

# Locality in Online Algorithms

Maciej Pacut  

Faculty of Computer Science, University of Vienna, Austria

Mahmoud Parham  



Faculty of Computer Science, University of Vienna, Austria

Joel Rybicki  

IST Austria, Austria

Stefan Schmid  

Faculty of Computer Science, University of Vienna, Austria

Jukka Suomela  

Aalto University, Finland

Aleksandr Tereshchenko 

Aalto University, Finland

---

## Abstract

Online algorithms make decisions based on past inputs. In general, the decision may depend on the entire history of inputs. If many computers run the same online algorithm with the same input stream but are started at different times, they do not necessarily make consistent decisions.

In this work we introduce *time-local online algorithms*. These are online algorithms where the output at a given time only depends on  $T = O(1)$  latest inputs. The use of (deterministic) time-local algorithms in a distributed setting automatically leads to globally consistent decisions.

Our key observation is that time-local online algorithms (in which the output at a given time only depends on local inputs in the temporal dimension) are closely connected to *local distributed graph algorithms* (in which the output of a given node only depends on local inputs in the spatial dimension). This makes it possible to interpret prior work on distributed graph algorithms from the perspective of online algorithms.

We describe an *algorithm synthesis method* that one can use to design optimal time-local online algorithms for small values of  $T$ . We demonstrate the power of the technique in the context of a variant of the *online file migration problem*, and show that e.g. for two nodes and unit migration costs there exists a 3-competitive time-local algorithm with horizon  $T = 4$ , while no deterministic online algorithm (in the classic sense) can do better. We also derive upper and lower bounds for a more general version of the problem; we show that there is a 6-competitive deterministic time-local algorithm and a 2.62-competitive randomized time-local algorithm for any migration cost  $\alpha \geq 1$ .

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Online algorithms; Theory of computation  $\rightarrow$  Distributed computing models

**Keywords and phrases** Online algorithms, distributed graph algorithms, locality

**Digital Object Identifier** 10.4230/LIPIcs...

**Acknowledgements** Research supported by the European Research Council (ERC) grant agreement No. 864228, Horizon 2020, 2020-2025 and the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 840605.

## 1 Introduction

Online algorithms [13] make decisions based on past inputs, with the goal of being competitive against an algorithm that sees also future inputs. In this work, we introduce *time-local online algorithms*; these are online algorithms in which the output at any given time is a function of only  $T$  latest inputs (instead of the full history of past inputs).



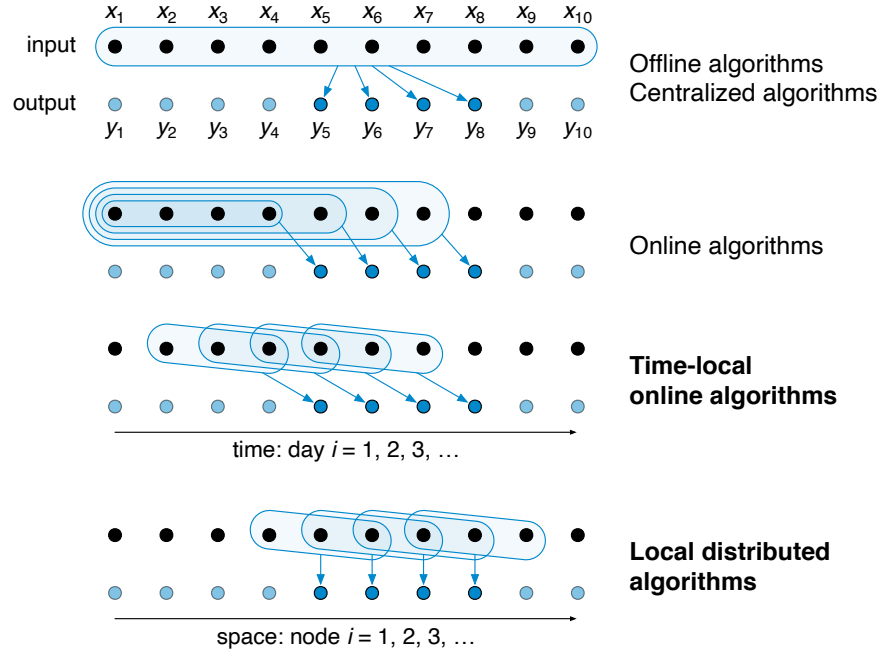
© Maciej Pacut, Mahmoud Parham, Joel Rybicki, Stefan Schmid, Jukka Suomela, and Aleksandr Tereshchenko;

licensed under Creative Commons License CC-BY 4.0



Leibniz International Proceedings in Informatics

LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** Local decision-making in time vs. space dimensions.

