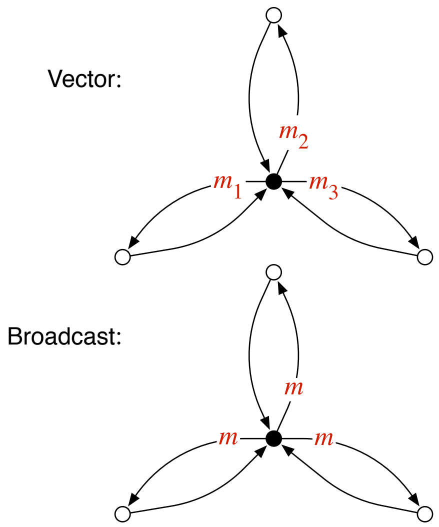


Fig. 4 Comparison of Vector and Broadcast.

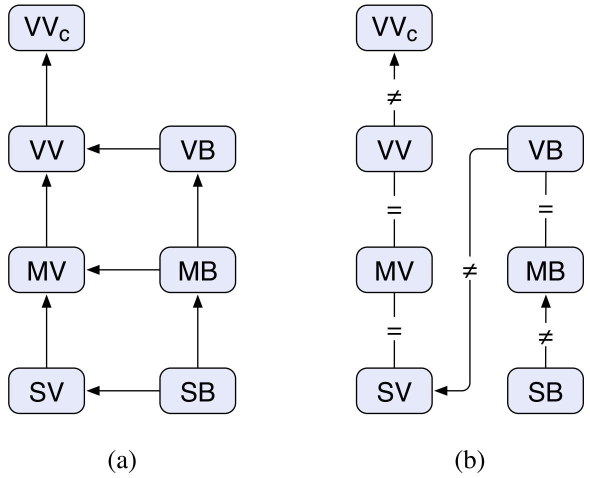


Fig. 5 Classes of graph problems. (a) Trivial subset relations between the classes. (b) The linear order identified in this work.

Class Broadcast is related to *outgoing* messages; see Fig. 4 for an example. Intuitively, a state machine in class Vector constructs a *vector* of outgoing messages, while a state machine in Broadcast can only *broadcast* the same message to all neighbours. In particular, state machines in Broadcast do not have any access to the numbering of outgoing ports.

We extend the definitions to sequences of state machines in a natural way:

$$\begin{aligned}
\mathbf{Vector} &= \{(\mathcal{A}_1, \mathcal{A}_2, \dots) : \mathcal{A}_\Delta \in \mathsf{Vector} \text{ for all } \Delta\}, \\
\mathbf{Multiset} &= \{(\mathcal{A}_1, \mathcal{A}_2, \dots) : \mathcal{A}_\Delta \in \mathsf{Multiset} \text{ for all } \Delta\}, \\
\mathbf{Set} &= \{(\mathcal{A}_1, \mathcal{A}_2, \dots) : \mathcal{A}_\Delta \in \mathsf{Set} \text{ for all } \Delta\}, \\
\mathbf{Broadcast} &= \{(\mathcal{A}_1, \mathcal{A}_2, \dots) : \mathcal{A}_\Delta \in \mathsf{Broadcast} \text{ for all } \Delta\}.
\end{aligned}$$

From now on, we will use the word *algorithm* to refer to both distributed state machines $\mathcal{A} \in \mathsf{Vector}$ and to sequences of distributed state machines $\mathbf{A} \in \mathbf{Vector}$, when there is no risk of confusion.

1.6 Problem Classes

So far we have defined classes of algorithms; now we will define seven classes of problems:

1. $\Pi \in \mathsf{VV_c}$ if there exists an algorithm $\mathbf{A} \in \mathbf{Vector}$ that solves problem Π assuming consistency,
2. $\Pi \in \mathsf{VV}$ if there exists an algorithm $\mathbf{A} \in \mathbf{Vector}$ that solves problem Π,
3. $\Pi \in \mathsf{MV}$ if there exists an algorithm $\mathbf{A} \in \mathbf{Multiset}$ that solves problem Π,
4. $\Pi \in \mathsf{SV}$ if there exists an algorithm $\mathbf{A} \in \mathbf{Set}$ that solves problem Π,
5. $\Pi \in \mathsf{VB}$ if there exists an algorithm $\mathbf{A} \in \mathbf{Broadcast}$ that solves problem Π,
6. $\Pi \in \mathsf{MB}$ if there exists an algorithm $\mathbf{A} \in \mathbf{Multiset} \cap \mathbf{Broadcast}$ that solves problem Π,
7. $\Pi \in \mathsf{SB}$ if there exists an algorithm $\mathbf{A} \in \mathbf{Set} \cap \mathbf{Broadcast}$ that solves problem Π.

We will also define the constant-time variants of the classes:

1. $\Pi \in \mathsf{VV_c}(1)$ if there exists a local algorithm $\mathbf{A} \in \mathbf{Vector}$ that solves problem Π assuming consistency,
2. $\Pi \in \mathsf{VV}(1)$ if there exists a local algorithm $\mathbf{A} \in \mathbf{Vector}$ that solves problem Π,

   ...

Note that consistency is irrelevant for all other classes; we only define the consistent variants of VV and VV(1). The classes are summarised in Fig. 5a. Fig. 6 summarises what information is available to an algorithm in each class.

Remark 2 In each problem class, we consider algorithms in which each node knows its own degree. While this is natural in all other cases, it may seem odd in the case of class SB. In principle, we could define yet another class of problems $\mathsf{SB_o}$, defined in terms of *degree-oblivious* algorithms in $\mathbf{Set} \cap \mathbf{Broadcast}$, i.e., algorithms with a constant initialisation function z_0. However, it is easy to see that $\mathsf{SB_o}$ is entirely trivial—in essence, one can only solve the problem of distinguishing non-isolated nodes from isolated nodes—while there are many non-trivial problems that we can solve in class SB. In particular, it is trivial to prove that $\mathsf{SB_o} \subsetneq \mathsf{SB}$. Hence we will not consider class $\mathsf{SB_o}$ in this work. However, class $\mathsf{SB_o}$ is more interesting if one considers labelled graphs; see Section 3.4.

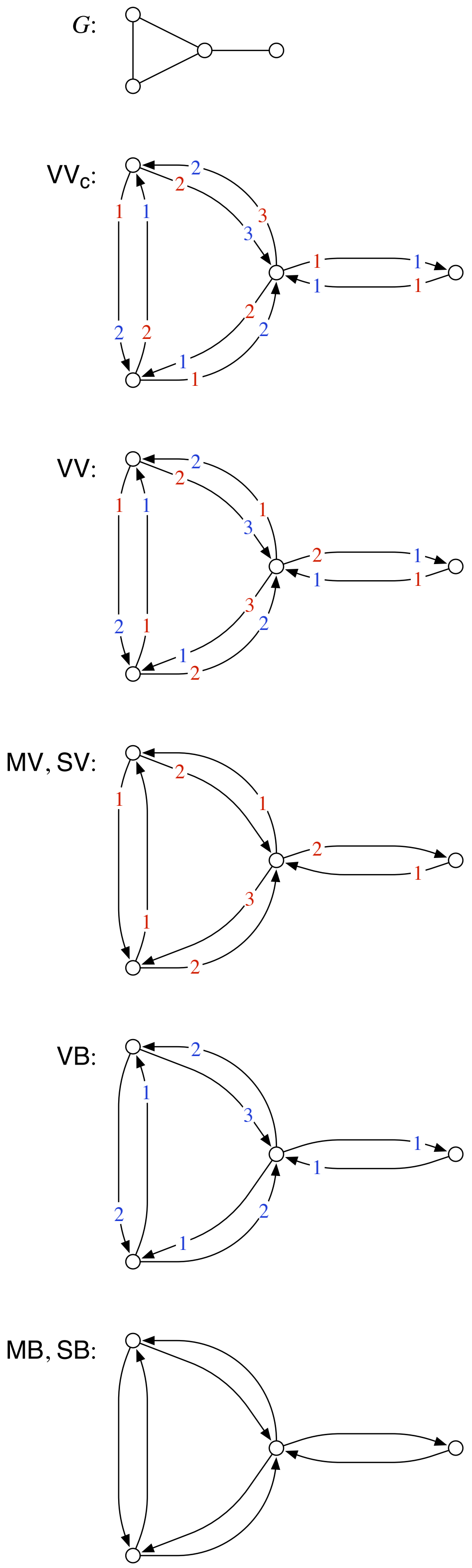


Fig. 6 Auxiliary information available to a distributed algorithm in each class.

2 Contributions

This work is a systematic study of the complexity classes $\mathsf{VV_c}$, VV, MV, SV, VB, MB, and SB, as well as their constant-time counterparts. Our main contributions are two-fold.

First, we present a complete characterisation of the containment relations between these classes. The definitions of the classes imply the partial order depicted in Fig. 5a. For example, classes VB and SV are seemingly orthogonal, and it would be natural to assume that neither $\mathsf{VB} \subseteq \mathsf{SV}$ nor $\mathsf{SV} \subseteq \mathsf{VB}$ holds. However, we show that this is not the case. Unexpectedly, the classes form a linear order (see Fig. 5b):

$$\mathsf{SB} \subsetneq \mathsf{MB} = \mathsf{VB} \subsetneq \mathsf{SV} = \mathsf{MV} = \mathsf{VV} \subsetneq \mathsf{VV_c}. \tag{1}$$

In summary, instead of seven classes that are possibly distinct, we have precisely four distinct classes. These four distinct classes of problems can be concisely characterised as follows, from the strongest to the weakest:

1. consistent port numbering (class $\mathsf{VV_c}$),
2. no incoming port numbers (class SV and equivalent),
3. no outgoing port numbers (class VB and equivalent),
4. neither (class SB).

We also show an analogous result for the constant-time versions:

$$\begin{aligned} \mathsf{SB}(1) &\subsetneq \mathsf{MB}(1) = \mathsf{VB}(1) \\ &\subsetneq \mathsf{SV}(1) = \mathsf{MV}(1) = \mathsf{VV}(1) \\ &\subsetneq \mathsf{VV_c}(1). \end{aligned} \tag{2}$$

The main technical achievement here is proving that $\mathsf{SV}(1) = \mathsf{MV}(1)$ and $\mathsf{SV} = \mathsf{MV}$. This together with the ideas of a prior work [5] leads to the linear orders (1) and (2).

As our second contribution, we identify a novel connection between distributed computational complexity and modal logic. In particular, classes $\mathsf{VV_c}(1)$, $\mathsf{VV}(1)$, $\mathsf{MV}(1)$, $\mathsf{SV}(1)$, $\mathsf{VB}(1)$, $\mathsf{MB}(1)$, and $\mathsf{SB}(1)$ have natural characterisations using certain variants of modal logic. This correspondence allows one to apply tools from the field of modal logic—in particular, bisimulation—to facilitate the proofs of (1) and (2). Conversely, we can lift our results from the field of distributed algorithms to modal logic, by re-interpreting the relations identified in (2).

Some of the equivalences between the classes are already known by prior work—in particular, results that are similar to $\mathsf{MB} = \mathsf{VB}$ and $\mathsf{MV} = \mathsf{VV} \subsetneq \mathsf{VV_c}$ are implied by e.g., Boldi et al. [10] and Yamashita and Kameda [62]. The main differences between our work and prior work can be summarised as follows.

1. All results related to classes SV and SB are new. In particular, we are not aware of any prior work that has studied class SV in this context.

2. We approach the classification from the perspective of *locality*. We not only prove the equivalences $\mathsf{MB} = \mathsf{VB}$ and $\mathsf{SV} = \mathsf{MV} = \mathsf{VV}$ but also show that in each case the simulation of the stronger model is *efficient*. The nodes do not need to know any global information on the graph in advance (such as an upper bound on the size of the graph), and the nodes do not need to gather any information beyond their constant-radius neighbourhood. Our proofs yield the identical collapses for the constant-time versions of the classes: $\mathsf{MB}(1) = \mathsf{VB}(1)$ and $\mathsf{SV}(1) = \mathsf{MV}(1) = \mathsf{VV}(1)$. Similarly, our separation results only rely on problems that can be solved in constant time in one of the classes, without any global information.
3. The focus on locality also enables us to introduce the connection with *modal logic*. We show how to derive all separations between the complexity classes with bisimulation arguments.

We will discuss related work in more detail in Section 3; see also Tables 1 and 2.

3 Motivation and Related Work

In this work, we study *deterministic* distributed algorithms in *anonymous* networks—all state transitions are deterministic, all nodes run the same algorithm, and initially each node knows only its own degree. This is a fairly weak model of computation, and traditionally research has focused on stronger models of distributed computing.

3.1 Stronger Models

There are two obvious extensions:

1. *Networks with unique identifiers*: Initially, all nodes are labelled with $O(\log n)$-bit, globally unique identifiers. With this extension, we arrive at Linial's [42] model of computation; Peleg [49] calls it the LOCAL model.
2. *Randomised distributed algorithms*: The nodes have access to a stream of random bits. The state transitions can depend on the random bits.

Both of these extensions lead to a model that is strictly stronger than any of the models studied in this work. The problem of finding a maximal independent set is a good example of a graph problem that separates the weak models from the above extensions. The problem is clearly not in VV_c—a cycle with a symmetric port numbering is a simple counterexample—while it is possible to find a maximal independent set *fast* in both of the above models.

3.2 Port-Numbering Model (VV_c)

While most of the attention is on stronger models, one of the weaker models has been studied extensively since the 1980s. Unsurprisingly, it is the strongest of the family, model VV_c, and it is commonly known as the *port-numbering model* in the literature.

The study of the port-numbering model was initiated by Angluin [2] in 1980. Initially the main focus was on problems that have a *global* nature—problems in which the local output of a node necessarily depends on the global properties of the input. Examples of papers from the first two decades after Angluin's pioneering work include Attiya et al. [6], Yamashita and Kameda [59–61], and Boldi and Vigna [13], who studied global functions, leader election problems, spanning trees, and topological properties.

Based on the earlier work, research related to the port-numbering model may look like a dead end: positive results were rare. However, very recently, distributed algorithms in the port-numbering model have become an increasingly important research topic—and surprisingly, the study of the port-numbering model is now partially motivated by the desire to understand distributed computing in *stronger* models of computation.

The background is in the study of *local algorithms*, i.e., constant-time distributed algorithms [55]. The research direction was initiated by Naor and Stockmeyer [46] in 1995, and initially it looked like another area where most of the results are negative—after all, it is difficult to imagine a non-trivial graph problem that could be solved in constant time. However, since 2005, we have seen a large number of local algorithms for a wide range of graph problems: these include algorithms for vertex covers [3, 5, 36, 37, 45, 51, 56], matchings [4, 31], dominating sets [20, 39–41], edge dominating sets [54], set covers [5, 36, 37], semi-matchings [19], stable matchings [31], and linear programming [26–30, 36, 37]. Naturally, most of these algorithms are related to approximations and special cases, but nevertheless the sheer number of such algorithms is a good demonstration of the unexpected capabilities of local algorithms.

