Weak Models of Distributed Computing, with Connections to Modal Logic

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

LH, AK, JV: School of Information Sciences, University of Tampere
MJ, JL, TL, JS: Helsinki Institute for Information Technology HIIT,
Department of Computer Science, University of Helsinki
KL: Department of Mathematics and Statistics, University of Helsinki

ABSTRACT

This work presents a classification of weak models of distributed computing. We focus on deterministic distributed algorithms, and we 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 it 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, e.g., 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, we can use tools from modal logic to study these classes.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems

Keywords

anonymous networks, deterministic distributed algorithms, graph problems, modal logic, port-numbering model

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 Distributed Algorithms

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 of maximum degree at most Δ. A *distributed algorithm* 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,

- $z_0 \colon \{0, 1, \ldots, \Delta\} \to Z$ 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 $\delta(y, \vec{m}) = y$ for any $\vec{m} \in M^{\Delta}$. In other words, a node that has stopped does not send any messages and does not change its state any more.
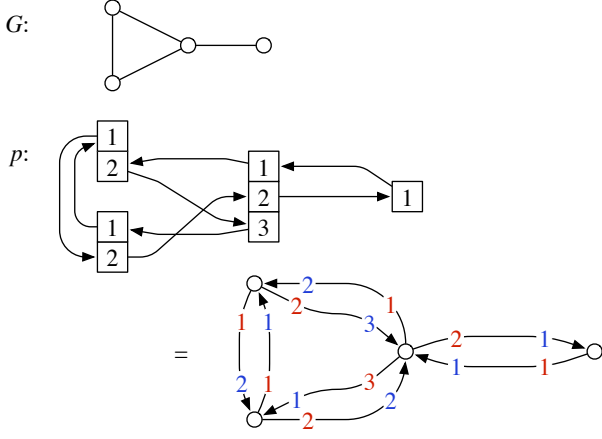
Figure 1: A port numbering p of graph G. Here we present p using two different notations.
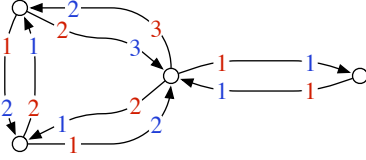

Figure 2: Consistent port numbering.

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

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

We say that p is a *port numbering* of G if $A(p) = A(G)$; see Figure 1. 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, i.e.,

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

See Figure 2 for an example.

1.3 Execution of an Algorithm

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

The state of the system at time $t = 0, 1, \ldots$ 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 $\vec{a}_{t+1}(u)$ of length Δ as follows:

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

That is, 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 $\vec{a}_{t+1}(u) \in M^{\Delta}$. Finally, we define the new state of a node $u \in V$ as follows:

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

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 algorithm $\mathcal{A}$ *solves* a graph problem 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 $T\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \ldots)$ be a family of distributed algorithms. We say that $\mathcal{A}$ *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:

(a) Algorithm $\mathcal{A}_\Delta$ stops in time $T(\Delta, |V|)$ in (G, p).
(b) The output of $\mathcal{A}_\Delta$ is in $\Pi(G)$.

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

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

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 $\vec{a} = (a_1, a_2, \ldots, a_\Delta) \in M^\Delta$ we define

$$\text{set}(\vec{a}) = \{a_1, a_2, \ldots, a_\Delta\},$$
$$\text{multiset}(\vec{a}) = \big\{(m, n) : m \in M, n = |\{i \in [\Delta] : m = a_i\}|\big\}.$$

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

Let Vector be the set of all distributed algorithms, as defined in Section 1.1. We define three subclasses of distributed algorithms, Set $\subseteq$ Multiset $\subseteq$ Vector, and Broadcast $\subseteq$ Vector:

- $\mathcal{A} \in$ Multiset if $\text{multiset}(\vec{a}) = \text{multiset}(\vec{b})$ implies $\delta(x, \vec{a}) = \delta(x, \vec{b})$ for all $x \in Z$,

- $\mathcal{A} \in$ Set if $\text{set}(\vec{a}) = \text{set}(\vec{b})$ implies $\delta(x, \vec{a}) = \delta(x, \vec{b})$ for all $x \in Z$.

- $\mathcal{A} \in$ Broadcast if $\mu(x, i) = \mu(x, j)$ for all $x \in Z$ and $i, j \in [\Delta]$.

Classes Multiset and Set are related to *incoming* messages. Intuitively, an algorithm in class Vector considers a *vector* of incoming messages, while an algorithm in Multiset considers a *multiset* of incoming messages, and an algorithm in Set considers a *set* of incoming messages. Algorithms in Multiset and Set do not have any access to the numbering of incoming ports.

Class Broadcast is related to *outgoing* messages. Intuitively, an algorithm in class Vector constructs a *vector* of outgoing messages, while an algorithm in Broadcast can only *broadcast* the same message to all neighbours. Algorithms in Broadcast do not have any access to the numbering of outgoing ports.