Our main observation is that time-local online algorithms are closely connected to *local distributed graph algorithms*: distributed algorithms make decisions based on the *local information in the spatial dimension*, while time-local online algorithms make decisions based on the *local information in the temporal dimension*; see Figure 1. We formalize this connection, and show how we can directly use the tools developed to study distributed approximability of graph optimization problems to prove upper and lower bounds on the competitive ratio achieved with time-local online algorithms. Moreover, we show how to use *computational techniques* to synthesize optimal time-local algorithms.

## 1.1 Motivation

Time-local online algorithms have some attractive properties. Let us give two examples.

**Fault-Tolerant Distributed Decision.** Consider a setting in which many geographically distributed computers need to make *consistent* decisions. All computers can observe the same input stream, and each day each of them has to announce its own decision.

If all computers are started at the same time, we can take any deterministic online algorithm and let each computer run its own copy of the algorithm. However, this approach does not *tolerate failures*: if a computer crashes and is restarted, the local state of the algorithm is lost, and as the decisions may depend in general on the full history of inputs, it will no longer make consistent decisions with the others.

One solution is to run a *consensus* protocol [38], with which the computers can ensure that they all agree on the same decision (or on the same local state). However, this introduces overhead, and is only applicable if we can assume that not too many computers are faulty; for example, with Byzantine failures consensus is possible only if fewer than  $1/3$  of the computers are faulty [38].

Deterministic time-local online algorithms provide automatically the guarantee that all computers will make consistent decisions. The system will tolerate an arbitrary number of failures and ensure that the computers will also recover from transient faults, i.e., it is *self-stabilizing* [24, 25]: in  $T$  steps since the latest failure, all computers will deterministically make consistent decisions, without any communication.

**Random Access to the Decision History.** A second benefit of time-local online algorithms is that they make it possible to efficiently access any past decision, with zero additional storage beyond the storage of the input stream. To look up a decision at any given time  $i$ , it is enough to look up the inputs at the  $T$  previous time points and apply the deterministic time-local algorithm. With classic online algorithms one would have to either store the decision, store the local state, or re-run the entire algorithm up to point  $i$ .

## 1.2 Online Problems and Algorithms

**Online Problems as Request-Answer Games.** Online problems are often formalized as request-answer games [13, Ch. 7] that consist of an input set  $X$  (requests), an output set  $Y$  (answers), and an infinite sequence  $(f_n)_{n \geq 1}$  of cost functions

$$f_n: X^n \times Y^n \rightarrow \mathbb{R} \cup \{\infty\} \text{ for each } n \in \mathbb{N}.$$

The *optimal offline cost* of an input sequence  $\mathbf{x} \in X^n$  is  $\text{OPT}(\mathbf{x}) = \min\{f_n(\mathbf{x}, \mathbf{y}) : \mathbf{y} \in Y^n\}$ .

**Classic Online Algorithms.** An online algorithm  $A$  in the classic sense, i.e., an algorithm that has access to all past inputs, can be defined as a sequence  $(A_i)_{i \geq 1}$  of functions  $A_i: X^{i-1} \rightarrow Y$ . The output  $\mathbf{y} = A(\mathbf{x})$  of the algorithm on input  $\mathbf{x} \in X^n$  is given by

$$y_i = A_i(x_1, \dots, x_{i-1}) \text{ for each } 1 \leq i \leq n.$$

The quality of an online algorithm is measured by comparing the cost of its output against the optimal offline cost. An algorithm is said to be  $c$ -competitive (have a *competitive ratio*  $c$ ) if on any input sequence  $\mathbf{x} \in X^n$  its output  $\mathbf{y} = A(\mathbf{x})$  satisfies  $f_n(\mathbf{x}, \mathbf{y}) \leq c \cdot \text{OPT}(\mathbf{x}) + d$  for a fixed constant  $d$ . We say that an algorithm is *strictly  $c$ -competitive* if additionally  $d = 0$ .