At first sight, constant-time algorithms in stronger models and distributed algorithms in the port-numbering model seem to be orthogonal concepts. However, in many cases a local algorithm is also an algorithm in the port-numbering model. Indeed, a formal connection between local algorithms and the port-numbering model has been recently identified [32].

3.3 Weaker Models

As the study of the port-numbering model has been recently revived, now is the right time to ask if it is justified to use VV_c as the standard model in the study of anonymous networks.

Algorithm class	Problem class	Term	References
Vector	$\mathsf{VV_c}$	port numbering	[2]
		local edge labelling	[60]
		local orientation	[16, 25]
		orientation	[44]
		complete port awareness	[10]
		monoid graph	[47]
		port-to-port	[58, 62]
		port-à-port	[15]
Vector	VV	input/output port awareness	[10]
Multiset	MV	output port awareness	[10]
		wireless in input	[11]
		mailbox	[11]
		port-to-mailbox	[58, 62]
		port-à-boîte	[15]
Set	SV	—	
Broadcast	VB	input port awareness	[10]
		wireless in output	[11]
		broadcast	[11, 58]
		broadcast-to-port	[62]
		diffusion-à-port	[15]
Multiset ∩ Broadcast	MB	totalistic	[57]
		wireless	[11, 22, 48]
		broadcast-to-mailbox	[62]
		diffusion-à-boîte	[15]
		mailbox-to-mailbox	[58]
		network without colours	[13]
		broadcast	[5]
		(no name)	[38]
Set ∩ Broadcast	SB	beeping	[1, 18]

Table 1 Prior work related to the weak models, and a summary of the related terminology. We have identified the closest equivalent in our classification, not necessarily an exact match—see also Table 2 for an overview of the main differences.

Focuses on the case of a known topology $G = (V, E)$, a known $\lvert V \rvert$, or a known upper bound on $\lvert V \rvert$.	[10, 13, 16, 48, 60, 62]
Proves equivalences between the models from a *global* perspective; the simulation overhead can be linear in $\lvert V \rvert$. Our work shows that the equivalences hold also from a *local* perspective; the simulation overhead is bounded by a constant.	[10, 62]
Studies functions that map the local inputs of the nodes to specific local outputs of the nodes. Our work studies graph problems—the local outputs depend on the structure of G, not on the local inputs.	[11, 47, 48]
Considers the problem of deciding whether a given problem can be solved in a given graph. In our work, we are interested in the existence of a problem and a graph that separates two models.	[13, 47, 48, 61]
Studies individual problems, not classes of problems.	[1, 5, 10, 38, 58, 61, 62]
Provides general results, but does not study the implications from the perspective of the weak models and their relative strength.	[12, 13]
Does not consider models that are weaker than the port-numbering model.	[2, 6, 16, 43, 60, 61]
Assumes a specific network structure (cycle, grid, etc.), or auxiliary information in local inputs.	[6, 22, 25, 38, 44, 57]
Studies randomised, asynchronous algorithms.	[1, 23]

Table 2 Main differences in the problem setting between this work and selected prior work.

First, the definition is somewhat arbitrary—it is not obvious that $\mathsf{VV_c}$ is the "right" class, instead of VV, for example. Second, while the existence of a port numbering is easily justified in the context of wired networks, weaker models such as Broadcast and Set seem to make more sense from the perspective of wireless networks.

If we had no positive examples of problems in classes below $\mathsf{VV_c}$, there would be little motivation for pursuing further. However, the recent work related to the vertex cover problem [5] calls for further investigation. It turned out that 2-approximation of vertex cover is a graph problem that is not only in $\mathsf{VV_c}(1)$, but also in $\mathsf{MB}(1)$—that is, we have a non-trivial graph problem that does not require any access to either outgoing or incoming port numbers. One ingredient of the vertex cover algorithm is the observation that $\mathsf{MB}(1) = \mathsf{VB}(1)$, which raises the question of the existence of other similar collapses in the hierarchy of weak models.

We are by no means the first to investigate the weak models. Computation in models that are strictly weaker than the standard port-numbering model has been studied since the 1990s, under various terms—see Table 1 for a summary of terminology, and Table 2 for an overview of the main differences in the research directions. Questions related to specific problems, models, and graph families have been studied previously, and indeed many of the techniques and ideas that we use are now standard—this includes the use of symmetry and isomorphisms, local views, covering graphs (lifts) and universal covering graphs, and factors and factorisations. Mayer, Naor, and Stockmeyer [43, 46] made it explicit that the parity of node degrees makes a huge difference in the port-numbering model, and Yamashita and Kameda [60] discussed factors and factorisations in this context; the underlying graph-theoretic observations can be traced back to as far as Petersen's 1891 work [50]. Some equivalences and separations between the classes are already known, or at least implicit in prior work—see, in particular, Boldi et al. [10] and Yamashita and Kameda [62].

However, it seems that a comprehensive classification of the weak models from the perspective of solvable graph problems has been lacking. Our main contribution is putting all pieces together in order to provide a complete characterisation of the relations between the weak models and the complexity classes associated with them.

We also advocate a new perspective for studying the weak models—the connections with modal logic can be used to complement the traditional graph-theoretic approaches. In particular, bisimulation is a convenient tool that complements the closely related graph-theoretic concepts of covering graphs and fibrations.

3.4 Local Inputs

In this article we study graph problems associated with simple undirected graphs of the type (V,E). It would also be worthwhile to study structures of the type (V,E,f), where $f\colon V \to \mathbb{N}$ is function encoding a *local input* $f(u)$ associated with each node $u \in V$. The related notion of a state machine would be the same as in Section 1.3, with the additional property that the initial state $x_0(u)$ of a machine at a node u would depend on the local information $f(u)$ in addition to the degree of u.

While we will not study the effects of local inputs, it is worth noticing that the classification given by (1) and (2) extends immediately to the context with local information—in particular, a separation with unlabelled graphs implies a separation in the more general case of labelled graphs.

As long as each node knows its own degree, local inputs do not seem to add anything interesting to the classification of weak models of distributed computing—a uniformly finite amount of local information could be encoded in the topological information of the graph. However, if we studied models that are strictly weaker than SB (for example, model SB_o that we briefly mentioned in Remark 2), local inputs would be necessary in order to arrive at non-trivial results.

3.5 Distributed Algorithms and Modal Logic

Modal logic (see Section 4) has, of course, been applied previously in the context of distributed systems. For example, in their seminal paper, Halpern and Moses [33] use modal logic to model epistemic phenomena in distributed systems. A distributed system S gives rise to a *Kripke model* (see Section 4.1), whose set W of domain points corresponds to the set of partial runs of S, that is, finite sequences of global states of S. For each processor i of S, there is an *accessibility relation* R_i such that $(v,w) \in R_i$ if and only if v and w are *indistinguishable* from the point of view of processor i. This framework suits well for epistemic considerations.

In traditional modal approaches, the domain elements of a Kripke model correspond to possible states of a distributed computation process. Our perspective is a radical departure from this approach. In our framework, a distributed system is—essentially—a Kripke model, where the domain points are processors and the accessibility relations are communication channels. While such an interpretation is of course always possible, it turns out to be particularly helpful in the study of weak models of distributed computing. With this interpretation, for example, a local algorithm in $\mathsf{Set} \cap \mathsf{Broadcast}$ corresponds to a formula of modal logic, while a local algorithm in $\mathsf{Multiset} \cap \mathsf{Broadcast}$ corresponds to a formula of *graded* modal logic—local algorithms are exactly as expressive as such formulas, and the running time of an algorithm equals the modal depth of a formula. Standard techniques from the field of modal logic can be directly applied in the study of distributed algorithms, and conversely our classification of the weak models of distributed computing can be rephrased as a result that characterises the expressibility of modal logics in certain classes of Kripke models.

4 Connections with Modal Logic

In this section, we show how to characterise each of the classes $\mathsf{SB}(1)$, $\mathsf{MB}(1)$, $\mathsf{VB}(1)$, $\mathsf{SV}(1)$, $\mathsf{MV}(1)$, $\mathsf{VV}(1)$, and $\mathsf{VV}_\mathsf{c}(1)$ by a corresponding modal logic, in the spirit of descriptive complexity theory (see Immerman [35]). We show that for each class there is a modal logic that is equally expressive: for any graph problem in the class there is a formula in the modal logic that defines a solution of the graph problem; conversely, any formula in the modal logic defines a solution of some graph problem in the class.

4.1 Logics ML, GML, MML, and GMML

Our characterisation uses *basic modal logic* ML, *graded modal logic* GML, *multimodal logic* MML, and *graded multimodal logic* GMML—see, e.g., Blackburn, de Rijke, and Venema [9] or Blackburn, van Benthem, and Wolter [8] for further details on modal logic.

Basic modal logic, ML, is obtained by extending propositional logic by a single (unary) modal operator $\Diamond$. More precisely, if Φ is a finite set of proposition symbols, then the set of ML(Φ)-formulas is given by the following grammar:

$$\varphi := q \mid (\varphi \wedge \varphi) \mid \neg\varphi \mid \Diamond\varphi, \quad \text{where } q \in \Phi.$$

The semantics of ML is defined on Kripke models. A *Kripke model* for the set Φ of proposition symbols is a tuple

$K = (W, R, \tau)$, where W is a nonempty set of *states* (or *possible worlds*), $R \subseteq W^2$ is a binary relation on W (*accessibility relation*), and τ is a *valuation* function $\tau\colon \Phi \to \mathcal{P}(W)$.

The truth of an ML(Φ)-formula φ in a state $v \in W$ of a Kripke model $K = (W, R, \tau)$ is defined recursively as follows:

$$\begin{aligned}
&K, v \vDash q && \text{iff} \quad v \in \tau(q), \quad \text{for each } q \in \Phi,\\
&K, v \vDash (\varphi \wedge \vartheta) && \text{iff} \quad K, v \vDash \varphi \text{ and } K, v \vDash \vartheta,\\
&K, v \vDash \neg\varphi && \text{iff} \quad K, v \nvDash \varphi,\\
&K, v \vDash \Diamond\varphi && \text{iff} \quad K, w \vDash \varphi \text{ for some } w \in W\\
& && \phantom{\text{iff}} \quad \text{such that } (v, w) \in R.
\end{aligned}$$

Usually in modal logic one defines the abbreviations

$$(\varphi \vee \vartheta) := \neg(\neg\varphi \wedge \neg\vartheta) \text{ and } \Box\varphi := \neg\Diamond\neg\varphi.$$

Classical modal logic has its roots in the philosophical analysis of the notion of possibility. In classical modal logic, a modal formula $\Diamond\varphi$ is interpreted to mean that it is *possible* that φ holds. The set W of a Kripke model $K = (W, R, \tau)$ is a collection of possible worlds v, or possible states of affairs. The relation R connects a possible world v to exactly those worlds that can be considered to be—in one sense or another—*possible* states of affairs, when the *actual* state of affairs is in fact v. The semantics of the formula $\Diamond\varphi$ reflects this idea; $\Diamond\varphi$ is true in v if and only if there is a possible state of affairs w accessible from v via R such that φ is true in w.

Modern systems of modal logic often have very little to do with the original philosophical motivations of the field. The reason is that modal logic and Kripke semantics seem to adapt rather well to the requirements of a wide range of different kinds of applications in computer science and various other fields. Our use of modal logic in this article is an example of such an adaptation.

One of the features of basic modal logic is that it is unable to count: there is no mechanism in ML for separating states v of Kripke models based only on the number of R-successors of v. The most direct way to overcome this defect is to add counting to the modalities. The syntax of *graded modal logic* [24], GML, extends the syntax of ML with the rules $\Diamond_{\geq k}\varphi$, where $k \in \mathbb{N}$. The semantics of these graded modalities $\Diamond_{\geq k}$ is the following:

$$K, v \vDash \Diamond_{\geq k}\varphi \quad \text{iff} \quad \bigl|\{w \in W : (v, w) \in R \text{ and } K, w \vDash \varphi\}\bigr| \geq k.$$

Up to this point we have considered modal logics with only one modality $\Diamond$. *Multimodal logic*, MML, is the natural generalisation of ML that allows an arbitrary (finite) number of modalities. The modalities are usually written as $\langle\alpha\rangle$, where $\alpha \in I$ for some index set I. Given the set I and a finite set Φ of proposition symbols, the set of MML(I, Φ)-formulas is defined by the following grammar:

$$\varphi := q \mid (\varphi \wedge \varphi) \mid \neg\varphi \mid \langle\alpha\rangle\varphi, \quad \text{where } q \in \Phi \text{ and } \alpha \in I.$$

The Kripke models corresponding to the multimodal language MML(I, Φ) are of the form $K = (W, (R_\alpha)_{\alpha \in I}, \tau)$, where $R_\alpha \subseteq W^2$ for each $\alpha \in I$, and τ is a function $\tau\colon \Phi \to \mathcal{P}(W)$.

The truth definition of MML(I, Φ) is the same as the truth definition of ML for Boolean connectives and atomic formulas. For diamond formulas $\langle\alpha\rangle\varphi$ the semantics are given by the condition

$$\begin{aligned}
&K, v \vDash \langle\alpha\rangle\varphi && \text{iff} \quad K, w \vDash \varphi \text{ for some } w \in W\\
& && \phantom{\text{iff}} \quad \text{such that } (v, w) \in R_\alpha.
\end{aligned}$$

We can naturally extend MML by graded modalities $\langle\alpha\rangle_{\geq k}$ for each $\alpha \in I$ and $k \in \mathbb{N}$ and obtain *graded multimodal logic* GMML(I, Φ).

If the index set I contains only one element α, then MML(I, Φ) can be identified with ML(Φ) simply by replacing $\langle\alpha\rangle$ with $\Diamond$. Similarly, GMML($\{\alpha\}, \Phi$) is identified with GML(Φ).

Let $\mathcal{L}$ be a modal logic and φ an $\mathcal{L}(I, \Phi)$-formula. The *modal depth* of φ, denoted by $\mathrm{md}(\varphi)$, is defined recursively as follows:

$$\begin{aligned}
\mathrm{md}(q) &= 0 \text{ for } q \in \Phi,\\
\mathrm{md}(\varphi \wedge \vartheta) &= \max\{\mathrm{md}(\varphi), \mathrm{md}(\vartheta)\},\\
\mathrm{md}(\neg\varphi) &= \mathrm{md}(\varphi),\\
\mathrm{md}(\langle\alpha\rangle\varphi) &= \mathrm{md}(\varphi) + 1 \text{ for } \alpha \in I.
\end{aligned}$$

Thus, $\mathrm{md}(\varphi)$ is the largest number of nested modalities in φ.