1.6 Problem Classes

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

$\Pi \in \mathsf{VV_c}$: there is an algorithm $\mathcal{A} \in$ Vector that solves Π assuming consistency,

$\Pi \in \mathsf{VV}$: there is an algorithm $\mathcal{A} \in$ Vector that solves Π,

$\Pi \in \mathsf{MV}$: there is an algorithm $\mathcal{A} \in$ Multiset that solves Π,

$\Pi \in \mathsf{SV}$: there is an algorithm $\mathcal{A} \in$ Set that solves Π,

$\Pi \in \mathsf{VB}$: there is an algorithm $\mathcal{A} \in$ Broadcast that solves Π,

$\Pi \in \mathsf{MB}$: there is an algorithm $\mathcal{A} \in$ Multiset $\cap$ Broadcast that solves Π,

$\Pi \in \mathsf{SB}$: there is an algorithm $\mathcal{A} \in$ Set $\cap$ Broadcast that solves Π.

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

$\Pi \in \mathsf{VV_c}(1)$: there is a local algorithm $\mathcal{A} \in$ Vector that solves Π assuming consistency,

$\Pi \in \mathsf{VV}(1)$: there is a local algorithm $\mathcal{A} \in$ Vector that solves Π, etc.

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 Figure 3a. Figure 4 summarises what information is available to an algorithm in each class.
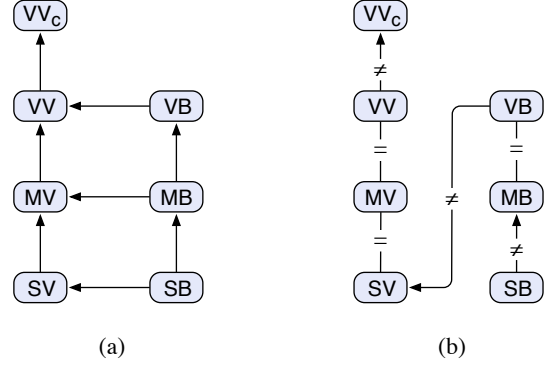

Figure 3: (a) Trivial subset relations between the classes. (b) The linear order identified in this work.
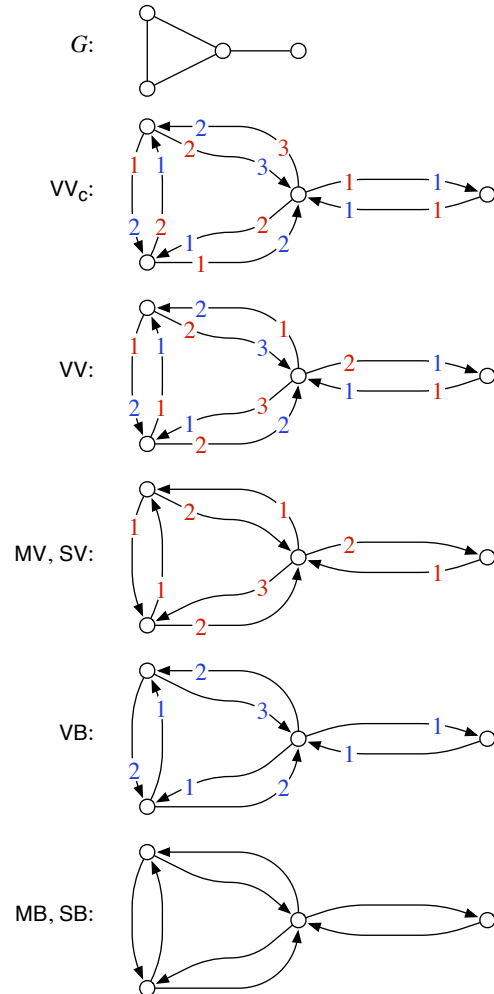

Figure 4: Auxiliary information that is available to a distributed algorithm in each class.

2. CONTRIBUTIONS

This work is a systematic study of the complexity classes 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 Figure 3a. For example, classes VB and SV are seemingly orthogonal, and it would be natural to assume that neither $VB \subseteq SV$ nor $SV \subseteq VB$ holds. However, we show that this is not the case. Unexpectedly, the classes form a linear order (see Figure 3b):

$$SB \subsetneq MB = VB \subsetneq SV = MV = VV \subsetneq VV_c. \qquad (1)$$

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

(1) consistent port numbering (class 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:

$$SB(1) \subsetneq MB(1) = VB(1) \subsetneq SV(1)$$
$$= MV(1) = VV(1) \subsetneq VV_c(1). \qquad (2)$$

The main technical achievement here is proving that $SV = MV$; this together with the ideas of a prior work [3] leads to the linear orders (1) and (2).

Second, we identify a novel connection between distributed computational complexity and modal logic. In particular, the classes $VV_c(1)$, $VV(1)$, $MV(1)$, $SV(1)$, $VB(1)$, $MB(1)$, and $SB(1)$ 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).