**Time-Local Algorithms.** We now start with a simplified definition of (deterministic) time-local algorithms; the general definitions are given later. Let  $T \in \mathbb{N}$  be a constant. A time-local algorithm that has access to  $T$  latest inputs is given by a sequence of maps  $(A_i)_{i \geq 1}$  of the form  $A_i: (X \cup \{\perp\})^T \rightarrow Y$ , where  $\perp \notin X$ . The output of the algorithm is given by

$$y_i = A_i(x_{i-T}, \dots, x_{i-1}) \text{ for each } 1 \leq i \leq n,$$

where we let  $x_i = \perp$  be placeholder values for  $i < 1$ . We say that the algorithm is *unclocked* if all maps  $A_i$  are equal and otherwise it is *clocked*. That is, in the latter case the  $i$ th decision  $y_i$  made by the algorithm may depend on the current time step  $i$ .

**The Online File Migration.** As our running example, we consider a simple variant of the *online file migration problem*: we are given a network, modeled as undirected graph with two nodes, and an indivisible shared resource, a *file*, initially stored at one of the nodes. Requests to access the file arrive from nodes of the network over time, and their cost is the distance from the requesting node to the file, i.e., 0 if the file is collocated with the request,

and 1 otherwise. After serving a request, we may decide to migrate the file to a different node of the network, paying  $\alpha$  units of migration cost for some parameter  $\alpha \geq 0$  (usually  $\alpha \geq 1$ ). Our goal is to minimize the total cost of accesses and migrations. In the classic setting, for  $\alpha \geq 1$ , there exists a 3-competitive deterministic algorithm for the problem [14], and no deterministic algorithm can do better [11]. However, randomized algorithms can beat this bound: there exists a  $(1 + \phi)$ -competitive randomized algorithm against the oblivious adversary [44], where  $\phi \approx 1.62$  is the golden ratio.

### 1.3 Contributions

In this work, we initiate the study of temporal locality in online algorithms. We give a series of results and techniques that illustrate different aspects of time-local algorithms.

**Introducing Time-Local Online Algorithms.** We formalize the new notion of temporal locality in online algorithms. We investigate the power of time-local online algorithms by focusing on two basic models, unclocked and clocked. We formally introduce the deterministic variants of these models in Section 3, and randomized variants of the models in Appendix C. We show that clocked algorithms are often strictly more powerful than unclocked algorithms, but more surprisingly, unclocked time-local algorithms can sometimes be as competitive as classic (non-local) online algorithms.

**Temporal vs. Spatial Locality: Online Algorithms and Distributed Computing Meet.** We establish that time-local online algorithms are equivalent to local distributed graph algorithms (see Appendix A). This bridges the study of *spatial locality* in distributed computing and *temporal locality* studied in online algorithms: impossibility results for distributed graph algorithms imply lower bounds for *time-local online algorithms*, and vice versa. We utilize this connection to establish impossibility results for e.g. simple variants of online load balancing.

**The Power of Clocked Algorithms.** We give a general result showing that the *clocked* time-local algorithms are often powerful: for a large class of online problems, classic (full-history) online algorithms can be turned into *clocked* time-local algorithms with negligible overhead to the competitive ratio (see Appendix B). In addition, we show that not all online problems admit competitive clocked time-local algorithms despite having competitive full-history algorithms.

**Synthesis of Time-Local Online Algorithms.** Again by leveraging the connection to local graph algorithms, we describe and implement a novel *algorithm synthesis method* that allows us to automate the design of *optimal* unclocked time-local algorithms (see Appendix D). Specifically, the synthesis task can be formulated as a certain weighted optimization problem in dual de Bruijn graphs.

For our case study problem of online file migration, we synthesize optimal deterministic algorithms for small values of  $T$  and a large range of  $\alpha$ . For example, we show that for unit costs ( $\alpha = 1$ ) there exists a 3-competitive time-local algorithm with  $T = 4$ , which is the best competitive ratio achieved by *any* deterministic online algorithm [11]. Moreover, we describe how to extend our synthesis framework to obtain efficient randomized algorithms as well.