Given a modal logic $\mathcal{L}$ and a Kripke model K for $\mathcal{L}$, each $\mathcal{L}$-formula φ *defines* a subset $\{v \in W \mid K, v \vDash \varphi\}$ of the set of states in K; this set is denoted by $\|\varphi\|^K$.

4.2 Bisimulation and Definability in Modal Logic

We will now define one of the most important concepts in modal logic, bisimulation. Bisimulation was first defined in the context of modal logic by van Benthem [7], who calls it a *p-relation*. Bisimulation was also discovered independently in a variety of other fields. See Sangiorgi [53] for the history and development of the notion.

The objective of bisimulation is to characterise definability in the corresponding modal logics, so that if two states w and w' are bisimilar they cannot be separated by any formula of the corresponding logic. Bisimulation can be defined in a canonical way for each of the logics ML, GML, MML, and GMML.

Bisimulation for MML is defined as follows. Let

$$\begin{aligned}
K &= \bigl(W, (R_\alpha)_{\alpha \in I}, \tau\bigr),\\
K' &= \bigl(W', (R'_\alpha)_{\alpha \in I}, \tau'\bigr)
\end{aligned}$$

be Kripke models for a set Φ of proposition symbols. A nonempty relation $Z \subseteq W \times W'$ is a *bisimulation* between K and K' if the following conditions hold.

(B1) If $(v,v') \in Z$, then $v \in \tau(q)$ iff $v' \in \tau'(q)$ for all $q \in \Phi$.
(B2) If $(v,v') \in Z$ and $(v,w) \in R_\alpha$ for some $\alpha \in I$, then there is a $w' \in W'$ with $(v',w') \in R'_\alpha$ and $(w,w') \in Z$.
(B3) If $(v,v') \in Z$ and $(v',w') \in R'_\alpha$ for some $\alpha \in I$, then there is a $w \in W$ with $(v,w) \in R_\alpha$ and $(w,w') \in Z$.

If there is a bisimulation Z such that $(v,v') \in Z$, we say that v and v' are *bisimilar*.

For the basic modal logic ML, bisimulation is defined in the same way just by replacing the relations R_α, $\alpha \in I$, in conditions (B2) and (B3) with the single relation R.

In the case of the graded modal logic GML, we use the notion of *graded bisimulation*: a nonempty relation $Z \subseteq W \times W'$ is a graded bisimulation between $K = (W,R,\tau)$ and $K' = (W',R',\tau')$ if it satisfies condition (B1) and the following modifications of (B2) and (B3); we use the notation $R(v) = \{w \in W : (v,w) \in R\}$.

(B2*) If $(v,v') \in Z$ and $X \subseteq R(v)$, then there is a set $X' \subseteq R'(v')$ such that $|X'| = |X|$ and for each $w' \in X'$ there is a $w \in X$ with $(w,w') \in Z$.
(B3*) If $(v,v') \in Z$ and $X' \subseteq R'(v')$, then there is a set $X \subseteq R(v)$ such that $|X| = |X'|$ and for each $w \in X$ there is a $w' \in X'$ with $(w,w') \in Z$.

We say that v and v' are *g-bisimilar* if there is a graded bisimulation Z such that $(v,v') \in Z$.

The definition of graded bisimulation for GMML is the obvious generalisation of the definition above to the case of several relations R_α instead of a single relation R.

The notion of graded bisimulation was first formulated by de Rijke [52]. Our definition follows the formulation of Conradie [17]. We state next the main result concerning bisimulation. For the proof of Fact 1a, we refer to Blackburn et al. [9]. The proof of Fact 1b can be found in Conradie [17].

Fact 1 (a) *Let $\mathcal{L}$ be* ML *or* MML*, and let K and K' be Kripke models, $v \in W$ and $v' \in W'$. If v and v' are bisimilar, then for all $\mathcal{L}$-formulas φ*

$$K,v \models \varphi \textit{ iff } K',v' \models \varphi.$$

(b) *Let $\mathcal{L}$ be* GML *or* GMML*, and let K and K' be Kripke models, $v \in W$ and $v' \in W'$. If v and v' are g-bisimilar, then for all $\mathcal{L}$-formulas φ*

$$K,v \models \varphi \textit{ iff } K',v' \models \varphi.$$

In what follows, we will develop a connection between modal logic and weak models of distributed computing. Informally, the states of a Kripke model will correspond to the nodes of a distributed system, and bisimilar states will correspond to nodes that are unable to distinguish their neighbourhoods, no matter which distributed algorithm we use. With the help of this connection, we can then use bisimulation in Section 5.3 to prove separations of problem classes.

Modal logic	Distributed algorithms
Kripke model $K = (W,(R_\alpha)_{\alpha\in I},\tau)$	input graph $G = (V,E)$ port numbering p
states W	nodes V
relations R_α, $\alpha \in I$	edges E and port numbering p
valuation τ proposition symbols $q_1, q_2, \ldots$	node degrees (initial state)
formula φ	algorithm $\mathcal{A}$
formula φ is true in state v	$\mathcal{A}$ outputs 1 in node v
modal depth of φ	running time of $\mathcal{A}$

Table 3 Correspondence between modal logic and distributed algorithms.

4.3 Characterising Constant-Time Classes by Modal Logics

There is a natural correspondence between the framework for distributed computing defined in this paper and the logics ML, GML, MML, and GMML. For any input graph G and port numbering p of G, the pair (G,p) can be transformed into a Kripke model $K(G,p) = (W,(R_\alpha)_{\alpha\in I},\tau)$ in a canonical way. Given a local algorithm $\mathcal{A}$, its execution can then be simulated by a modal formula φ. The crucial idea is that the truth condition for a diamond formula $\langle\alpha\rangle\psi$ is interpreted as communication between the nodes:

$$K,v \models \langle\alpha\rangle\psi \quad \text{iff} \quad \begin{array}{l} v \text{ receives the message "}\psi\text{ is true"} \\ \text{from some } u \text{ such that } (v,u) \in R_\alpha. \end{array}$$

Conversely, for any modal formula φ, there is a local algorithm $\mathcal{A}$ that can evaluate the truth of φ in the Kripke model $K(G,p)$.

The general idea of the correspondence between modal logic and distributed algorithms is described in Table 3. We will assume that $\mathcal{A}$ produces a one-bit output, i.e., $Y = \{0,1\}$; other cases can be handled by defining a separate formula for each output bit.

We start by defining the Kripke models $K(G,p)$. There are in fact four different versions of $K(G,p)$, reflecting the fact that algorithms in the lower classes do not use all the information encoded in the port numbering. Let $G = (V,E) \in \mathcal{F}(\Delta)$, and let p be a port numbering of G. The accessibility relations used in the different versions of $K(G,p)$ are the following; see Fig. 7 for illustrations:

$$R_{(i,j)} = \{(u,v) \in V \times V : p((v,j)) = (u,i)\} \quad \text{for each pair } (i,j) \in [\Delta] \times [\Delta].$$

Given Δ, these relations together with the vertex set V contain the same information as the pair (G,p): graph G and port numbering p can be reconstructed from the pair

$$\big(V,(R_{(i,j)})_{(i,j)\in[\Delta]\times[\Delta]}\big).$$

Since algorithms in classes below Vector have access to a restricted part of the information in p, we need alternative accessibility relations with corresponding restrictions on their information about p:

$$R_{(i,*)} = \bigcup_{j\in[\Delta]} R_{(i,j)} \qquad \text{for each } i \in [\Delta],$$

$$R_{(*,j)} = \bigcup_{i\in[\Delta]} R_{(i,j)} \qquad \text{for each } j \in [\Delta],$$

$$R_{(*,*)} = \bigcup_{(i,j)\in[\Delta]\times[\Delta]} R_{(i,j)}.$$

Note that $R_{(*,*)} = \{(u,v) : \{u,v\} \in E\}$ is the edge set E interpreted as a symmetric relation.

In addition to the accessibility relations, we encode the local information on the degrees of vertices into a valuation $\tau\colon \Phi_\Delta \to \mathcal{P}(V)$, where $\Phi_\Delta = \{q_i : i \in [\Delta]\}$. The valuation τ is given by

$$\tau(q_i) = \{v \in V : \deg(v) = i\}.$$

The four versions of a Kripke model corresponding to graph G and port numbering p are now defined as follows:

$$K_{+,+}(G,p) = (V, (R_\alpha)_{\alpha\in I^\Delta_{+,+}}, \tau), \quad \text{where } I^\Delta_{+,+} = [\Delta]\times[\Delta],$$
$$K_{-,+}(G,p) = (V, (R_\alpha)_{\alpha\in I^\Delta_{-,+}}, \tau), \quad \text{where } I^\Delta_{-,+} = \{*\}\times[\Delta],$$
$$K_{+,-}(G,p) = (V, (R_\alpha)_{\alpha\in I^\Delta_{+,-}}, \tau), \quad \text{where } I^\Delta_{+,-} = [\Delta]\times\{*\},$$
$$K_{-,-}(G,p) = (V, (R_\alpha)_{\alpha\in I^\Delta_{-,-}}, \tau), \quad \text{where } I^\Delta_{-,-} = \{(*,*)\}.$$

For all $a,b \in \{-,+\}$, we denote the class of all Kripke models of the form $K_{a,b}(G,p)$ by $\mathcal{K}_{a,b}$. Furthermore, we denote by $\mathcal{K}^{\mathrm{c}}_{+,+}$ the subclass of $\mathcal{K}_{+,+}$ consisting of the models $K_{+,+}(G,p)$, where p is a consistent port numbering of G.

In order to give a precise formulation to the correspondence between modal logics and the constant-time classes of graph problems, we define the concept of modal formulas solving graph problems. Without loss of generality, we consider here only problems Π such that the solutions $S \in \Pi(G)$ are functions $V \to \{0,1\}$, or equivalently, subsets of V. This is a natural restriction, since a modal formula ψ defines a subset

$$\|\psi\|^{K_{a,b}(G,p)} := \{v \in V \mid K_{a,b}(G,p), v \models \psi\}$$

of the vertex set V. Other cases can be handled by using tuples of formulas.

Let $a,b \in \{-,+\}$, and let $\Psi = (\psi_1, \psi_2, \ldots)$ be a sequence of modal formulas such that ψ_Δ is in the signature $(I^\Delta_{a,b}, \Phi_\Delta)$ for each $\Delta \in \mathbb{N}$. Then Ψ *defines a solution* for a graph problem Π on the class $\mathcal{K}_{a,b}$ if the following condition holds:

- For all $\Delta \in \mathbb{N}$, all $G \in \mathcal{F}(\Delta)$, and all port numberings p of G, the subset $\|\psi_\Delta\|^{K_{a,b}(G,p)}$ defined by the formula ψ_Δ in the model $K_{a,b}(G,p)$ is in the set $\Pi(G)$.

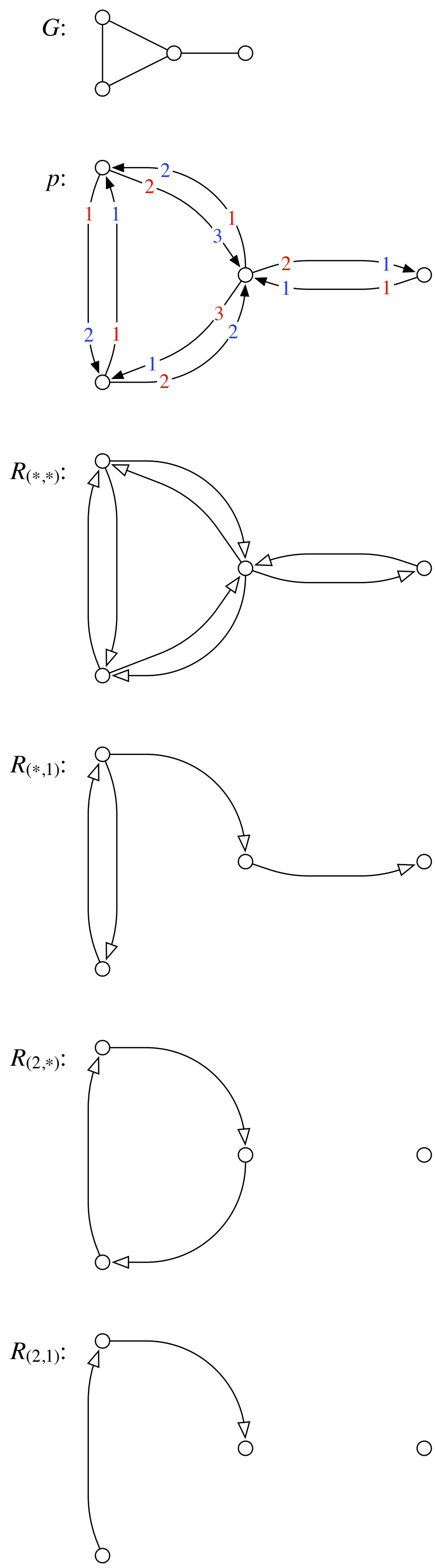


Fig. 7 Relations $R_{(i,j)}$—note the directions of the arrows.

Furthermore, the sequence Ψ defines a solution for Π on the class $\mathcal{K}^{\mathsf{c}}_{+,+}$, if the condition above with $a = b = +$ holds for all *consistent* port numberings p.

Note that any sequence $\Psi = (\psi_1, \psi_2, \dots)$ of modal formulas as above gives rise to a canonical graph problem Π_Ψ that it defines a solution for: for each graph G, the solution set $\Pi_\Psi(G)$ simply consists of the sets $\|\psi_\Delta\|^{K_{a,b}(G,p)}$ where $G \in \mathcal{F}(\Delta)$ and p ranges over the (consistent) port numberings of G.

Let $\mathcal{L}$ be a modal logic, let $a, b \in \{-,+\}$, and let C be a class of graph problems. We say that $\mathcal{L}$ *is contained in* C on $\mathcal{K}_{a,b}$, in symbols $\mathcal{L} \le C$ on $\mathcal{K}_{a,b}$, if the following condition holds:

– If $\Psi = (\psi_1, \psi_2, \dots)$ is a sequence of formulas such that $\psi_\Delta \in \mathcal{L}(I^\Delta_{a,b}, \Phi_\Delta)$ for all $\Delta \in \mathbb{N}$, then $\Pi_\Psi \in C$.