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:

(a) *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 [17] model of computation; Peleg [21] calls it the *local* model.

(b) *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. This 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 *quickly* in both of the above models.

3.2 Port-Numbering Model

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. [4], Yamashita and Kameda [25–27], and Boldi and Vigna [9], who studied global functions, leader election problems, spanning trees, and topological properties.

Based on the earlier work, the study of 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. The research direction was initiated by Naor and Stockmeyer [19] 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; see the survey [22] for an overview.

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 [13].

3.3 Weaker Models

While the study of the port-numbering model has been recently revived, it is also time to question whether its role as the the standard model in the study of anonymous networks is justified. First, the definition is somewhat arbitrary—it is not obvious that 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 VV_c, there would be little motivation for pursuing further. However, the recent work related to the vertex cover problem [3] calls for further investigation. It turned out that 2-approximation of vertex cover is a graph problem that is not only in $VV_c(1)$, but also in $MB(1)$—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 [1, 3, 7–9, 11, 16, 20, 23, 24, 28]—see the full version of this work [14] for an overview. 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 [18, 19] made it explicit that the parity of node degrees makes a huge difference in the port-numbering model, and Yamashita and Kameda [26] discussed factors and factorisations in this context.

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, and in particular bisimulation is a very convenient tool that complements the closely related graph-theoretic concept of covering graphs.

3.4 Distributed Algorithms and Modal Logic

Modal logic (see Section 4) has, of course, been applied previously in the context of distributed systems. However, our perspective is a radical departure from the traditional approach.

In the textbook approach, possible worlds are possible states of the (distributed) system and accessibility relations are possible state transitions. We turn this setting upside down.

In our approach, possible worlds correspond to machines and accessibility relations correspond to communication links. Hence a Kripke model is, in essence, an encoding of the structure of a computer network.

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 in modal logic, while a local algorithm in $\mathsf{Multiset} \cap \mathsf{Broadcast}$ corresponds to a formula in *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 $\mathsf{VV_c}(1)$ by a corresponding modal logic, in the spirit of descriptive complexity theory (see Immerman [15]). For each

class there is a modal logic such that every algorithm in the class can be described by a formula in the modal logic, and conversely, each formula defines an algorithm 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 [5] or Blackburn, van Benthem, and Wolter [6] 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 $\mathrm{ML}(\Phi)$-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 $\tau : \Phi \to \mathcal{P}(W)$.

The truth of an $\mathrm{ML}(\Phi)$-formula φ in a state $v \in W$ of a Kripke model $K = (W, R, \tau)$ is defined recursively in the obvious way. For the modal case it holds that

$$K, v \models \Diamond\varphi \quad \text{iff}$$
$$K, w \models \varphi \text{ for some } w \in W \text{ s.t. } (v, w) \in R.$$

The syntax of *graded modal logic* [12], 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:

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

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. Given a set I of indices and a finite set Φ of proposition symbols, the set of $\mathrm{MML}(I, \Phi)$-formulas is defined by the same grammar as for $\mathrm{ML}(\Phi)$ with $\Diamond\varphi$ replaced by $\langle\alpha\rangle\varphi$, for each $\alpha \in I$.

The Kripke models corresponding to the multimodal language $\mathrm{MML}(I, \Phi)$ are of the form

$$K = (W, (R_\alpha)_{\alpha \in I}, \tau),$$

where $R_\alpha \subseteq W^2$ for each $\alpha \in I$. The truth of an $\mathrm{MML}(I, \Phi)$-formula in K is defined in the obvious way.

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* $\mathrm{GMML}(I, \Phi)$.

Let $\mathcal{L}$ be a modal logic and φ an $\mathcal{L}$-formula. The *modal depth* of φ, denoted by $\mathrm{md}(\varphi)$, 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 $\{v \in W : K, v \models \varphi\}$ of the set of states in K; this set is denoted by $\|\varphi\|^K$.

4.2 Bisimulation and Definability

We will now define one of the most important concepts in modal logic, bisimulation. The objective 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 $K = (W, (R_\alpha)_{\alpha \in I}, \tau)$ and $K' = (W', (R'_\alpha)_{\alpha \in I}, \tau')$ 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 $w' \in W'$ such that $(v', w') \in R'_\alpha$ 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 $w \in W$ such that $(v, w) \in R_\alpha$ 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 w.r.t. MML.

FACT 1. *Let $\mathcal{L}$ be one of the modal logics* ML, MML, GML *and* GMML, *and let K and K' be Kripke models, $v \in W$ and $v' \in W'$. If v and v' are bisimilar w.r.t. $\mathcal{L}$, then for all $\mathcal{L}$-formulas φ we have $K, v \models \varphi$ iff $K', v' \models \varphi$.*

4.3 Characterising Constant Time Classes

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:

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

Conversely, given a modal formula φ, the evaluation of its truth in the Kripke model $K(G, p)$ can be done by a local algorithm $\mathcal{A}$.