**The Impact of Locality: A Case Study.** Finally, we investigate analytically how the length  $T$  of the visible input horizon influences the quality of solutions given by time-local online algorithms. For the online file migration problem, we show the following (see Appendix E):

1. There is no randomized time-local algorithm that has a competitive ratio smaller than  $2\alpha/T$  against an oblivious offline adversary, and this holds also for clocked algorithms. This confirms the intuition that the more limited the access to input history is, the worse the competitive ratio gets.
  2. There is a deterministic, unclocked time-local algorithm that is 6-competitive for some  $T = \Theta(\alpha)$ . This shows that deterministic time-local algorithms can be competitive even without clock.
  3. Well-known randomized algorithms can be adapted to the clocked time-local setting: there is a randomized, clocked time-local algorithm that is 2.62-competitive for some  $T = \Theta(\alpha)$ .
- These results illustrate how the style of arguments and techniques used in the analysis of time-local online algorithms differs from the classic online algorithms setting.

## 2 Related Work

**Restricted Models of Online Algorithms.** To our best knowledge, *temporal locality* of online algorithms has not been systematically studied. However, other restricted forms of online algorithms have received some attention. For example, Chrobak [19] introduced the notion of *memoryless* online algorithms: the answer to the current request can only depend on the current configuration instead of being an arbitrary function of the entire past history as in general online algorithms. In particular, memoryless online algorithms can be synthesized using a fixed point approach. However, memoryless online algorithms differ from time-local algorithms, as memoryless algorithms have access to the previous configuration of the algorithm instead of last  $T$  inputs.

Ben-David et al. [8] investigated local online problems within the request-answer game framework of online algorithms. However, their notion of locality applies to the *cost functions* defining the online problem instead of algorithms solving them: the cost of a solution cannot depend too much on past inputs. In this work, to avoid confusion, we call these games *bounded monotone minimization games*, and show that if a problem in this class admits a competitive online algorithm, then any such algorithm can be converted into a competitive clocked time-local algorithm. Moreover, we introduce an alternative characterization of local games, *local optimization games*, which are different from bounded monotone minimization games, but they have a natural interpretation as distributed optimization problems on paths. We give a general synthesis method to automate the design of optimal time-local algorithms for local optimization problems.

**Synthesis of Online Algorithms.** As mentioned, already the early work on online algorithms considered synthesis in the context of memoryless algorithms [20]. Computer-aided design techniques have also been used to design optimal online algorithms for specific problems. Coppersmith et al. [22] studied the design and analysis of randomized online algorithms for  $k$ -server problems, metrical task systems and a class of graph games; they show that algorithm synthesis is equivalent the synthesis of random walks on graphs. For a variant of the *online knapsack* problem [31], Horiyama, Iwama and Kawahara [29] obtained an optimal algorithm by using a problem-specific finite automaton and solving a set of inequalities for each of its states. More recently, the synthesis of optimal algorithms for preemptive variants of *online scheduling* [7] was reduced to a two-player graph game [18, 37].

**Synthesis of Local Algorithms.** Synthesis of distributed graph algorithms has a long history, mostly focusing on so-called locally checkable labeling (LCL) problems in the LOCAL model of distributed computing [4, 5, 15, 17, 28, 40]. In their foundational work, Naor and Stockmeyer [36] showed that it is undecidable to determine whether an LCL problem admits a local algorithm in general, but it is decidable for unlabeled directed paths and cycles. Balliu et al. [4] showed that determining the distributed round complexity of LCL problems on paths and cycles with inputs is decidable, but PSPACE-hard. Moreover, when restricted to LCL problems with binary inputs, there is a simple synthesis procedure [5]. Recently, Chang et al. [17] showed that synthesis in unlabeled paths, cycles and rooted trees can be done efficiently.

Beyond decidability results, synthesis has also been applied in practice to obtain optimal local algorithms. Rybicki and Suomela [40] showed how to synthesize optimal distributed coloring algorithms on directed paths and cycles. Brandt et al. [15] gave a technique for synthesizing efficient distributed algorithms in 2-dimensional toroidal grids, but showed that in general determining the complexity of an LCL problem is undecidable in grids. Similarly to our work, Hirvonen et al. [28] considered the synthesis of *optimization problems*. They gave a method for synthesizing randomized algorithms for the max cut problem in triangle-free regular graphs.