Furthermore, we say that $\mathcal{L}$ *simulates* C on $\mathcal{K}_{a,b}$, in symbols $C \le \mathcal{L}$ on $\mathcal{K}_{a,b}$, if the following condition holds:

– For every graph problem $\Pi \in C$ there is a sequence $\Psi = (\psi_1, \psi_2, \dots)$ of formulas such that $\psi_\Delta \in \mathcal{L}(I^\Delta_{a,b}, \Phi_\Delta)$ for all $\Delta \in \mathbb{N}$, which defines a solution for Π on $\mathcal{K}_{a,b}$.

Finally, we say that $\mathcal{L}$ *captures* C on $\mathcal{K}_{a,b}$ if both $\mathcal{L} \le C$ and $C \le \mathcal{L}$ on $\mathcal{K}_{a,b}$.

The notions of $\mathcal{L}$ being contained in C on $\mathcal{K}^{\mathsf{c}}_{+,+}$, $\mathcal{L}$ simulating C on $\mathcal{K}^{\mathsf{c}}_{+,+}$, and $\mathcal{L}$ capturing C on $\mathcal{K}^{\mathsf{c}}_{+,+}$ are defined similarly with the obvious restriction to consistent port numberings.

The main result of this section is that the constant-time version of each of the classes $\mathsf{VV_c}$, VV, MV, SV, VB, MB, and SB is captured by one of the modal logics MML, ML, GMML, and GML on an appropriate class $\mathcal{K}_{a,b}$.

Theorem 1 (a) MML *captures* $\mathsf{VV_c}(1)$ *on* $\mathcal{K}^{\mathsf{c}}_{+,+}$.
(b) MML *captures* $\mathsf{VV}(1)$ *on* $\mathcal{K}_{+,+}$.
(c) GMML *captures* $\mathsf{MV}(1)$ *on* $\mathcal{K}_{-,+}$.
(d) MML *captures* $\mathsf{SV}(1)$ *on* $\mathcal{K}_{-,+}$.
(e) MML *captures* $\mathsf{VB}(1)$ *on* $\mathcal{K}_{+,-}$.
(f) GML *captures* $\mathsf{MB}(1)$ *on* $\mathcal{K}_{-,-}$.
(g) ML *captures* $\mathsf{SB}(1)$ *on* $\mathcal{K}_{-,-}$.

Proof of Theorem 1: Overview. Note first that (a) follows directly from (b) by restricting to consistent port numberings. Furthermore, the only difference between GMML and MML is the ability to count the number of neighbours satisfying a formula, which corresponds in a natural way to the difference between algorithms in Multiset and Set. Hence, we omit the proof of (d), as it is obtained from the proof of (c) by minor modifications. Similarly, the proof of (g) is a minor modification of the proof of (f), so we omit it, too.

Thus, we are left with the task of proving claims (b), (c), (e) and (f). The structure of the proofs of all these claims is the same—there are differences only in technical details. Hence, we divide the proof in four parts as follows.

1. We prove the first half of (b): MML $\le \mathsf{VV}(1)$ on $\mathcal{K}_{+,+}$.
2. We describe the changes in part 1 needed for proving the first halves of (c), (e) and (f).
3. We prove the second half of (b): $\mathsf{VV}(1) \le$ MML on $\mathcal{K}_{+,+}$.
4. We describe the changes in part 3 needed for proving the second halves of (c), (e) and (f).

Proof of Theorem 1, Part 1. Assume that $\Psi = (\psi_1, \psi_2, \dots)$ is a sequence of formulas with $\psi_\Delta \in \mathrm{MML}(I^\Delta_{+,+}, \Phi_\Delta)$ for each $\Delta \in \mathbb{N}$. We give for each $\Delta \in \mathbb{N}$ a local algorithm $\mathcal{A}_\Delta \in$ Vector that simulates the recursive evaluation of the truth of ψ_Δ on a Kripke model $K_{+,+}(G, p)$.

Let Σ be the set of all subformulas of ψ_Δ, and let D_j, $j \in [\Delta]$, be the subset of Σ consisting of subformulas η such that $\langle\alpha\rangle\eta \in \Sigma$, where $\alpha = (i, j)$ for some $i \in [\Delta]$. The set of stopping states, intermediate states, and messages of the algorithm $\mathcal{A}_\Delta$ (see Section 1.1) are defined as follows:

$$
\begin{aligned}
Y &:= \{0,1\},\\
Z &:= \big\{f : f \text{ is a function } \Sigma \to \{0,1,U\}\big\},\\
M &:= \bigcup_{j\in[\Delta]} \big\{h : h \text{ is a function } D_j \to \{0,1,U\} \times \{j\}\big\}\\
&\quad\ \cup \{m_0\}.
\end{aligned}
$$

The idea behind these choices is that before stopping, the state $x_t(v)$ of the computation of $\mathcal{A}_\Delta$ on a node v of an input (G, p) encodes the truth value of each subformula of ψ_Δ with modal depth at most t; for subformulas with modal depth greater than t, the state $x_t(v)$ gives the value U (undefined). In other words, our aim is to make sure that at each step t of the computation, $x_t(v) = f$, where $f \in Z$ is the function defined by

$$
f(\eta) = \begin{cases} 0, \text{ if } \mathrm{md}(\eta) \le t \text{ and } K_{+,+}(G,p), v \not\models \eta \\ 1, \text{ if } \mathrm{md}(\eta) \le t \text{ and } K_{+,+}(G,p), v \models \eta \\ U, \text{ if } \mathrm{md}(\eta) > t \end{cases}
$$

for each $\eta \in \Sigma$. First, we define the function $z_0 \colon [\Delta] \to Y \cup Z$ that gives the initial state $x_0(v) = z_0(\deg(v))$ of each node v. For each $i \in [\Delta]$, we set $z_0(i) = g$, where g is the function defined recursively as follows:

$$g(\eta) = 1 \quad \text{for each } \eta = q_i \in \Sigma,$$

$$g(\eta) = 0 \quad \text{for each } \eta = q_j \in \Sigma, j \in [\Delta] \setminus \{i\},$$

$$
g(\eta) = \begin{cases} 0, \text{ if } 0 \in \{g(\vartheta), g(\gamma)\} \subseteq \{0,1\} \\ 1, \text{ if } \{g(\vartheta), g(\gamma)\} = \{1\} \\ U, \text{ if } U \in \{g(\vartheta), g(\gamma)\} \end{cases}
$$

for each $\eta = (\vartheta \wedge \gamma) \in \Sigma$,

$$g(\eta) = \begin{cases} 0, \text{ if } g(\vartheta) = 1 \\ 1, \text{ if } g(\vartheta) = 0 \\ U, \text{ if } g(\vartheta) = U \end{cases}$$
for each $\eta = \neg\vartheta \in \Sigma$,

$g(\eta) = U$ for each $\eta = \langle\alpha\rangle\vartheta \in \Sigma$.

If a node v of the input graph G is in the state $x_t(v) = f \in Z$ at time step t, then the message $\mu(f, j)$ it sends to its port $j \in [\deg(v)]$ at step $t+1$ is obtained from the restriction of f to the set D_j by adding j as a marker: that is, $\mu(f, j)$ is the function $h\colon D_j \to \{0,1,U\} \times \{j\}$ such that $h(\eta) = (f(\eta), j)$ for all $\eta \in D_j$.

Finally, the state transition function δ of $\mathcal{A}_\Delta$ is described as follows. Assume that the state of a node v at time t is $x_t(v) = f \in Z$, and the vector of messages it receives at time $t+1$ from the ports is $\mathbf{a}_{t+1}(v) = (h_1, \ldots, h_\Delta)$. If $f(\psi_\Delta) = U$, then $x_{t+1}(v)$ is the function $g \in Z$ defined as follows:

1. For each $\eta \in \Sigma$ with $f(\eta) \neq U$, we set $g(\eta) = f(\eta)$.
2. For each $\eta \in \Sigma$ with $f(\eta) = U$, we define $g(\eta)$ by the following recursion:

$$g(\eta) = \begin{cases} 0, \text{ if } 0 \in \{g(\vartheta), g(\gamma)\} \subseteq \{0,1\} \\ 1, \text{ if } \{g(\vartheta), g(\gamma)\} = \{1\} \\ U, \text{ if } U \in \{g(\vartheta), g(\gamma)\} \end{cases} \qquad (\delta_\wedge)$$
for each $\eta = (\vartheta \wedge \gamma) \in \Sigma$,

$$g(\eta) = \begin{cases} 0, \text{ if } g(\vartheta) = 1 \\ 1, \text{ if } g(\vartheta) = 0 \\ U, \text{ if } g(\vartheta) = U \end{cases} \qquad (\delta_\neg)$$
for each $\eta = \neg\vartheta \in \Sigma$,

$$g(\eta) = \begin{cases} 0, \text{ if } f(\vartheta) \neq U \text{ and } h_i(\vartheta) \neq (1, j) \\ 1, \text{ if } f(\vartheta) \neq U \text{ and } h_i(\vartheta) = (1, j) \\ U, \text{ if } f(\vartheta) = U \end{cases} \qquad (\delta_\Diamond)$$
for each $\eta = \langle(i, j)\rangle\vartheta \in \Sigma$.

For convenience, we interpret m_0 as a function with $m_0(\vartheta) = (0,1)$ for each subformula ϑ.

If $f(\psi_\Delta) \neq U$, we let $x_{t+1}(v) = f(\psi_\Delta) \in Y$.

It is now straightforward to prove by induction on modal depth that the following holds for any input graph $G \in \mathcal{F}(\Delta)$, port numbering p of G, and node v of G:

– If $\eta \in \Sigma$, $\mathrm{md}(\eta) \leq t \leq \mathrm{md}(\psi_\Delta)$, and $x_t(v) = f \in Z$, then $f(\eta) \in \{0,1\}$ and $f(\eta) = 1$ iff $K_{+,+}(G, p), v \vDash \eta$.

Thus, if $t = \mathrm{md}(\psi_\Delta)$ and $x_t(v) = f$, then $f(\psi_\Delta)$ reveals the truth value of ψ_Δ on v. This means that the computation of $\mathcal{A}_\Delta$ stops at step $t+1$, and the output $x_{t+1}(v)$ on node v is 1 if and only if $K_{+,+}(G, p), v \vDash \psi_\Delta$. In other words, the running time of $\mathcal{A}_\Delta$ is the constant $\mathrm{md}(\psi_\Delta) + 1$, and its output on the input (G, p) is the set $\|\psi_\Delta\|^{K_{+,+}(G,p)}$. Hence, the sequence $\mathbf{A} = (\mathcal{A}_1, \mathcal{A}_2, \ldots)$ of algorithms solves the graph problem Π_Ψ, and we conclude that $\Pi_\Psi \in \mathsf{VV}(1)$.

Proof of Theorem 1, Part 2. We will now consider the proofs of the first halves of (c), (e) and (f). In each of these cases, we are given a formula ψ_Δ in the corresponding modal logic, and we define an algorithm $\mathcal{A}_\Delta$ which simulates the recursive truth definition of ψ_Δ. The definitions of the state sets Y and Z, as well as the definition of the initial state function z_0 remain unchanged in all cases.

However, since the modal operators occurring in subformulas of Ψ_Δ are different in each of the cases, the sets D_j, $j \in [\Delta]$ have to be redefined accordingly. Moreover, in cases (e) and (f), we have to remove the markers j from the messages, since the algorithm $\mathcal{A}_\Delta$ should be in the class $\mathsf{Broadcast}$. Thus, the message set M and the message constructing function μ have to be redefined for the proof of (e) and (f). Finally, in all cases, the clause $(\delta_\Diamond)$ in the recursive definition of the next state $x_{t+1}(v)$ has to be modified according to the semantics of the corresponding modal operators. Below, we list these modifications for each case separately.

(c) For each $j \in [\Delta]$, set D_j is redefined as

$D_j := \{\eta : \langle(*, j)\rangle_{\geq k}\eta \in \Sigma \text{ for some } k \in \mathbb{N}\}$.

Clause $(\delta_\Diamond)$ is replaced with

$$g(\eta) = \begin{cases} 0, \text{ if } f(\vartheta) \neq U \text{ and } |H| < k \\ 1, \text{ if } f(\vartheta) \neq U \text{ and } |H| \geq k \\ U, \text{ if } f(\vartheta) = U \end{cases} \qquad (\delta'_\Diamond)$$

for each $\eta = \langle(*, j)\rangle_{\geq k}\vartheta \in \Sigma$. Here

$H = \{i \in [\Delta] : h_i(\vartheta) = (1, j)\}$.

(e) The definition of $(D_j)_{j \in [\Delta]}$ is replaced with

$D := \{\eta : \langle(i, *)\rangle\eta \in \Sigma\}$.

Set M is redefined as

$M := \{h : h \text{ is a function } D \to \{0,1,U\}\} \cup \{m_0\}$.

Function $\mu(f, j)$ is redefined to be the restriction of f to the set D. Clause $(\delta_\Diamond)$ is replaced with

$$g(\eta) = \begin{cases} 0, \text{ if } f(\vartheta) \neq U \text{ and } h_i(\vartheta) = 0 \\ 1, \text{ if } f(\vartheta) \neq U \text{ and } h_i(\vartheta) = 1 \\ U, \text{ if } f(\vartheta) = U \end{cases} \qquad (\delta''_\Diamond)$$

for each $\eta = \langle(i, *)\rangle\vartheta \in \Sigma$. Here we interpret m_0 as a function with $m_0(\vartheta) = 0$ for all ϑ.

(f) The definition of $(D_j)_{j\in[\Delta]}$ is replaced with

$$D' := \{\eta : \langle(*,*)\rangle_{\geq k}\eta \in \Sigma \text{ for some } k\in\mathbb{N}\}.$$

Set M is redefined as

$$M := \{h : h \text{ is a function } D' \to \{0,1,U\}\}\cup\{m_0\}.$$

Function $\mu(f,j)$ is redefined to be the restriction of f to the set D'. Clause $(\delta_\diamond)$ is replaced with

$$g(\eta) = \begin{cases} 0, \text{ if } f(\vartheta)\neq U \text{ and } |H'|<k \\ 1, \text{ if } f(\vartheta)\neq U \text{ and } |H'|\geq k \\ U, \text{ if } f(\vartheta)=U \end{cases} \qquad (\delta_\diamond''')$$

for each $\eta = \langle(*,*)\rangle_{\geq k}\vartheta \in \Sigma$. Here

$$H' = \{i\in[\Delta] : h_i(\vartheta)=1\}.$$

Again, we interpret m_0 as a function with $m_0(\vartheta)=0$ for all ϑ.

It is now straightforward to check that in all the cases $\mathcal{A}_\Delta$ computes the truth value of ψ_Δ correctly in $\mathrm{md}(\psi_\Delta)+1$ steps, whence $\mathbf{A}=(\mathcal{A}_1,\mathcal{A}_2,\dots)$ solves Π_Ψ in constant time. Furthermore, it is easy to see that in case (c), $\mathcal{A}_\Delta$ is in the class Multiset, whence Π_ψ is in $\mathsf{MV}(1)$. Similarly, in case (e), $\mathcal{A}_\Delta$ is in Broadcast, and in case (f) $\mathcal{A}_\Delta$ is in Multiset $\cap$ Broadcast, as desired.