The general idea of the correspondence between modal logic and distributed algorithms is described in Table 1.

We define four different Kripke models $K_i(G, p)$ for (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 $K_1(G, p)$ are the following:

$$R_{(i,j)} = \big\{ (u, v) \in V \times V : p((v, j)) = (u, i) \big\}$$

for each pair $(i, j) \in [\Delta] \times [\Delta]$. For algorithms in classes below Vector we need alternative accessibility relations with corresponding restrictions on their information about p. For each $i \in [\Delta]$ we define that

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

In addition we define that $R_{(*,*)} = \bigcup_{(i,j) \in [\Delta] \times [\Delta]} R_{(i,j)}$. 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 $\tau(q_i) = \{v \in V : \deg(v) = i\}$.

Table 1: Correspondence between modal logic and distributed algorithms.

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

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

$$K_1(G, p) = (V, (R_\alpha)_{\alpha \in I_{\Delta,1}}, \tau), \quad \text{where} \quad I_{\Delta,1} = [\Delta] \times [\Delta],$$
$$K_2(G, p) = (V, (R_\alpha)_{\alpha \in I_{\Delta,2}}, \tau), \quad \text{where} \quad I_{\Delta,2} = \{*\} \times [\Delta],$$
$$K_3(G, p) = (V, (R_\alpha)_{\alpha \in I_{\Delta,3}}, \tau), \quad \text{where} \quad I_{\Delta,3} = [\Delta] \times \{*\},$$
$$K_4(G, p) = (V, (R_\alpha)_{\alpha \in I_{\Delta,4}}, \tau), \quad \text{where} \quad I_{\Delta,4} = \{(*, *)\}.$$

For each $i \in \{1, 2, 3, 4\}$, we denote the class of all Kripke models of the form $K_i(G, p)$ by $\mathcal{K}_i$. Furthermore, we denote by $\mathcal{K}_0$ the subclass of $\mathcal{K}_1$ consisting of the models $K_1(G, p)$, where p is a consistent port numbering of G.

In order to give a precise formulation of the correspondence between modal logics and the constant time classes of graph problems, we define now the concept of modal formulas solving graph problems. W.l.o.g., 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.

Let $\mathcal{L}$ be a modal logic, let $i \in \{1, 2, 3, 4\}$, and let $\Psi = (\psi_1, \psi_2, \ldots)$ be a sequence of modal formulas such that $\psi_\Delta \in \mathcal{L}(I_{\Delta,i}, \Phi_\Delta)$ for each $\Delta \in \mathbb{N}$. Then Ψ *defines a solution* for a graph problem Π on the class $\mathcal{K}_i$ if for all $\Delta \in \mathbb{N}$, all $G \in \mathcal{F}(\Delta)$, and all port numberings p of G, the subset $\|\psi_\Delta\|^{K_i(G,p)}$ defined by the formula ψ_Δ in the model $K_i(G, p)$ is in set $\Pi(G)$. Furthermore, the sequence Ψ defines a solution for Π on the class $\mathcal{K}_0$, if the condition above with $i = 1$ holds for all *consistent* port numberings p.

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

Let $\mathcal{L}$ be a modal logic, let $i \leq 4$, and let C be a class of graph problems. We say that $\mathcal{L}$ *captures* C on $\mathcal{K}_i$ if the following two conditions hold:

- If $\Psi = (\psi_1, \psi_2, \ldots)$ is a sequence of formulas such that $\psi_\Delta \in \mathcal{L}(I_{\Delta,i}, \Phi_\Delta)$ for all $\Delta \in \mathbb{N}$, then $\Pi_\Psi \in C$.

- For every graph problem $\Pi \in C$ there is a sequence $\Psi = (\psi_1, \psi_2, \ldots)$ of formulas such that $\psi_\Delta \in \mathcal{L}(I_{\Delta,i}, \Phi_\Delta)$ for all $\Delta \in \mathbb{N}$, which defines a solution for Π on $\mathcal{K}_i$.

The main result of this section is that the constant time version of each of the classes VV$_c$, VV, MV, SV, VB, MB,

Table 2: The intended meaning of the subformulas.

Subformulas of ψ_Δ	Algorithm $\mathcal{A}_\Delta$
$\varphi_{\ell,t}$ is true in world v	bit ℓ of local state $x_t(v)$ is 1
$\vartheta_{m,j,t}$ 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 a neighbour to port j

Table 3: Constructing the formula ψ_Δ, given an algorithm $\mathcal{A}_\Delta$.