Our work identifies the connection between temporal locality online algorithms and spatial locality in distributed algorithms. As we show in this work, time-local online algorithms can be seen as local graph algorithms on directed paths. However, formally the computational power of the LOCAL model resides between unclocked and clocked time-local models we study in this work. Thus, decidability results and synthesis techniques do not directly carry over to the time-local online algorithms setting.

## 3 Models and Formalism

### 3.1 Local Optimization Problems

We now introduce the family of local online problems that we study in this work. The definition is somewhat technical, but the basic idea is simple: at each time step  $i$ , the cost (or utility) of our decision  $y_i$  is defined to be some function of the current input  $x_i$  and up to  $r = O(1)$  previous inputs and outputs.

This formalism has several attractive features. First, it is flexible enough to e.g. define online problems in which we reward correct decisions (e.g. whenever we predict correctly  $y_i = x_i$ , we get some profit), we penalize costly moves (e.g. whenever we change our mind and switch to a new output  $y_i \neq y_{i-1}$ , we get some penalty), and we prevent invalid choices (e.g. by defining infinite penalties for decisions that are not compatible with the previous inputs and/or previous decisions). Second, this formalism can capture problems that are relevant in distributed graph algorithms (e.g.  $x_i$  represents the weight of node  $i$  along a path,  $y_i$  indicates which nodes are selected, and we pay  $x_i$  whenever we select a node). Finally, this family of problems is amenable to automated algorithm synthesis, as we will later see.

We will now present the formal definition, and then give several examples of different kinds of problems, both from the areas of online and distributed graph algorithms.

**Formalism.** A *local optimization problem* is a tuple  $\Pi = (X, Y, r, v, \text{aggr}, \text{obj})$ , where

- $X$  is the *set of inputs*,
- $Y$  is the *set of outputs*,
- $r \in \mathbb{N}$  is the *horizon*,

236 ■  $v: X^{r+1} \times Y^{r+1} \rightarrow \mathbb{R} \cup \{-\infty, +\infty\}$  is the *value function*,

237 ■  $\text{aggr} \in \{\text{sum}, \text{min}, \text{max}\}$  is the *aggregation function*,

238 ■  $\text{obj} \in \{\text{min}, \text{max}\}$  is the *objective*.

239 The input for the problem  $\Pi$  is a sequence  $\mathbf{x} = (x_1, x_2, \dots, x_n) \in X^n$  and a solution is  
 240 a sequence  $\mathbf{y} = (y_1, y_2, \dots, y_n) \in Y^n$ . For convenience, we will use placeholder values  
 241  $x_i = y_i = \perp$  for  $i < 1$  and  $i > n$ . With each index, we associate a *value*  $u_i(\mathbf{x}, \mathbf{y})$  defined as

$$242 \quad u_i(\mathbf{x}, \mathbf{y}) = v(x_{i-r}, \dots, x_i, y_{i-r}, \dots, y_i).$$

243 Finally, we apply the aggregation function  $\text{aggr}$  to values  $u_i$  to determine the value  $u(\mathbf{x}, \mathbf{y})$   
 244 of the solution. That is, if the aggregation function is sum, the cost function is given by

$$245 \quad f_n(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^n u_i(\mathbf{x}, \mathbf{y}).$$

246 For example, if the objective is min, the task in  $\Pi$  is to find a solution  $\mathbf{y}$  that minimizes  
 247  $u(\mathbf{x}, \mathbf{y})$  for a given input  $\mathbf{x}$ , and so on. Note that  $X$ ,  $Y$  and  $(f_n)_{n \geq 1}$  define a request-answer  
 248 game. It is common to consider strict competitive ratios when the aggregation function is  
 249 max and the objective is min, see e.g. [3].

250 **Shorthand Notation.** In general, the value function  $v$  is a function with  $2(r+1)$  arguments.  
 251 However, it is often more convenient to represent  $v$  as a function that takes one matrix with  
 252 two rows and  $r+1$  columns and use “.” to denote irrelevant parameters, e.g.

$$253 \quad v\left(\begin{smallmatrix} \cdot & \cdot & c \\ d & e & \cdot \end{smallmatrix}\right) = \gamma$$

254 is equivalent to saying that  $v(a, b, c, d, e, f) = \gamma$  for all  $a, b \in X$  and  $f \in Y$ .