Proof of Theorem 1, Part 3. Assume now that Π is a graph problem in $\mathsf{VV}(1)$. Thus, there is a sequence

$$\mathbf{A}=(\mathcal{A}_1,\mathcal{A}_2,\dots)$$

of local algorithms in **Vector** such that for every $G\in\mathcal{F}(\Delta)$ and port numbering p of G, the output of $\mathcal{A}_\Delta$ on (G,p) is in $\Pi(G)$. We will encode information on the states of computation and messages sent during the execution of $\mathcal{A}_\Delta$ on an input (G,p) by suitable formulas of MML.

Using the definitions of Section 1.1, let

$$\mathcal{A}_\Delta = (Y,Z,z_0,M,m_0,\mu,\delta),$$

and let T be the running time of $\mathcal{A}_\Delta$. We will build a formula $\psi_\Delta\in\mathrm{MML}(I^\Delta_{+,+},\Phi_\Delta)$ simulating $\mathcal{A}_\Delta$ from the following subformulas:

- $\varphi_{z,t}$ for $z\in Y\cup Z$ and $t\in[T]$,
- $\vartheta_{m,j,t}$ for $m\in M$, $j\in[\Delta]$ and $t\in[T]$,
- $\chi_{m,i,j,t}$ for $m\in M$, $i,j\in[\Delta]$ and $t\in[T]$.

The intended meaning of these subformulas are given in Table 4, and their recursive definitions are indicated in Table 5.

Note that the set Z of intermediate states, as well as the set M of messages, may be infinite, whence there are potentially infinitely many formulas of the form $\varphi_{z,t}$, $\vartheta_{m,j,t}$ and $\chi_{m,i,j,t}$. However, it is easy to prove by induction on t that there are only finitely many *different* formulas in the families

$$\Psi_t = \{\varphi_{z,t} : z\in Y\cup Z\},$$
$$\Theta_t = \{\vartheta_{m,j,t} : m\in M \text{ and } j\in[\Delta]\},$$
$$\Xi_t = \{\chi_{m,i,j,t} : m\in M \text{ and } i,j\in[\Delta]\}.$$

Indeed, for each $z\in Y\cup Z$, subformula $\varphi_{z,0}$ is a disjunction of the form $\bigvee_{i\in J} q_i$ for some $J\subseteq[\Delta]$; here $\bigvee_{i\in\emptyset} q_i$ is understood as some fixed contradictory formula. Furthermore, assuming Ψ_t is finite, there are only finitely many different Boolean combinations of formulas in Ψ_t, whence Θ_{t+1} is finite. By the same argument, if Θ_{t+1} is finite, then so is Ξ_{t+1}, and if Ψ_t and Ξ_{t+1} are finite, then so is Ψ_{t+1}.

Clearly the formulas $\varphi_{z,t}$, $\vartheta_{m,j,t}$ and $\chi_{m,i,j,t}$ can be defined in such a way that each of them has its intended meaning. In particular, given an input (G,p) to the algorithm $\mathcal{A}_\Delta$, the output on a node v is 1 if and only if $v\in\|\varphi_{1,T}\|^{K_{+,+}(G,p)}$. Thus, defining $\psi_\Delta := \varphi_{1,T}$ we get $\|\psi_\Delta\|^{K_{+,+}(G,p)}\in\Pi(G)$ for all $G\in\mathcal{F}(\Delta)$ and all port numberings p of G. Hence we conclude that the sequence $\Psi=(\psi_1,\psi_2,\dots)$ defines a solution to Π.

As an additional remark, we note that the modal depth of each $\varphi_{z,t}$ is t, as an easy induction shows. In particular, $\mathrm{md}(\psi_\Delta)$ is equal to the running time T of $\mathcal{A}_\Delta$.

Proof of Theorem 1, Part 4. To complete the proof, we will now describe the changes needed in the technical details for proving the second halves of claims (c), (e) and (f). Thus, assume that Π is a graph problem, and $\mathbf{A}=(\mathcal{A}_1,\mathcal{A}_2,\dots)$ is a local algorithm which solves Π and is in the class **Multiset**, **Broadcast** or **Multiset** $\cap$ **Broadcast**, respectively. The corresponding modal formula ψ_Δ is constructed from subformulas in the same way as in (3) with appropriate modifications in technical details.

Since algorithms in Multiset cannot distinguish between the port numbers of incoming messages, the subscript i in the formulas $\chi_{m,i,j,t}$ has to be removed in cases of (c) and (f). On the other hand, the algorithms can count the multiplicities of incoming messages, whence a new parameter $k\in[\Delta]$ for these formulas is needed. Furthermore, in cases (e) and (f), the subscript j has to be removed from the formulas $\vartheta_{m,j,t}$ and $\chi_{m,i,j,t}$, as algorithms in the class Broadcast cannot send different messages through different ports. Below, we summarise the modifications in each case separately.

(c) The formulas $\chi_{m,i,j,t}$ are replaced with $\chi^k_{m,j,t}$, $k\in[\Delta]$. The recursive definition of these formulas is as follows:

$$\chi^k_{m,j,t+1} := \langle(*,j)\rangle_{\geq k}\vartheta_{m,j,t+1}.$$

The formulas $\vartheta_{m,j,t+1}$ and $\varphi_{z,t+1}$ are defined as in Table 5, with $\chi^k_{m,j,t}$ in place of $\chi_{m,i,j,t}$.

Subformulas of ψ_Δ	Algorithm $\mathcal{A}_\Delta$
$\varphi_{z,t}$ is true in world v	local state $x_t(v)$ is z
$\vartheta_{m,j,t}$ is true in world v	node v sends message m to port j in round t
$\chi_{m,i,j,t}$ is true in world v	node v receives message m from port i in round t; the message was sent by an adjacent node to port j

Table 4 The intended meaning of the subformulas.

Recursive definition of the formulas		Execution of $\mathcal{A}_\Delta$	
$\varphi_{z,0}$:	Boolean combination of $q_i \in \Phi_\Delta$	initialisation:	$x_0(u) = z_0(\deg(u))$
$\vartheta_{m,j,t+1}$:	Boolean combination of $\varphi_{z,t}$, $z \in Y \cup Z$	local computation:	$m = \mu(x_t(v), j)$
$\chi_{m,i,j,t+1} := \langle\alpha\rangle\vartheta_{m,j,t+1}$ with $\alpha = (i,j)$		communication:	construct $\mathbf{a}_{t+1}(v)$
$\varphi_{z,t+1}$:	Boolean combination of $\varphi_{x,t}$, $x \in Y \cup Z$, and $\chi_{m,i,j,t+1}$, $m \in M$, $i,j \in [\Delta]$	local computation:	$x_{t+1}(v) = \delta(x_t(v), \mathbf{a}_{t+1}(v))$

Table 5 Constructing the formula ψ_Δ, given an algorithm $\mathcal{A}_\Delta$.

(e) The formulas $\vartheta_{m,j,t}$ and $\chi_{m,i,j,t}$ are replaced with $\vartheta_{m,t}$, and $\chi_{m,i,t}$, respectively. The recursive definition of the latter is as follows:

$$\chi_{m,i,t+1} := \langle(i,*)\rangle\vartheta_{m,t+1}.$$

The formulas $\vartheta_{m,t+1}$ are defined as Boolean combinations of $\varphi_{z,t}$, and the formulas $\varphi_{z,t+1}$ are defined as in Table 5.

(f) The formulas $\vartheta_{m,j,t}$ and $\chi_{m,i,j,t}$ are replaced with $\vartheta_{m,t}$, and $\chi^k_{m,t}$, respectively. The recursive definition of the latter is as follows:

$$\chi^k_{m,t+1} := \langle(*,*)\rangle_{\geq k}\vartheta_{m,t+1}.$$

The formulas $\vartheta_{m,t+1}$ are defined as Boolean combinations of $\varphi_{z,t}$, and the formulas $\varphi_{z,t+1}$ are defined as in Table 5.

As in the proof of claim (b), it is easy to see that in each case the subformulas used in the construction of $\psi_\Delta := \varphi_{1,T}$ can be defined in such a way that they have their intended meanings. Thus, for every graph $G \in \mathcal{F}(\Delta)$ and every port numbering p of G, the output of $\mathcal{A}_\Delta$ in (G,p) equals

$$\|\psi_\Delta\|^{K_{a,b}(G,p)},$$

where $a,b \in \{-,+\}$ is selected appropriately for each case. Hence, we conclude that the sequence $\Psi = (\psi_1, \psi_2, \dots)$ defines a solution to Π.

This concludes the proof of Theorem 1. □

There is a slight asymmetry in the proof of Theorem 1: in the first half of the proof the running time of the constructed algorithm $\mathcal{A}_\Delta$ is $\mathrm{md}(\psi_\Delta)+1$, while in the second half the modal depth of the formula ψ_Δ constructed is exactly the running time of the given algorithm $\mathcal{A}_\Delta$. However, this mismatch can be rectified by modifying the proof of the first part. We did not write this modified proof simply to avoid unnecessary technicalities.

Theorem 1 gives us a tool for proving that a given graph problem Π is not in one of the classes considered in this paper. The idea is to use bisimulation for showing that the corresponding modal logic cannot define a solution for Π. At first it may appear that this tool can be applied only for the constant-time versions of the classes, as the logical characterisations in Theorem 1 are valid only in the constant-time case. However, in the following corollary we show that the method based on bisimulation can be used also in the general case. In principle, this result is valid for all seven classes, but we formulate it here only for VV, VB and SB; these are the cases we use later in Section 5.3. We remind the reader that throughout this section we focus on the case of binary outputs, i.e., $Y = \{0,1\}$, in which case we can interpret a solution $S \in \Pi$ as a subset $S \subseteq V$.

Corollary 1 *Let $G = (V,E) \in \mathcal{F}(\Delta)$ be a graph, $X \subseteq V$, and let Π be a graph problem such that for every $S \in \Pi(G)$, there are $u,v \in X$ with $u \in S$ and $v \notin S$.*

(a) *If there is a port numbering p of G such that all nodes in X are bisimilar in the model $K_{+,+}(G,p)$, then Π is not in the class* VV.
(b) *If there is a port numbering p of G such that all nodes in X are bisimilar in the model $K_{+,-}(G,p)$, then Π is not in the class* VB.
(c) *If there is a port numbering p of G such that all nodes in X are bisimilar in the model $K_{-,-}(G,p)$, then Π is not in the class* SB.

Proof We prove only claim (a); the other claims can be proved in the same way. Let $\mathbf{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots)$ be any algorithm in **Vector**, and let Δ be the maximum degree of G.

The key observation is that there is a *local* algorithm $\mathcal{B}_\Delta \in$ Vector such that $\mathcal{B}_\Delta$ and $\mathcal{A}_\Delta$ produce the same output S in (G,p). We can obtain such a local algorithm $\mathcal{B}_\Delta$ from algorithm $\mathcal{A}_\Delta$ by adding a counter that stops the computation after T steps, where T is the running time of $\mathcal{A}_\Delta$ on (G,p).

As we have a local algorithm $\mathcal{B}_\Delta$ that produces output S in (G,p), by Theorem 1b there is also a formula $\psi \in \mathrm{MML}(I^\Delta_{+,+})$ such that

$$S = \|\psi\|^{K_{+,+}(G,p)}.$$

By assumption, all nodes in X are bisimilar in the model $K_{+,+}(G,p)$. By Fact 1, there can be no $u,v \in X$ such that $u \in S$ and $v \notin S$. Hence we have $S \notin \Pi(G)$, and we conclude that **A** cannot solve Π. □

5 Relations between the Classes

Now we are ready to prove relations (1) and (2) that we gave in Section 2.

5.1 Equality $\mathsf{MV} = \mathsf{SV}$

Theorem 2 is the most important technical contribution of this work. Informally, it shows that *outgoing* port numbers necessarily break symmetry even if we do not have *incoming* port numbers—provided that we are not too short-sighted.

Theorem 2 *Let Π be a graph problem and let $T : \mathbb{N}\times\mathbb{N} \to \mathbb{N}$. Assume that there is an algorithm* $\mathbf{A} \in$ **Multiset** *that solves Π in time T. Then there is an algorithm* $\mathbf{B} \in$ **Set** *that solves Π in time $T + O(\Delta)$.*

To prove Theorem 2, we define the following local algorithm $\mathcal{C}_\Delta \in$ Set. Each node v constructs two sequences, $\beta_t(v)$ and $B_t(v)$ for $t = 0,1,\dots,2\Delta$. Before the first round, each node v sets $\beta_0(v) = \emptyset$ and $B_0(v) = \emptyset$. Then in round $t = 1,2,\dots,2\Delta$, each node v does the following:

1. Set $\beta_t(v) = (\beta_{t-1}(v), B_{t-1}(v))$.
2. For each port i, send $(\beta_t(v), \deg(v), i)$ to port i.
3. Let $B_t(v)$ be the set of all messages received by v.

Let $G = (V,E) \in \mathcal{F}(\Delta)$, and let p be a port numbering of graph G. We will analyse the execution of $\mathcal{C}_\Delta$ on (G,p). If $p((v,i)) = (u,j)$, we define that $\pi(v,u) = i$. That is, $\pi(v,u)$ is the outgoing port number in v that is connected to u. Let

$$m_t(u,v) = \big(\beta_t(u),\, \deg(u),\, \pi(u,v)\big)$$

denote the message that node u sends to node v in round t; it follows that $m_t(u,v) \in B_t(v)$ for all $\{u,v\} \in E$.

We begin with a following technical lemma. To pinpoint the key notion, let us call u and w a *pair of indistinguishable neighbours* of v in round t, if they are distinct neighbours of v such that

$$\beta_t(u) = \beta_t(w),\ \deg(u) = \deg(w), \text{ and } \pi(u,v) = \pi(w,v).$$

This is the same as saying that the node v receives the same message from u and w in round t. Let us say that u and w are a *pair of indistinguishable neighbours of order k* if further it holds that v has k distinct neighbours $v_1, v_2, \dots, v_k$ such that

$$\beta_t(u) = \beta_t(w) = \beta_t(v_i) \text{ for all } i = 1,2,\dots,k.$$

Here, u or w may belong to the set $\{v_1, v_2, \dots, v_k\}$. Note, however, that we do not require each pair (v_i, v_j) to be a pair of indistinguishables.