Recursive definition of the formulas	Execution of $\mathcal{A}_\Delta$
$\varphi_{\ell,0}$: Boolean combination of $q_i \in \Phi_\Delta$	initialisation: $x_0(u) = z_0(\deg(u))$
$\vartheta_{m,j,t+1}$: Boolean combination of $\varphi_{\ell,t}$, $\ell \leq L$	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 $\vec{a}_{t+1}(v)$
$\varphi_{\ell,t+1}$: Boolean combin. of $\varphi_{k,t}$, $k \leq L$, and $\chi_{m,i,j,t+1}$, $m \in M$, $i,j \in [\Delta]$	local computation: $x_{t+1}(v) = \delta(x_t(v), \vec{a}_{t+1}(v))$

and SB is captured by one of the modal logics MML, ML, GMML, and GML on an appropriate class $\mathcal{K}_i$.

THEOREM 1. *(a) MML captures* $\mathsf{VV_c}(1)$ *on* $\mathcal{K}_0$.
(b) MML captures $\mathsf{VV}(1)$ *on* $\mathcal{K}_1$.
(c) GMML captures $\mathsf{MV}(1)$ *on* $\mathcal{K}_2$.
(d) MML captures $\mathsf{SV}(1)$ *on* $\mathcal{K}_2$.
(e) MML captures $\mathsf{VB}(1)$ *on* $\mathcal{K}_3$.
(f) GML captures $\mathsf{MB}(1)$ *on* $\mathcal{K}_4$.
(g) ML captures $\mathsf{SB}(1)$ *on* $\mathcal{K}_4$.

PROOF SKETCH. We describe here the idea behind the proof of (b); other cases are similar.

Assume first that $\Psi = (\psi_1, \psi_2, \ldots)$ is a sequence of formulas with $\psi_\Delta \in \mathrm{MML}(I_{\Delta,1}, \Phi_\Delta)$ for each $\Delta \in \mathbb{N}$. We describe a local algorithm $\mathcal{A}_\Delta \in \mathsf{Vector}$ that simulates the recursive evaluation of the truth of ψ_Δ on a Kripke model $K_1(G, p)$.

The idea is that in the j:th step of computation of the algorithm $\mathcal{A}_\Delta$ each node u sends the truth values of subformulas of ψ_Δ of modal depth $j - 1$ in u to its neighbours. After receiving these truth values from its neighbours each node u computes the values of subformulas of ψ_Δ of modal depth j. Now after $\mathrm{md}(\psi_\Delta)$ computation steps of the algorithm $\mathcal{A}_\Delta$ each node u knows whether ψ_Δ holds in it or not.

For the other direction, assume that Π is a graph problem in $\mathsf{VV}(1)$. Thus, there is a sequence $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \ldots)$ of local algorithms in Vector that solves Π. 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 use a binary encoding for the states $x_t(v)$ of nodes in the computation of $\mathcal{A}_\Delta$; let L be the length of this encoding. We assume w.l.o.g. that the output of the algorithm is the first bit of x_T. We will build a formula $\psi_\Delta \in \mathrm{MML}(I_{\Delta,1}, \Phi_\Delta)$ that simulates $\mathcal{A}_\Delta$ from the following subformulas:

- $\varphi_{\ell,t}$ for $\ell \leq L$ and $t \leq T$,
- $\vartheta_{m,j,t}$ 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 is given in Table 2, and their recursive definitions are given in Table 3.

Given an input (G, p) to the algorithm $\mathcal{A}_\Delta$, the output on a node v is 1 if and only if

$$v \in \|\varphi_{0,T}\|^{K_1(G,p)}.$$

Thus, defining $\psi_\Delta := \varphi_{0,T}$ we get $\|\psi_\Delta\|^{K_1(G,p)} \in \Pi(G)$ for all $G \in \mathcal{F}(\Delta)$ and all port numberings p of G. Hence we conclude that the sequence $\Psi = (\psi_1, \psi_2, \ldots)$ defines a solution to Π.

As an additional remark, we note that $\mathrm{md}(\psi_\Delta)$ is equal to the running time T of $\mathcal{A}_\Delta$. $\square$

5. RELATIONS BETWEEN THE CLASSES

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

5.1 Equality MV = 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 $\mathcal{A}_1 \in \mathsf{Multiset}$ that solves Π in time T. Then there is an algorithm $\mathcal{A}_2 \in \mathsf{Set}$ that solves Π in time $T + O(\Delta)$.*

To prove Theorem 2, we define the following local algorithm $\mathcal{A}'_\Delta \in \mathsf{Set}$. Each node v constructs two sequences, $\beta_t(v)$ and $B_t(v)$ for $t = 0, 1, \ldots, 2\Delta$. Before the first round, each node v sets $\beta_0(v) = \emptyset$ and $B_0(v) = \emptyset$. Then in round $t = 1, 2, \ldots, 2\Delta$, 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 $\mathcal{A}'_\Delta$ 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

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

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 $\{u, v\} \in E$.

191

We begin with the following technical lemma. We emphasise that in the statement of the lemma, the sets $\{u, w\}$, $\{v_1, v_2, \ldots, v_k\}$, and $\{v_1', v_2', \ldots, v_{k+1}'\}$ may intersect—for example, we may have $u = v_1 = v_1'$.