255 **Examples of Online Problems.** Let us first see how to encode typical online problems in  
 256 our formalism. We start with a highly simplified version of the *online file migration problem*,  
 257 a.k.a. *online page migration* [11]. This will be the running example that we use throughout.

258 ► **Example 3.1** (online file migration). The problem is modeled so that input  $x_i \in X = \{0, 1\}$   
 259 represents access to the file at time  $i$  from the node  $x_i$  of the network, and output  $y_i \in Y =$   
 260  $\{0, 1\}$  represents the location of the file at time  $i$ . We choose the horizon  $r = 1$ , aggregation  
 261 function “sum”, and objective “min”, and define the value function as

$$262 \quad v\left(\begin{smallmatrix} \cdot & 0 \\ 0 & 0 \end{smallmatrix}\right) = 0, \quad v\left(\begin{smallmatrix} \cdot & 0 \\ 1 & 1 \end{smallmatrix}\right) = 1, \quad v\left(\begin{smallmatrix} \cdot & 0 \\ 1 & 0 \end{smallmatrix}\right) = \alpha, \quad v\left(\begin{smallmatrix} \cdot & 0 \\ 0 & 1 \end{smallmatrix}\right) = 1 + \alpha,$$

$$263 \quad v\left(\begin{smallmatrix} \cdot & 1 \\ 1 & 1 \end{smallmatrix}\right) = 0, \quad v\left(\begin{smallmatrix} \cdot & 1 \\ 0 & 0 \end{smallmatrix}\right) = 1, \quad v\left(\begin{smallmatrix} \cdot & 1 \\ 0 & 1 \end{smallmatrix}\right) = \alpha, \quad v\left(\begin{smallmatrix} \cdot & 1 \\ 1 & 0 \end{smallmatrix}\right) = 1 + \alpha.$$

265 Recall that  $\alpha > 0$  is the cost of migrating the file. Intuitively, the four columns represent  
 266 local access, remote access, local and remote access after reconfiguration.

267 ► **Example 3.2** (online load balancing). Each day  $i$  a job arrives; the job has a duration  
 268  $x_i \in X = \{1, 2, \dots, \ell\}$ . We need to choose a machine  $y_i \in Y$  that will process the job. If,  
 269 e.g.,  $x_i = 3$ , then machine  $y_i$  will process job  $i$  during days  $i$ ,  $i+1$ , and  $i+2$ . The *load* of a  
 270 machine is the number of concurrent jobs that it is processing at a given day, and our task is  
 271 to minimize the maximum load of any machine at any point of time.

272 In this case we can choose the horizon  $r = \ell - 1$ , aggregation function “max”, and  
 273 objective “min”, and define the value function as follows:

$$274 \quad v(x_{i-r}, \dots, x_i, y_{i-r}, \dots, y_i) = \max_{y \in Y} \left| \{j \in X : x_{i-j+1} \geq j \text{ and } y_{i-j+1} = y\} \right|.$$



275 That is, we count the number of jobs that were assigned to each machine  $y \in Y$  on days  
 276  $i - r, \dots, i$  and that are long enough so that they are still being processed during day  $i$ . For  
 277 example, if  $X = Y = \{1, 2\}$ , this is equivalent to

$$278 \quad v\left(\begin{smallmatrix} 1 \\ \vdots \end{smallmatrix}\right) = 1, \quad v\left(\begin{smallmatrix} 2 \\ 2 \end{smallmatrix}\right) = 2, \quad v\left(\begin{smallmatrix} 2 \\ 1 \end{smallmatrix}\right) = 2, \quad v\left(\begin{smallmatrix} 2 \\ 1 \end{smallmatrix}\right) = 1, \quad v\left(\begin{smallmatrix} 2 \\ 2 \end{smallmatrix}\right) = 1.$$

279 **Encoding Online Problems: Answers vs. Configurations.** The same online problem can  
 280 often be formalized and encoded in several different ways. In general, the output of an  
 281 algorithm in each step can naturally be defined either as

- 282 1. the *action* (i.e. transition) chosen (e.g. “enqueue the incoming job on machine 2”), or
- 283 2. the new *configuration* of the system (e.g. “machine 1 is executing two jobs with remaining  
 284 durations  $a$  and  $b$ , and machine 2 is executing one job with remaining duration  $c$ ”).