Lemma 1 *Suppose that u and w are a pair of indistinguishable neighbours of v of order k in round $t \geq 4$. Then u and w are a pair of indistinguishable neighbours of v of order $k+1$ in round $t-2$.*

Proof From $\beta_t(u) = \beta_t(w)$ it follows that $\beta_{t-2}(u) = \beta_{t-2}(w)$. This implies $m_{t-2}(u,v) = m_{t-2}(w,v)$.

For all $i = 1,2,\dots,k$, node v_i receives the message

$$m_{t-1}(v,v_i) = \big(\beta_{t-1}(v),\, \deg(v),\, \pi(v,v_i)\big)$$

from v in round $t-1$. By assumption, we have $\beta_t(v_i) = \beta_t(v_j)$ for all i and j, which implies $B_{t-1}(v_i) = B_{t-1}(v_j)$. Now $m_{t-1}(v,v_i) \in B_{t-1}(v_i)$ implies $m_{t-1}(v,v_j) \in B_{t-1}(v_i)$ for all i and j.

In any port numbering, we have $\pi(v,v_i) \neq \pi(v,v_j)$ for $i \neq j$. Therefore $m_{t-1}(v,v_i) \neq m_{t-1}(v,v_j)$, and $B_{t-1}(v_1)$ contains k distinct messages. That is, node v_1 has k distinct neighbours, $u_1, u_2, \dots, u_k$, such that

$$\begin{aligned}\big(\beta_{t-1}(u_i),\, \deg(u_i),\, \pi(u_i,v_1)\big) = m_{t-1}(u_i,v_1) &= m_{t-1}(v,v_i)\\ &= \big(\beta_{t-1}(v),\, \deg(v),\, \pi(v,v_i)\big).\end{aligned}$$

In particular, $\beta_{t-1}(u_i) = \beta_{t-1}(v)$ for all i.

Now let us investigate the messages that the nodes u_i receive in round $t-2$. We have

$$m_{t-2}(v_1,u_i) = \big(\beta_{t-2}(v_1),\, \deg(v_1),\, \pi(v_1,u_i)\big).$$

However, $\beta_{t-1}(u_i) = \beta_{t-1}(v)$ implies $B_{t-2}(u_i) = B_{t-2}(v)$ for all i. In particular,

$$m_{t-2}(v_1,u_i) \in B_{t-2}(v)$$

for all i. Now $\pi(v_1,u_i) \neq \pi(v_1,u_j)$ implies $m_{t-2}(v_1,u_i) \neq m_{t-2}(v_1,u_j)$ for all $i \neq j$.

To summarise, v receives the following messages in round $t-2$: k distinct messages,

$$m_{t-2}(v_1,u_i) = \big(\beta_{t-2}(v_1),\, \deg(v_1),\, \pi(v_1,u_i)\big)$$

for $i = 1,2,\dots,k$, and two identical messages,

$$m_{t-2}(u,v) = m_{t-2}(w,v) = \big(\beta_{t-2}(u),\, \deg(u),\, \pi(u,v)\big).$$

Moreover, $\beta_{t-2}(v_1) = \beta_{t-2}(u)$. Hence v receives at least $k+1$ messages in round $t-2$, each of the form $(\beta_{t-2}(u),\cdot,\cdot)$. Therefore v has at least $k+1$ distinct neighbours v'_i with $\beta_{t-2}(u) = \beta_{t-2}(v'_i)$. □

Lemma 2 *If a node v has two indistinguishable neighbours u and w of order k in round $2t$, then v has at least $t+k-1$ neighbours. Consequently, no node has a pair of indistinguishable neighbours in round 2Δ.*

Proof The proof of the first claim is by induction on t. The base case $t=1$ is trivial. For the inductive step we can apply Lemma 1.

For the consequence, we observe that if u and w were a pair of indistinguishable neighbours of v, then the first claim would imply that $\deg(v) \geq \Delta + 1$, which is a contradiction, as the maximum degree of G is at most Δ. □

To summarise: $m_{2\Delta}(u,v) \neq m_{2\Delta}(w,v)$ whenever u and w are two distinct neighbours of v in G. In particular,

$$\big(\beta_{2\Delta}(u),\, \deg(u),\, \pi(u,v)\big) \neq \big(\beta_{2\Delta}(w),\, \deg(w),\, \pi(w,v)\big).$$

Once we have finished running $\mathcal{C}_\Delta$, which takes $O(\Delta)$ time, we can *simulate* the execution of $\mathcal{A}_\Delta \in \mathsf{Multiset}$ with an algorithm $\mathcal{B}_\Delta \in \mathsf{Set}$ as follows: if a node u in the execution of $\mathcal{A}_\Delta$ sends the message a to port i, algorithm $\mathcal{B}_\Delta$ sends the message

$$\big(\beta_{2\Delta}(u), \deg(u),\, i,\, a\big)$$

to port i. Now all messages received by a node are distinct. Hence given the set of messages received by a node v in $\mathcal{B}_\Delta$, we can reconstruct the multiset of messages received by v in $\mathcal{A}_\Delta$. This concludes the proof of Theorem 2.

Corollary 2 *We have* $\mathsf{MV} = \mathsf{SV}$ *and* $\mathsf{MV}(1) = \mathsf{SV}(1)$.

Proof Immediate from Theorem 2. □

Remark 3 With minor modifications, the proof of Theorem 2 would also imply $\mathsf{VV} = \mathsf{MV} = \mathsf{SV}$. However, as we will see next, there is also a more direct way to prove $\mathsf{VV} = \mathsf{MV}$. The proof in the following section avoids the additive overhead in running time (but the overhead in message size may be much larger).

5.2 Equalities $\mathsf{VB} = \mathsf{MB}$ and $\mathsf{VV} = \mathsf{MV}$

The following theorem is implicit in prior work [5, Section 5]; we give a bit more detailed version for the general case here. The basic idea is that algorithm $\mathbf{B}$ augments each message with the full communication history, and orders the incoming messages lexicographically by the communication histories—the end result is equal to the execution of algorithm $\mathbf{A}$ in the same graph G for a very specific choice of incoming port numbers.

Theorem 3 *Let Π be a graph problem and let $T\colon \mathbb{N}\times\mathbb{N} \to \mathbb{N}$. Assume that there is an algorithm $\mathbf{A} \in$ **Vector** that solves Π in time T. Then there is an algorithm $\mathbf{B} \in$ **Multiset** that solves Π in time T.*

Proof Let $\mathbf{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots) \in$ **Vector** be an algorithm, and let $G \in \mathcal{F}(\Delta)$ be a graph of maximum degree at most Δ. Consider a port numbering p of G, and the execution of $\mathcal{A}_\Delta$ on (G,p). For each port $(u,i) \in P(G)$, let

$$\beta_t(u,i) = (a_1(u,i), a_2(u,i), \dots, a_t(u,i))$$

be the full history of messages that node u received from port i in rounds $1,2,\dots,t$. Let $<$ be the lexicographical order of such vectors, that is, $\beta_t(u,i) < \beta_t(v,j)$ if there is a time ℓ such that $a_\ell(u,i) <_M a_\ell(v,j)$ and $a_k(u,i) = a_k(v,j)$ for $k = 1,2,\dots,\ell-1$. Here $<_M$ is a fixed order of the message set M of $\mathcal{A}_\Delta$.

We say that p is *compatible with the message history up to time t* if $\beta_t(u,i) \leq \beta_t(u,j)$ for all nodes $u \in V$ and all $i < j$. Clearly, if p is compatible with the message history up to time t, it is also compatible with the message history up to time $t-1$.

Now fix any port numbering p_0 of G. Let $\mathcal{P}_0$ be the family of all port numberings p of G such that for each port $(u,i) \in P(G)$ there are v, j, and k such that $p((u,i)) = (v,j)$ and $p_0((u,i)) = (v,k)$. Put otherwise, any $p \in \mathcal{P}_0$ is equivalent to p_0 from the perspective of a $\mathsf{Multiset}$ algorithm. We make the following observations:

1. $\mathcal{P}_0$ is non-empty, and each $p \in \mathcal{P}_0$ is compatible with the message history up to time 0.
2. State vector x_0 at time 0 does not depend on the choice of $p \in \mathcal{P}_0$.
3. The message sent by a node v to port j in round 1 does not depend on the choice of $p \in \mathcal{P}_0$.
4. There is at least one $p \in \mathcal{P}_0$ that is compatible with the message history up to time 1.

Now let $\mathcal{P}_t \subseteq \mathcal{P}_{t-1}$ consist of *all* port numberings $p \in \mathcal{P}_{t-1}$ that are compatible with the message history up to time t. By induction, we have:

1. $\mathcal{P}_t$ is non-empty, and each $p \in \mathcal{P}_t$ is compatible with the message history up to time t.
2. The vector $\mathbf{a}_t(u)$ of messages received by u in round t does not depend on the choice of $p \in \mathcal{P}_t$. State vector x_t at time t does not depend on the choice of $p \in \mathcal{P}_t$.
3. The message sent by a node v to port j in round $t+1$ does not depend on the choice of $p \in \mathcal{P}_t$.
4. There is at least one $p \in \mathcal{P}_t$ that is compatible with the message history up to time $t+1$.

Let $T = T(\Delta, |V|)$. In particular, $\mathcal{A}_\Delta$ stops in time T in (G,p) for any $p \in \mathcal{P}_T$. Intuitively, a port numbering $p \in \mathcal{P}_T$ is constructed as follows: we have copied the *outgoing* port numbers from a given port numbering p_0, and we have adjusted the *incoming* port numbers so that p becomes compatible with the message history up to time T. This choice of incoming port numbers is particularly convenient from the perspective of the $\mathsf{Multiset}$ model: $\beta_t(u,i) < \beta_t(u,j)$ implies

$i < j$, and $\beta_t(u,i) = \beta_t(u,j)$ implies $a_t(u,i) = a_t(u,j)$. That is, if we know the multiset

$$\text{multiset}(\beta_t(u,1), \beta_t(u,2), \ldots, \beta_t(u,\Delta)),$$

we can reconstruct the vector $\mathbf{a}_t(u)$.

We design an algorithm $\mathcal{B}_\Delta \in \mathsf{Multiset}$ with the following property: the execution of $\mathcal{B}_\Delta$ on (G, p_0) simulates the execution of $\mathcal{A}_\Delta$ on (G, p), where $p \in \mathcal{P}_T$. Note that the output of $\mathcal{A}_\Delta$ does not depend on the choice of $p \in \mathcal{P}_T$. As the output of $\mathcal{A}_\Delta$ is in $\Pi(G)$ for any port numbering of G, it follows that the output of $\mathcal{B}_\Delta$ is also in $\Pi(G)$.

The simulation works as follows. For each port $(v, j) \in P(G)$, algorithm $\mathcal{B}_\Delta$ keeps track of all messages sent by node v to port j in $\mathcal{A}_\Delta$. Each outgoing message is augmented with the full message history. Hence in each round t, a node u can reconstruct the unique vector $\mathbf{a}_t(u)$ that matches the execution of $\mathcal{A}_\Delta$ on (G, p) for any $p \in \mathcal{P}_t$. □

Theorem 4 *Let Π be a graph problem. Let $T\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. Assume that there is an algorithm $\mathbf{A} \in$ **Broadcast** that solves Π in time T. Then there is an algorithm $\mathbf{B} \in$ **Multiset** $\cap$ **Broadcast** that solves Π in time T.*

Proof This is similar to the proof of Theorem 3. In the simulation, each node keeps track of the history of broadcasts. □

Corollary 3 *We have* $\mathsf{VB} = \mathsf{MB}$, $\mathsf{VB}(1) = \mathsf{MB}(1)$, $\mathsf{VV} = \mathsf{MV}$, *and* $\mathsf{VV}(1) = \mathsf{MV}(1)$.

Proof Follows from Theorems 3 and 4. □

Remark 4 Boldi et al. [10] and Yamashita and Kameda [62] already give simulation results that, in essence, imply $\mathsf{VB} = \mathsf{MB}$ and $\mathsf{VV} = \mathsf{MV}$ (albeit for a slightly different model of computation). However, in prior work, the simulation overhead is linear in the number of nodes; in particular, it does not imply $\mathsf{VB}(1) = \mathsf{MB}(1)$ or $\mathsf{VV}(1) = \mathsf{MV}(1)$.

5.3 Separating the Classes

Trivially, $\mathsf{SB} \subseteq \mathsf{MB} \subseteq \mathsf{MV}$ and $\mathsf{SB}(1) \subseteq \mathsf{MB}(1) \subseteq \mathsf{MV}(1)$. Together with Corollaries 2 and 3 these imply

$$\mathsf{SB} \subseteq \mathsf{MB} = \mathsf{VB} \subseteq \mathsf{SV} = \mathsf{MV} = \mathsf{VV},$$
$$\mathsf{SB}(1) \subseteq \mathsf{MB}(1) = \mathsf{VB}(1) \subseteq \mathsf{SV}(1) = \mathsf{MV}(1) = \mathsf{VV}(1).$$

Now we proceed to show that the subset relations are proper. We only need to come up with a graph problem that separates a pair of classes—here the connections to modal logic and bisimulation are a particularly helpful tool. Many of the separation results are already known by prior work (in particular, see Yamashita and Kameda [62]), but we give the proofs here to demonstrate the use of bisimulation arguments.

For the case of $\mathsf{VB} \neq \mathsf{SV}$, the separation is easy: we can consider the problem of breaking symmetry in a star graph.

Theorem 5 *There is a graph problem Π such that $\Pi \in \mathsf{SV}(1)$ and $\Pi \notin \mathsf{VB}$.*

Proof An appropriate choice of Π is the (artificial) problem of selecting a leaf node in a star graph. More formally, we have the set of outputs $Y = \{0,1\}$. We define $\Pi(G)$ as follows, depending on G:

1. $G = (V,E)$ is a k-star for a $k > 1$. That is,
$$V = \{c, v_1, v_2, \ldots, v_k\},$$
$$E = \{\{c, v_i\} : i = 1, 2, \ldots, k\}.$$
Then we have $S \in \Pi(G)$ if $S\colon V \to Y$, $S(c) = 0$, and there is a j such that $S(v_j) = 1$ and $S(v_i) = 0$ for all $i \neq j$.
2. $G = (V,E)$ is not a star. Then we do not restrict the output, i.e., $S \in \Pi(G)$ for any function $S\colon V \to Y$.

It is easy to design a local algorithm $\mathbf{A} \in$ **Set** that solves problem Π: First, all nodes send message i to port i for each i; then a node outputs 1 if it has degree 1 and if it received the set of messages $\{1\}$. Thus, Π is in $\mathsf{SV}(1)$.