LEMMA 1. *Suppose that all of the the following hold for some $t \geq 4$:*

(a) Nodes u and w are two distinct neighbours of v.
(b) Nodes $v_1, v_2, \ldots, v_k$ are k distinct neighbours of v.
(c) We have $\beta_t(u) = \beta_t(w) = \beta_t(v_i)$ for all $i = 1, 2, \ldots, k$.
(d) We have $\deg(u) = \deg(w)$ and $\pi(u, v) = \pi(w, v)$.

In particular, node v receives the same message from u and w in round t. Then the following holds:

(e) There are $k + 1$ distinct neighbours of v, denoted by $v_1', v_2', \ldots, v_{k+1}'$, such that

$$\beta_{t-2}(u) = \beta_{t-2}(w) = \beta_{t-2}(v_i')$$

for all $i = 1, 2, \ldots, k + 1$.

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 $i = 1, 2, \ldots, k$, node v_i receives the message

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

from v in round $t - 1$. By assumption, we have $\beta_t(v_i) = \beta_t(v_j)$ for all i, 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, j.

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

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

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

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

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 $i = 1, 2, \ldots, k$, 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 $(\beta_{t-2}(u), \cdot, \cdot)$. Hence v has at least $k + 1$ distinct neighbours v_i' with $\beta_{t-2}(u) = \beta_{t-2}(v_i')$. $\square$

LEMMA 2. *If a node v receives the same message from its neighbours u and w in round $2t$, and v has k distinct neighbours $v_1, v_2, \ldots, v_k$ such that $\beta_{2t}(v_i) = \beta_{2t}(u) = \beta_{2t}(w)$ for all $i = 1, 2, \ldots, k$, then v has at least $t + k - 1$ neighbours.*

PROOF. The proof is by induction on t. The base case $t = 1$ is trivial. For the inductive step, apply Lemma 1. $\square$

Hence if a node v has two neighbours u and w with $m_{2t}(u, v) = m_{2t}(w, v)$, node v has at least $t + 1$ neighbours. As the maximum degree of graph G is at most Δ, we know that $m_{2\Delta}(u, v) \neq m_{2\Delta}(w, v)$ whenever u and w are two distinct neighbours of v. In particular,

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

Once we have finished running $\mathcal{A}_\Delta'$, which takes $O(\Delta)$ time, we can *simulate* the execution of $\mathcal{A}_1 \in \mathsf{Multiset}$ with an algorithm $\mathcal{A}_2 \in \mathsf{Set}$ as follows: if a node u in the execution of $\mathcal{A}_1$ sends the message a to port i, algorithm $\mathcal{A}_2$ sends the message

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

to port i. Now all messages received by a node are distinct. Hence given the set of messages received by a node v in $\mathcal{A}_2$, we can reconstruct the multiset of messages received by v in $\mathcal{A}_1$. This concludes the proof of Theorem 2. $\square$

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

PROOF. Immediate from Theorem 2. $\square$

5.2 Equalities VB = MB and VV = MV

The following theorems are implicit in prior work [3]; we give more detailed proofs in the full version of this work [14]. The basic idea is that $\mathcal{A}_2$ 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 $\mathcal{A}_1$ 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 $\mathcal{A}_1 \in \mathsf{Vector}$ that solves Π in time T. Then there is an algorithm $\mathcal{A}_2 \in \mathsf{Multiset}$ that solves Π in time T.*

THEOREM 4. *Let Π be a graph problem and let $T \colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. Assume that there is an algorithm $\mathcal{A}_1 \in \mathsf{Broadcast}$ that solves Π in time T. Then there is an algorithm $\mathcal{A}_2 \in \mathsf{Multiset} \cap \mathsf{Broadcast}$ that solves Π in time T.*

COROLLARY 2. *We have $\mathsf{VB} = \mathsf{MB}$, $\mathsf{VB}(1) = \mathsf{MB}(1)$, $\mathsf{VV} = \mathsf{MV}$, and $\mathsf{VV}(1) = \mathsf{MV}(1)$.*

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 1 and 2 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.

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:

(a) $G = (V, E)$ is a k-star for a $k > 1$. That is, $V = \{c, v_1, v_2, \ldots, v_k\}$ and $E = \{\{c, v_i\} : i = 1, 2, \ldots, k\}$. 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 $i \neq j$.

(b) $G = (V, E)$ 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 $\mathcal{A}_1 \in \mathsf{Set}$ that solves 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 $\{1\}$.

It is equally easy to see that an algorithm $\mathcal{A}_2 \in \mathsf{Broadcast}$ cannot solve the problem. Let G be a k-star, and let p be any port numbering of G. Now in the Kripke model $K_3(G, p)$, all leaf nodes are bisimilar w.r.t. MML. Equivalently, in any execution of $\mathcal{A}_2$ on (G, p), all leafs are in the same state at each time step. $\square$

COROLLARY 3. *We have* $\mathsf{VB} \neq \mathsf{SV}$ *and* $\mathsf{VB}(1) \neq \mathsf{SV}(1)$.