285 In the classic online setting, where algorithms have access to the entire past input sequence,  
 286 this distinction tends to matter little (actions can be computed from the full sequence of  
 287 configurations, and vice versa); one simply uses the most convenient formulation.

288 However, in restricted models of computation, such as time-local online algorithms, the  
 289 situation is different. The choice of encoding—e.g. whether the algorithm outputs the actions  
 290 it makes or the current configuration—may dramatically impact the solvability of a problem.  
 291 We discuss this question in more detail in Appendix E.4. Unless otherwise specified, we  
 292 assume that algorithms for online problems output the current configuration in each step.

293 **Examples of Graph Problems on Paths.** In this paper, we uncover and exploit connections  
 294 between time-local online algorithms and distributed graph algorithms on paths. We have  
 295 seen that the formalism that we use is expressive enough to capture typical online problems;  
 296 we now express some classic graph optimization problems studied in distributed computing.  
 297 let us now see how to express some classic graph optimization problems that have been  
 298 studied in the theory of distributed computing.

299 Here, each index  $i$  is interpreted to be a node in a path, and nodes  $i$  and  $i + 1$  are  
 300 connected by an edge. Input  $x_i$  is the *weight* of node  $i$ , and output  $y_i$  encodes a subset of  
 301 nodes  $S \subseteq \{1, 2, \dots, n\}$ , with the interpretation that  $i \in S$  whenever  $y_i = 1$ . Hence  $X = \mathbb{R}_{\geq 0}$   
 302 and  $Y = \{0, 1\}$ .

303 ► **Example 3.3 (maximum-weight independent set).** We can capture a problem equivalent  
 304 to the classic maximum-weight independent set as follows: we choose the horizon  $r = 1$ ,  
 305 aggregation function “sum”, and objective “max”, and define the value function as follows:

$$306 \quad v\left(\begin{smallmatrix} \vdots \\ \vdots \end{smallmatrix}\right) = 0, \quad v\left(\begin{smallmatrix} \vdots \\ 0 \end{smallmatrix} \alpha\right) = \alpha, \quad v\left(\begin{smallmatrix} \vdots \\ 1 \end{smallmatrix}\right) = -\infty.$$

307 That is, a node of weight  $\alpha$  is worth  $\alpha$  units if we select it. The last case ensures that the  
 308 solution represents a valid independent set (no two nodes selected next to each other).

309 ► **Example 3.4 (minimum-weight dominating set).** To represent minimum-weight dominating  
 310 sets, we choose  $r = 2$ , aggr = sum, and obj = min. We define the value function as follows:

$$311 \quad v\left(\begin{smallmatrix} \vdots \\ \vdots \end{smallmatrix} \alpha\right) = \alpha, \quad v\left(\begin{smallmatrix} \vdots \\ 0 \end{smallmatrix} \alpha\right) = 0, \quad v\left(\begin{smallmatrix} \vdots \\ 1 \end{smallmatrix} \alpha\right) = 0, \quad v\left(\begin{smallmatrix} \vdots \\ 0 \end{smallmatrix} \alpha\right) = +\infty.$$

312 Here if we select a node of cost  $\alpha$ , we pay  $\alpha$  units. Nodes that are not selected but that are  
 313 correctly dominated by a neighbor are free. We ensure correct domination by assigning an  
 314 infinite cost to unhappy nodes.

315 Technically, when we select a node  $i$ , we will pay for it at time  $i + 1$ , not at time  $i$ , but  
 316 this is fine, as we will in any case sum over all nodes (and ignore constantly many nodes  
 317 near the boundaries).