We use Corollary 1b to prove that Π is not in VB. Let $G = (V,E)$ be a k-star, and let $X \subseteq V$ be the set of leaf nodes of G. Then Π and X satisfy the assumption in Corollary 1. Furthermore, it is easy to see that given any port numbering p of graph G, all nodes in X are bisimilar in the model $K_{+,-}(G,p)$. □

Corollary 4 *We have* $\mathsf{VB} \neq \mathsf{SV}$ *and* $\mathsf{VB}(1) \neq \mathsf{SV}(1)$.

Proof Follows from Theorem 5. □

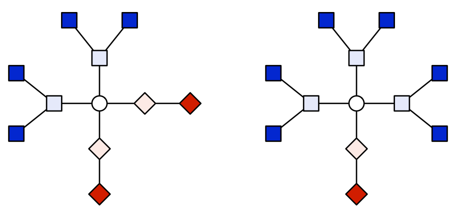
To show that $\mathsf{SB} \neq \mathsf{MB}$, we can consider, for example, the problem of identifying nodes that have an odd number of neighbours with odd degrees.

Theorem 6 *There is a graph problem Π such that $\Pi \in \mathsf{MB}(1)$ and $\Pi \notin \mathsf{SB}$.*

Proof We define Π as follows. Let $G = (V,E)$ and $S\colon V \to \{0,1\}$. We have $S \in \Pi(G)$ if the following holds: $S(v) = 1$ iff v is a node with an odd number of neighbours of an odd degree.

The problem is trivially in $\mathsf{MB}(1)$: first each node broadcasts the parity of its degree, and then a node outputs 1 if it received an odd number of messages that indicate the odd parity.

To see that the problem is not in SB, it is sufficient to argue that the white nodes in the following graphs are bisimilar, yet they are supposed to produce different outputs.



More precisely, we can partition the nodes in five equivalence classes (indicated with the shading and shapes in the above illustration), and the nodes in the same equivalence class are bisimilar in the Kripke model $K_{-,-}(G,p)$; recall that the model is independent of the choice of the port numbering p. Thus, we can apply Corollary 1c with X consisting of the two white nodes. □

Corollary 5 *We have* $\mathsf{SB} \neq \mathsf{MB}$ *and* $\mathsf{SB}(1) \neq \mathsf{MB}(1)$.

Proof Follows from Theorem 6. □

Finally, to separate VV and $\mathsf{VV_c}$, we make use of the fact that there are graphs G such that some inconsistent port numbering of G is totally symmetric, while all consistent port numberings of G necessarily break symmetry between nodes.

We start by proving that any regular graph has a totally symmetric port numbering. Recall that a graph G is *k-regular* if $\deg(v) = k$ for every node v of G. Furthermore, G is *regular* if it is k-regular for some $k \in \mathbb{N}$. Recall also that a *1-factor* (or *perfect matching*) of a graph $G = (V,E)$ is a set $F \subseteq E$ of edges such that every node $v \in V$ has degree 1 in the graph (V,F).

Lemma 3 *If G is a regular graph, then there is a port numbering p of G such that all nodes of G are bisimilar in the model $K_{+,+}(G,p)$.*

Proof Assume that $G = (V,E)$ is k-regular. Let $V_s = V \times \{s\}$ for $s \in \{1,2\}$, and let $E^* = \{\{(u,1),(v,2)\} : \{u,v\} \in E\}$. Then $G^* = (V_1 \cup V_2, E^*)$ is a k-regular bipartite graph; see Fig. 8. It is a well-known corollary of Hall's marriage theorem [21, Section 2.1] that the edge set of any such graph is the union of k mutually disjoint 1-factors. Thus, there are sets $E_1,\dots,E_k \subseteq E^*$ such that $E_i \cap E_j = \emptyset$ whenever $i \neq j$, and each E_i is a one-to-one correspondence between the sets V_1 and V_2.

Instead of defining a port numbering p, we use the sets E_i to define a Kripke model

$$K = (V, (R_\alpha)_{\alpha \in I^k_{+,+}}, \tau).$$

For each $i \in [k]$, we let $R_{(i,i)} = \{(u,v) : \{(u,1),(v,2)\} \in E_i\}$, and if $i \neq j$, we let $R_{(i,j)} = \emptyset$. Furthermore, we let τ to be as in the definition of the models $K_{a,b}(G,p)$. Clearly, there is a port numbering p such that $K = K_{+,+}(G,p)$. Moreover, for every node $u \in V$ the set $\{v \in V : (u,v) \in R_{(i,j)}\}$ is nonempty if and only if $i = j$. Using this, it is easy to see that the full relation $Z = V \times V$ is a bisimulation, whence all nodes are bisimilar in the model $K_{+,+}(G,p)$. □

Note that the converse of Lemma 3 is true as well: if u and v are nodes in G such that $\deg(u) \neq \deg(v)$, then u and v obviously cannot be bisimilar in the model $K_{+,+}(G,p)$ for any port numbering p.

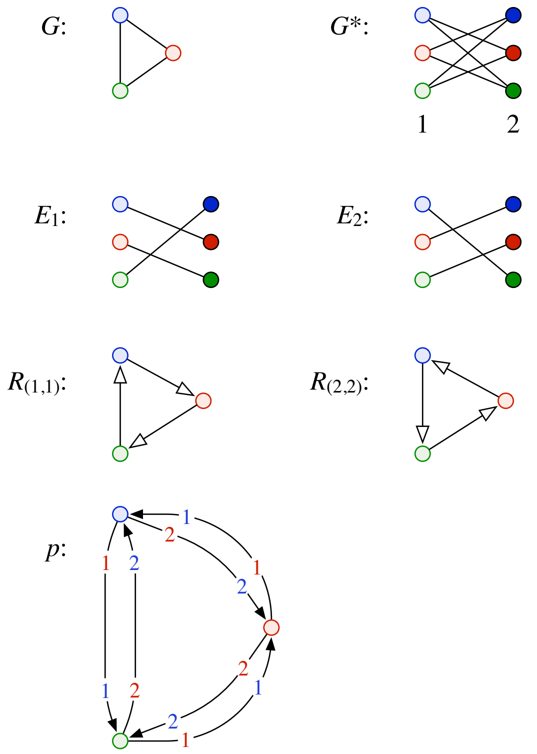


Fig. 8 An illustration of the proof of Lemma 3.

Our next aim is to show that there is a class of regular graphs G such that all consistent port numberings of G break symmetry—this is known from prior work [62] but we give a proof here for completeness.

Lemma 4 *If G is a k-regular graph for an odd k, and there is a consistent port numbering p of G such that all its nodes are bisimilar in the model $K_{+,+}(G,p)$, then G has a 1-factor.*

Proof Let p be a consistent port numbering of $G = (V,E)$ such that all nodes of G are bisimilar in $K_{+,+}(G,p)$. Let $F \subseteq [k]^2$ be the relation $\{(i,j) \in [k]^2 : R_{(i,j)} \neq \emptyset\}$. Then F is a function, since otherwise there are $u,v,u',v' \in V$ and such that $(u,v) \in R_{(i,j)}$ and $(u',v') \in R_{(i,j')}$ for some $i,j,j' \in [k]$ with $j \neq j'$, which would imply that u and u' are not bisimilar. Note further, that by consistency of p, relation F is symmetric: if $(i,j) \in F$, then $(j,i) \in F$. Thus, F is a permutation of $[k]$ such that $F^{-1} = F$. Since k is odd, there exists $i \in [k]$ such that $(i,i) \in F$. It is now easy to see that the relation $R_{(i,i)}$ is a 1-factor of G. □

By the previous two lemmas, each regular graph of odd degree and without 1-factors has an inconsistent symmetric port numbering, but no consistent symmetric port numberings. In the proof of the separation result, we also need the assumption that all graphs we consider are connected. Thus, we define $\mathcal{G}$ to be the class of all connected regular graphs of odd degree which do not have a 1-factor. It is easy to construct k-regular graphs in $\mathcal{G}$ for each odd degree $k \geq 3$.

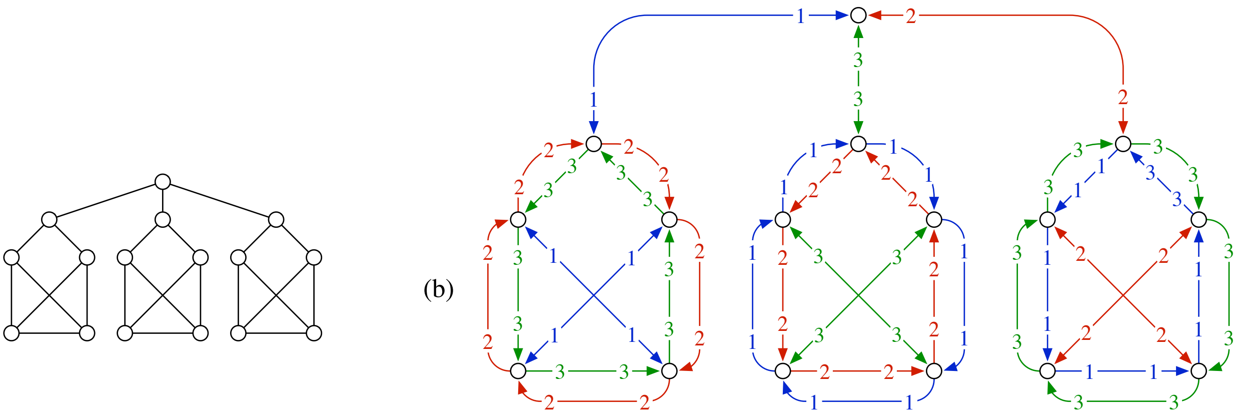


Fig. 9 (a) A 3-regular graph G that does not have a 1-factor [14, Figure 5.10]. (b) A symmetric port numbering of G.

The graph illustrated in Fig. 9a is an example with $k = 3$. Fig. 9b shows an example of an inconsistent symmetric port numbering of the same graph.

Theorem 7 *There is a graph problem Π such that $\Pi \in \mathsf{VV_c}(1)$ and $\Pi \notin \mathsf{VV}$.*

Proof We define the graph problem Π as follows: For all graphs $G = (V,E) \in \mathcal{G}$, $\Pi(G)$ consists of all non-constant functions $S\colon V \to \{0,1\}$, that is, we have $u, v \in V$ with $S(u) \neq S(v)$. For all graphs $G \notin \mathcal{G}$, $\Pi(G)$ consists of all functions $S\colon V \to \{0,1\}$.

Let us first prove that the problem is in $\mathsf{VV_c}(1)$. Let $G = (V,E) \in \mathcal{F}(\Delta)$ be a graph and p a consistent port numbering of G. We define the local type of a node $v \in V$ to be the tuple $t(v) = (j_1, j_2, \dots, j_\Delta)$, where j_i is the number of the port of a neighbour u to which the port (v,i) is connected if $i \in [\deg(v)]$, and $j_i = 0$ for $i > \deg(v)$. Fix some linear ordering $\leq$ of the local types. Then there is a local algorithm $\mathcal{A}_\Delta \in \mathsf{Vector}$ such that its output on a node $v \in V$ is 1 if $t(u) \leq t(v)$ for all neighbours u of v, and 0 otherwise: $\mathcal{A}_\Delta$ computes first in one step the local types of the nodes, and then in a second step it sends the types to neighbouring nodes, compares the types, and outputs either 0 or 1 depending on the comparison.

The crucial observation now is that if $G \in \mathcal{G}$, then G has nodes with different local types. This is seen as follows. If the local types of all nodes of G are the same, then it is easy to see that all nodes are bisimilar in the model $K_{+,+}(G,p)$. Thus, by Lemma 4, either G is not k-regular for any odd k, or G has a 1-factor.

Assume then that $G \in \mathcal{G}$ and p is a consistent port numbering of G, and consider the output $S\colon V \to \{0,1\}$ that is produced by $\mathcal{A}_\Delta$ in (G,p). Since the local types of all nodes are not the same and G is connected, there are nodes $u, v \in V$ such that $t(u) < t(v)$, and $t(v)$ is maximal w.r.t. the ordering $\leq$. This means that $S(u) = 0$ and $S(v) = 1$, whence $S \in \Pi(G)$. We conclude that the sequence $\mathbf{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots)$ of algorithms solves Π assuming consistency.

To see that Π is not in VV, consider a graph $G = (V,E) \in \mathcal{G}$. By Lemma 3, there exists a port numbering p of G such that all nodes of G are bisimilar in the model $K_{+,+}(G,p)$ (as seen above, p is inconsistent). The claim follows now from Corollary 1a, since the graph problem Π and the set $X = V$ clearly satisfy its assumption. □

Corollary 6 *We have $\mathsf{VV_c} \neq \mathsf{VV}$ and $\mathsf{VV_c}(1) \neq \mathsf{VV}(1)$.*

Proof Follows from Theorem 7. □

5.4 Conclusions

In summary, we have established that the classes we have studied form a linear order of length four:

$$\mathsf{SB} \subsetneq \mathsf{MB} = \mathsf{VB} \subsetneq \mathsf{SV} = \mathsf{MV} = \mathsf{VV} \subsetneq \mathsf{VV_c}, \tag{1}$$

$$\begin{aligned} \mathsf{SB}(1) &\subsetneq \mathsf{MB}(1) = \mathsf{VB}(1) \\ &\subsetneq \mathsf{SV}(1) = \mathsf{MV}(1) = \mathsf{VV}(1) \subsetneq \mathsf{VV_c}(1). \end{aligned} \tag{2}$$

As a corollary of (2) and Theorem 1, we can make, for example, the following observations.

1. MML captures the same class of problems on $\mathcal{K}_{+,+}$ and $\mathcal{K}_{-,+}$.
2. Both MML and GMML capture the same class of problems on $\mathcal{K}_{-,+}$.
3. The class of problems captured by MML becomes strictly smaller if we replace $\mathcal{K}_{-,+}$ with $\mathcal{K}_{+,-}$.
4. MML on $\mathcal{K}_{+,-}$ captures the same class of problems as GML on $\mathcal{K}_{-,-}$.

Open Questions Related to Equalities. In the proofs of Corollaries 2 and 3, our main focus was on devising a simulation scheme in which the simulation overhead is only proportional to maximum degree Δ and running time T—this implies that local algorithms of a stronger model can be simulated with local algorithms in a weaker model. However, in our approach the simulation overhead is large in terms of message size. It is an open question if such a high overhead is necessary.

Open Questions Related to Separations. To keep the proofs of Theorems 5, 6, and 7 as simple as possible, we introduced graph problems that were highly contrived. An interesting challenge is to come up with *natural* graph problems that could be used to prove the same separation results. It should be noted that prior work [10, 62] presents some separation results that use a natural graph problem—leader election. However, leader election is a global problem; it cannot be solved in $\mathsf{VV_c}(1)$, and hence we cannot use it to separate any of the constant-time versions of the classes.

Another challenge is to come up with *decision problems* that separate the classes. Indeed, it is not known if the separation results hold if we restrict ourselves to decision problems.