PROOF. Follows from Theorem 5. $\square$

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.
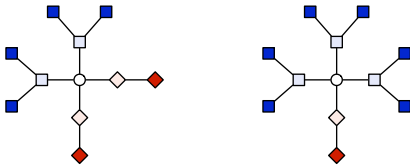
More precisely, we can partition the nodes in five equivalence classes (indicated with the shading and shapes in the above illustration), and the nodes in the same equivalence class are bisimilar in the Kripke model $K_4(G, p)$ w.r.t. ML; recall that the model is independent of the choice of the port numbering p. Equivalently, in the execution of any algorithm $\mathcal{A} \in \mathsf{Set} \cap \mathsf{Broadcast}$, all nodes in the same equivalence class have the same state and hence produce the same output. $\square$

COROLLARY 4. *We have* $\mathsf{SB} \neq \mathsf{MB}$ *and* $\mathsf{SB}(1) \neq \mathsf{MB}(1)$.

PROOF. Follows from Theorem 6. $\square$

Finally, to separate VV and $\mathsf{VV_c}$, we can use the construction of Figure 5—there a consistent port numbering necessarily breaks symmetry, while an inconsistent port numbering may be symmetric.
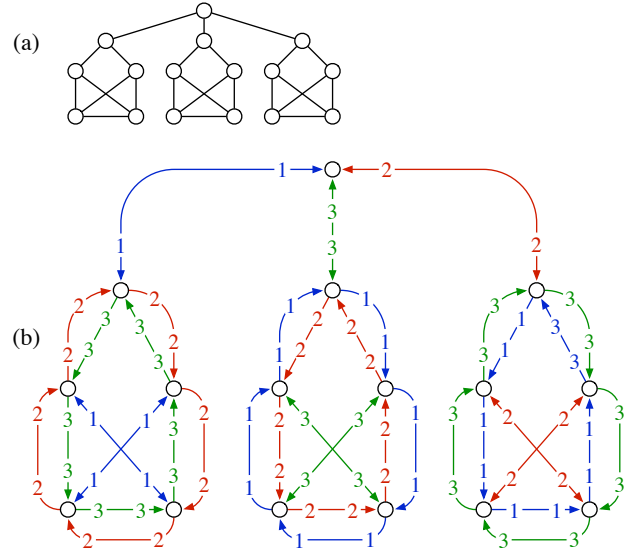
Figure 5: (a) A 3-regular graph G that does not have a 1-factor [10, Figure 5.10]. (b) A port numbering of G.

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

PROOF. We define Π so that for the graph $G = (V, E)$ illustrated in Figure 5a, $\Pi(G)$ 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 other graphs $\Pi(G)$ 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 be the graph of Figure 5a. Graph G is 3-regular. It can be verified that G does not have any 1-factor (perfect matching); hence it does not have any 2-factor, either. Now the arguments of Yamashita and Kameda [26, Section 5.1] imply that in any consistent port numbering of G, there are two nodes with distinct local views; moreover, it suffices to focus on radius-T local views for a (small) constant T. We define that the *type* of a node is an encoding of its radius-T local view (as an integer, using some canonical encoding). Now in G we have always nodes of at least two different types. Hence we can solve the problem as follows: a node outputs 1 iff it is adjacent to a node with a strictly smaller type. This produces a non-constant output in G, for any port numbering p, and it stops in constant time in any graph; hence it is a local algorithm that solves Π.

To see that an algorithm $\mathcal{A} \in \mathsf{Vector}$ cannot solve the problem without assuming consistency, consider the port numbering p in Figure 5b. If we reverse the arrows, the same figure can be interpreted as an illustration of the Kripke model $K_1(G, p)$, and it is straightforward to verify that all nodes are bisimilar w.r.t. MML. $\square$

COROLLARY 5. *We have* $\mathsf{VV_c} \neq \mathsf{VV}$ *and* $\mathsf{VV_c}(1) \neq \mathsf{VV}(1)$.

PROOF. Follows from Theorem 7. $\square$

In summary, we have established that the classes we have studied form a linear order of length four—see (1) and (2) in Section 2. As a corollary of (2) and Theorem 1, we can make, for example, the following observations.

Corollary 6. *(a)* MML *captures the same class of problems on $\mathcal{K}_1$ and $\mathcal{K}_2$.*

(b) Both MML *and* GMML *capture the same class of problems on $\mathcal{K}_2$.*

(c) The class of problems captured by MML *becomes strictly smaller if we replace $\mathcal{K}_2$ with $\mathcal{K}_3$.*

(d) MML *on $\mathcal{K}_3$ captures the same class of problems as* GML *on $\mathcal{K}_4$.*

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 separate the classes. Another challenge is to come up with *decision problems* that separate the classes.