Acknowledgements This work is an extended and revised version of a preliminary conference report [34]. We thank anonymous reviewers for their helpful feedback, and Jérémie Chalopin, Mika Göös, and Joel Kaasinen for discussions and comments.

This work was supported in part by Academy of Finland (grants 129761, 132380, 132812, and 252018), the research funds of University of Helsinki, and Finnish Cultural Foundation. Part of this work was conducted while Tuomo Lempiäinen was with the Department of Information and Computer Science at Aalto University.

References

1. Afek, Y., Alon, N., Bar-Joseph, Z., Cornejo, A., Haeupler, B., Kuhn, F.: Beeping a maximal independent set. In: Proc. 25th International Symposium on Distributed Computing (DISC 2011), *Lecture Notes in Computer Science*, vol. 6950, pp. 32–50. Springer (2011). doi:10.1007/978-3-642-24100-0_3.
2. Angluin, D.: Local and global properties in networks of processors. In: Proc. 12th Annual ACM Symposium on Theory of Computing (STOC 1980), pp. 82–93. ACM Press (1980). doi:10.1145/800141.804655.
3. Åstrand, M., Floréen, P., Polishchuk, V., Rybicki, J., Suomela, J., Uitto, J.: A local 2-approximation algorithm for the vertex cover problem. In: Proc. 23rd International Symposium on Distributed Computing (DISC 2009), *Lecture Notes in Computer Science*, vol. 5805, pp. 191–205. Springer (2009). doi:10.1007/978-3-642-04355-0_21.
4. Åstrand, M., Polishchuk, V., Rybicki, J., Suomela, J., Uitto, J.: Local algorithms in (weakly) coloured graphs (2010). arXiv:1002.0125.
5. Åstrand, M., Suomela, J.: Fast distributed approximation algorithms for vertex cover and set cover in anonymous networks. In: Proc. 22nd Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2010), pp. 294–302. ACM Press (2010). doi:10.1145/1810479.1810533.
6. Attiya, H., Snir, M., Warmuth, M.K.: Computing on an anonymous ring. Journal of the ACM **35**(4), 845–875 (1988). doi:10.1145/48014.48247.
7. Benthem, J.v.: Modal correspondence theory. Ph.D. thesis, Instituut voor Logica en Grondslagenonderzoek van de Exacte Wetenschappen, Universiteit van Amsterdam (1977)
8. Blackburn, P., Benthem, J.v., Wolter, F. (eds.): Handbook of Modal Logic, *Studies in Logic and Practical Reasoning*, vol. 3. Elsevier, Amsterdam (2007)
9. Blackburn, P., Rijke, M.d., Venema, Y.: Modal Logic, *Cambridge Tracts in Theoretical Computer Science*, vol. 53. Cambridge University Press, Cambridge, UK (2001)
10. Boldi, P., Shammah, S., Vigna, S., Codenotti, B., Gemmell, P., Simon, J.: Symmetry breaking in anonymous networks: characterizations. In: Proc. 4th Israel Symposium on the Theory of Computing and Systems (ISTCS 1996), pp. 16–26. IEEE (1996)
11. Boldi, P., Vigna, S.: Computing vector functions on anonymous networks. In: Proc. 4th Colloquium on Structural Information and Communication Complexity (SIROCCO 1997), pp. 201–214. Carleton Scientific (1997)
12. Boldi, P., Vigna, S.: Computing anonymously with arbitrary knowledge. In: Proc. 18th Annual ACM Symposium on Principles of Distributed Computing (PODC 1999), pp. 181–188. ACM Press (1999). doi:10.1145/301308.301355.
13. Boldi, P., Vigna, S.: An effective characterization of computability in anonymous networks. In: Proc. 15th International Symposium on Distributed Computing (DISC 2001), *Lecture Notes in Computer Science*, vol. 2180, pp. 33–47. Springer (2001). doi:10.1007/3-540-45414-4_3.
14. Bondy, J.A., Murty, U.S.R.: Graph Theory with Applications. North-Holland, New York (1976)
15. Chalopin, J.: Algorithmique distribuée, calculs locaux et homomorphismes de graphes. Ph.D. thesis, LaBRI, Université Bordeaux 1 (2006)
16. Chalopin, J., Das, S., Santoro, N.: Groupings and pairings in anonymous networks. In: Proc. 20th International Symposium on Distributed Computing (DISC 2006), *Lecture Notes in Computer Science*, vol. 4167, pp. 105–119. Springer (2006). doi:10.1007/11864219_8.
17. Conradie, W.: Definability and changing perspectives: The beth property for three extensions of modal logic. Master's thesis, Institute for Logic, Language and Computation, University of Amsterdam (2002)
18. Cornejo, A., Kuhn, F.: Deploying wireless networks with beeps. In: Proc. 24th International Symposium on Distributed Computing (DISC 2010), *Lecture Notes in Computer Science*, vol. 6343, pp. 148–162. Springer (2010). doi:10.1007/978-3-642-15763-9_15.
19. Czygrinow, A., Hańćkowiak, M., Krzywdziński, K., Szymańska, E., Wawrzyniak, W.: Brief announcement: distributed approximations for the semi-matching problem. In: Proc. 25th International Symposium on Distributed Computing (DISC 2011), *Lecture Notes in Computer Science*, vol. 6950, pp. 200–201. Springer (2011). doi:10.1007/978-3-642-24100-0_18.
20. Czygrinow, A., Hańćkowiak, M., Wawrzyniak, W.: Fast distributed approximations in planar graphs. In: Proc. 22nd International Symposium on Distributed Computing (DISC 2008), *Lecture Notes in Computer Science*, vol. 5218, pp. 78–92. Springer (2008). doi:10.1007/978-3-540-87779-0_6.
21. Diestel, R.: Graph Theory, 4th edn. Springer, Berlin (2010). http://diestel-graph-theory.com/.
22. Diks, K., Kranakis, E., Malinowski, A., Pelc, A.: Anonymous wireless rings. Theoretical Computer Science **145**(1–2), 95–109 (1995). doi:10.1016/0304-3975(94)00178-L.
23. Emek, Y., Smula, J., Wattenhofer, R.: Stone age distributed computing (2012). arXiv:1202.1186.
24. Fine, K.: In so many possible worlds. Notre Dame Journal of Formal Logic **13**(4), 516–520 (1972). doi:10.1305/ndjfl/1093890715.
25. Flocchini, P., Roncato, A., Santoro, N.: Computing on anonymous networks with sense of direction. Theoretical Computer Science **301**(1–3), 355–379 (2003). doi:10.1016/S0304-3975(02)00592-3.
26. Floréen, P., Hassinen, M., Kaasinen, J., Kaski, P., Musto, T., Suomela, J.: Local approximability of max-min and min-max linear programs. Theory of Computing Systems **49**(4), 672–697 (2011). doi:10.1007/s00224-010-9303-6.
27. Floréen, P., Hassinen, M., Kaski, P., Suomela, J.: Local approximation algorithms for a class of 0/1 max-min linear programs (2008). arXiv:0806.0282.
28. Floréen, P., Hassinen, M., Kaski, P., Suomela, J.: Tight local approximation results for max-min linear programs. In: Proc. 4th

International Workshop on Algorithmic Aspects of Wireless Sensor Networks (Algosensors 2008), *Lecture Notes in Computer Science*, vol. 5389, pp. 2–17. Springer (2008). doi:10.1007/978-3-540-92862-1_2. arXiv:0804.4815.

29. Floréen, P., Kaasinen, J., Kaski, P., Suomela, J.: An optimal local approximation algorithm for max-min linear programs. In: Proc. 21st Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2009), pp. 260–269. ACM Press (2009). doi:10.1145/1583991.1584058. arXiv:0809.1489.
30. Floréen, P., Kaski, P., Musto, T., Suomela, J.: Approximating max-min linear programs with local algorithms. In: Proc. 22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS 2008). IEEE (2008). doi:10.1109/IPDPS.2008.4536235. arXiv:0710.1499.
31. Floréen, P., Kaski, P., Polishchuk, V., Suomela, J.: Almost stable matchings by truncating the Gale–Shapley algorithm. Algorithmica **58**(1), 102–118 (2010). doi:10.1007/s00453-009-9353-9. arXiv:0812.4893.
32. Göös, M., Hirvonen, J., Suomela, J.: Lower bounds for local approximation. In: Proc. 31st Annual ACM Symposium on Principles of Distributed Computing (PODC 2012), pp. 175–184. ACM Press (2012). doi:10.1145/2332432.2332465. arXiv:1201.6675.
33. Halpern, J.Y., Moses, Y.: Knowledge and common knowledge in a distributed environment. Journal of the ACM **37**(3), 549–587 (1990). doi:10.1145/79147.79161.
34. Hella, L., Järvisalo, M., Kuusisto, A., Laurinharju, J., Lempiäinen, T., Luosto, K., Suomela, J., Virtema, J.: Weak models of distributed computing, with connections to modal logic. In: Proc. 31st Annual ACM Symposium on Principles of Distributed Computing (PODC 2012), pp. 185–194. ACM Press (2012). doi:10.1145/2332432.2332466. arXiv:1205.2051.
35. Immerman, N.: Descriptive Complexity. Graduate Texts in Computer Science. Springer, Berlin (1999)
36. Kuhn, F.: The price of locality: Exploring the complexity of distributed coordination primitives. Ph.D. thesis, ETH Zurich (2005)
37. Kuhn, F., Moscibroda, T., Wattenhofer, R.: The price of being near-sighted. In: Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2006), pp. 980–989. ACM Press (2006). doi:10.1145/1109557.1109666.
38. Kuhn, F., Wattenhofer, R.: On the complexity of distributed graph coloring. In: Proc. 25th Annual ACM Symposium on Principles of Distributed Computing (PODC 2006), pp. 7–15. ACM Press (2006). doi:10.1145/1146381.1146387.
39. Lenzen, C.: Synchronization and symmetry breaking in distributed systems. Ph.D. thesis, ETH Zurich (2011)
40. Lenzen, C., Oswald, Y.A., Wattenhofer, R.: What can be approximated locally? TIK Report 331, ETH Zurich, Computer Engineering and Networks Laboratory (2010). ftp://ftp.tik.ee.ethz.ch/pub/publications/TIK-Report-331.pdf.
41. Lenzen, C., Wattenhofer, R.: Minimum dominating set approximation in graphs of bounded arboricity. In: Proc. 24th International Symposium on Distributed Computing (DISC 2010), *Lecture Notes in Computer Science*, vol. 6343, pp. 510–524. Springer (2010). doi:10.1007/978-3-642-15763-9_48.
42. Linial, N.: Locality in distributed graph algorithms. SIAM Journal on Computing **21**(1), 193–201 (1992). doi:10.1137/0221015.
43. Mayer, A., Naor, M., Stockmeyer, L.: Local computations on static and dynamic graphs. In: Proc. 3rd Israel Symposium on the Theory of Computing and Systems (ISTCS 1995), pp. 268–278. IEEE (1995). doi:10.1109/ISTCS.1995.377023.
44. Moran, S., Warmuth, M.K.: Gap theorems for distributed computation. SIAM Journal on Computing **22**(2), 379–394 (1993). doi:10.1137/0222028.
45. Moscibroda, T.: Locality, scheduling, and selfishness: Algorithmic foundations of highly decentralized networks. Ph.D. thesis, ETH Zurich (2006)
46. Naor, M., Stockmeyer, L.: What can be computed locally? SIAM Journal on Computing **24**(6), 1259–1277 (1995). doi:10.1137/S0097539793254571.
47. Norris, N.: Classifying anonymous networks: when can two networks compute the same set of vector-valued functions? In: Proc. 1st Colloquium on Structural Information and Communication Complexity (SIROCCO 1994), pp. 83–98. Carleton University Press (1995)
48. Norris, N.: Computing functions on partially wireless networks. In: Proc. 2nd Colloquium on Structural Information and Communication Complexity (SIROCCO 1995), pp. 53–64. Carleton University Press (1996)
49. Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, Philadelphia (2000)
50. Petersen, J.: Die Theorie der regulären graphs. Acta Mathematica **15**(1), 193–220 (1891). doi:10.1007/BF02392606.
51. Polishchuk, V., Suomela, J.: A simple local 3-approximation algorithm for vertex cover. Information Processing Letters **109**(12), 642–645 (2009). doi:10.1016/j.ipl.2009.02.017. arXiv:0810.2175.
52. de Rijke, M.: A note on graded modal logic. Studia Logica **64**(2), 271–283 (2000). doi:10.1023/A:1005245900406.
53. Sangiorgi, D.: On the origins of bisimulation and coinduction. ACM Transactions on Programming Languages and Systems **31**(4), Article 15 (2009). doi:10.1145/1516507.1516510.
54. Suomela, J.: Distributed algorithms for edge dominating sets. In: Proc. 29th Annual ACM Symposium on Principles of Distributed Computing (PODC 2010), pp. 365–374. ACM Press (2010). doi:10.1145/1835698.1835783.
55. Suomela, J.: Survey of local algorithms. ACM Computing Surveys **45**(2), 24:1–40 (2013). doi:10.1145/2431211.2431223. http://www.cs.helsinki.fi/local-survey/.
56. Wiese, A., Kranakis, E.: Impact of locality on location aware unit disk graphs. Algorithms **1**, 2–29 (2008). doi:10.3390/a1010002.
57. Wolfram, S.: Statistical mechanics of cellular automata. Reviews of Modern Physics **55**(3), 601–644 (1983). doi:10.1103/RevModPhys.55.601.
58. Yamashita, M., Kameda, T.: Electing a leader when processor identity numbers are not distinct (extended abstract). In: Proc. 3rd International Workshop on Distributed Algorithms (WDAG 1989), *Lecture Notes in Computer Science*, vol. 392, pp. 303–314. Springer (1989). doi:10.1007/3-540-51687-5_52.
59. Yamashita, M., Kameda, T.: Computing functions on asynchronous anonymous networks. Mathematical Systems Theory **29**(4), 331–356 (1996). doi:10.1007/BF01192691.
60. Yamashita, M., Kameda, T.: Computing on anonymous networks: part I—characterizing the solvable cases. IEEE Transactions on Parallel and Distributed Systems **7**(1), 69–89 (1996). doi:10.1109/71.481599.
61. Yamashita, M., Kameda, T.: Computing on anonymous networks: part II—decision and membership problems. IEEE Transactions on Parallel and Distributed Systems **7**(1), 90–96 (1996). doi:10.1109/71.481600.
62. Yamashita, M., Kameda, T.: Leader election problem on networks in which processor identity numbers are not distinct. IEEE Transactions on Parallel and Distributed Systems **10**(9), 878–887 (1999). doi:10.1109/71.798313.