6. ACKNOWLEDGEMENTS

The full version of this work is available online [14].

7. REFERENCES

[1] Yehuda Afek, Noga Alon, Ziv Bar-Joseph, Alejandro Cornejo, Bernhard Haeupler, and Fabian Kuhn. Beeping a maximal independent set. In *Proc. DISC 2011*, volume 6950 of *LNCS*, pages 32–50. Springer, 2011.

[2] Dana Angluin. Local and global properties in networks of processors. In *Proc. STOC 1980*, pages 82–93. ACM Press, 1980.

[3] Matti Åstrand and Jukka Suomela. Fast distributed approximation algorithms for vertex cover and set cover in anonymous networks. In *Proc. SPAA 2010*, pages 294–302. ACM Press, 2010.

[4] Hagit Attiya, Marc Snir, and Manfred K. Warmuth. Computing on an anonymous ring. *J. ACM*, 35(4):845–875, 1988.

[5] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2001.

[6] Patrick Blackburn, Johan van Benthem, and Frank Wolter, editors. *Handbook of Modal Logic*, volume 3 of *Studies in Logic and Practical Reasoning*. Elsevier, 2007.

[7] Paolo Boldi, Shella Shammah, Sebastiano Vigna, Bruno Codenotti, Peter Gemmell, and Janos Simon. Symmetry breaking in anonymous networks: characterizations. In *Proc. ISTCS 1996*, pages 16–26. IEEE, 1996.

[8] Paolo Boldi and Sebastiano Vigna. Computing vector functions on anonymous networks. In *Proc. SIROCCO 1997*, pages 201–214. Carleton Scientific, 1997.

[9] Paolo Boldi and Sebastiano Vigna. An effective characterization of computability in anonymous networks. In *Proc. DISC 2001*, volume 2180 of *LNCS*, pages 33–47. Springer, 2001.

[10] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. North-Holland, New York, 1976.

[11] Krzysztof Diks, Evangelos Kranakis, Adam Malinowski, and Andrzej Pelc. Anonymous wireless rings. *Theoret. Comput. Sci.*, 145(1–2):95–109, 1995.

[12] Kit Fine. In so many possible worlds. *Notre Dame J. Formal Logic*, 13(4):516–520, 1972.

[13] Mika Göös, Juho Hirvonen, and Jukka Suomela. Lower bounds for local approximation. In *Proc. PODC 2012*. ACM Press, 2012.

[14] Lauri Hella, Matti Järvisalo, Antti Kuusisto, Juhana Laurinharju, Tuomo Lempiäinen, Kerkko Luosto, Jukka Suomela, and Jonni Virtema. Weak models of distributed computing, with connections to modal logic, 2012. Manuscript, arXiv:1205.2051 [cs.DC].

[15] Neil Immerman. *Descriptive Complexity*. Graduate Texts in Computer Science. Springer, 1999.

[16] Fabian Kuhn and Roger Wattenhofer. On the complexity of distributed graph coloring. In *Proc. PODC 2006*, pages 7–15. ACM Press, 2006.

[17] Nathan Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992.

[18] Alain Mayer, Moni Naor, and Larry Stockmeyer. Local computations on static and dynamic graphs. In *Proc. ISTCS 1995*, pages 268–278. IEEE, 1995.

[19] Moni Naor and Larry Stockmeyer. What can be computed locally? *SIAM J. Comput.*, 24(6):1259–1277, 1995.

[20] Nancy Norris. Computing functions on partially wireless networks. In *Proc. SIROCCO 1995*, pages 53–64. Carleton Scientific, 1996.

[21] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM Monographs on Discrete Mathematics and Applications. SIAM, 2000.

[22] Jukka Suomela. Survey of local algorithms. *ACM Comput. Surveys*. To appear.

[23] Stephen Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55(3):601–644, 1983.

[24] Masafumi Yamashita and Tsunehiko Kameda. Electing a leader when processor identity numbers are not distinct (extended abstract). In *Proc. WDAG 1989*, volume 392 of *LNCS*, pages 303–314. Springer, 1989.

[25] Masafumi Yamashita and Tsunehiko Kameda. Computing functions on asynchronous anonymous networks. *Mathematical Systems Theory*, 29(4):331–356, 1996.

[26] Masafumi Yamashita and Tsunehiko Kameda. Computing on anonymous networks: Part I—characterizing the solvable cases. *IEEE Trans. Parallel Distrib. Systems*, 7(1):69–89, 1996.

[27] Masafumi Yamashita and Tsunehiko Kameda. Computing on anonymous networks: Part II—decision and membership problems. *IEEE Trans. Parallel Distrib. Systems*, 7(1):90–96, 1996.

[28] Masafumi Yamashita and Tsunehiko Kameda. Leader election problem on networks in which processor identity numbers are not distinct. *IEEE Trans. Parallel Distrib. Systems*, 10(9):878–887, 1999.