## 3.2 Local Algorithms in Time and Space

So far we have introduced the formalism we use to define computational problems. Let us now define the model of computing. For convenience, we will extend the definition of inputs to include a placeholder value  $\perp$  and let  $x_i = \perp$  for  $i < 1$  and for  $i > n$ . The key models of computing that we study are all captured by the following definition:

► **Definition 3.5** (local algorithm). An  $[a, b]$ -local algorithm is a sequence  $(A_i)_{i \geq 1}$  of functions of the form  $A_i: X^{a+1+b} \rightarrow Y$ . The output  $\mathbf{y}$  of algorithm  $A$  for input  $\mathbf{x} \in X^n$ , in notation  $\mathbf{y} = A(\mathbf{x})$ , is defined as follows:

$$y_i = A_i(x_{i-a}, \dots, x_{i+b}) \text{ for each } i = 1, \dots, n.$$

If  $A_i = A_j$  for all  $i, j \in \mathbb{N}$ , then algorithm  $A$  is *unclocked*. Otherwise, algorithm is *clocked*.

Note that *unclocked* time-local algorithms as defined above are unaware of the current time step  $i$ ; they make the same deterministic decision every time for the same (local) input pattern. We can quantify the cost of not being aware of the current time step, by comparing *unclocked* algorithms against the stronger model of *clocked* algorithms, which can make different decisions based on the current time step  $i$ .

**Classic Models of Online and Distributed Algorithms.** Using the notion of *unclocked* time-local algorithms, we can characterize algorithms studied in prior work as follows; see also Figure 1. In what follows,  $T$  is a constant independent of the length  $n$  of input:

- **$[\infty, \infty]$ -local:** These are algorithms with access to the full input. In the context of online algorithms, these are usually known as *offline algorithms*, while in the context of distributed computing, these are usually known as *centralized algorithms*.
- **$[\infty, -1]$ -local:** These are *online algorithms* in the usual sense. The output for time step  $i$  is chosen based on inputs for all previous time steps up to time step  $i - 1$ . This is an appropriate definition for the online file migration problem (Example 3.1): we need to decide where to move the file before we see the next request.
- **$[\infty, 0]$ -local:** These are online algorithms with one unit of *lookahead*. The output for time step  $i$  is chosen based on inputs up to time step  $i$ . This is an appropriate definition for the online load balancing problem (Example 3.2): we can choose the machine once we see the parameters of the new job.
- **$[T, T]$ -local:** These can be interpreted as  $T$ -round *distributed algorithms* in directed paths in the port-numbering model. In the port-numbering model, in  $T$  synchronous communication rounds, each node can gather full information about the inputs of all nodes within distance  $T$  from it, and nothing else. This is a setting in which it is interesting to study graph problems such as the maximum-weight independent set (Example 3.3) and the minimum-weight dominating set (Example 3.4).

**New Models: Time-Local Online Algorithms.** Now we are ready to introduce the main objects of study for the present work:

- **unclocked  $[T, -1]$ -local:** These are *time-local algorithms* with horizon  $T$ , i.e., online algorithms that make decisions based on only  $T$  latest inputs.
- **unclocked  $[T, 0]$ -local:** These are time-local algorithms with one unit of *lookahead*.
- **clocked  $[T, T]$ -local:** As we will see later, these algorithms are equivalent to  $T$ -round distributed algorithms a restricted variant of the *supported LOCAL* model [26, 41].
- **clocked  $[T, -1]$ -local:** These are *clocked time-local algorithms* that make decisions based on only  $T$  latest inputs, but the decision may depend on the current time step  $i$ .

■ **Table 1** Correspondence between time-local online algorithms and distributed graph algorithms.

	Time-local online algorithms	Local distributed graph algorithms on directed paths
Weakest	unclocked $[T, T]$ -local N/A	$T$ rounds in the PN model [1, 2, 45] $T$ rounds in the LOCAL model [33, 39]
	clocked $[T, T]$ -local	$T$ rounds in the numbered LOCAL model
Strongest	N/A	$T$ rounds in the supported LOCAL model [26, 41]

We note that there is nothing fundamental about the constants  $-1$  and  $0$  that appear above; they are merely constants that usually make most sense in applications. Hence one can perfectly well study, say,  $[10, 7]$ -local algorithms, and interpret them either as distributed algorithms that make decisions based on an asymmetric local neighborhood, or interpret them as time-local algorithms that can postpone decisions and choose  $y_i$  only after seeing inputs up to  $i + 7$ .

## 4 Overview of Technical Contributions

We are now ready to overview our main technical contributions; all technical details and further results are provided in the appendix.

**Transferring Results from Distributed Computing (Appendix A).** First, we identify connections between different models studied in *distributed graph algorithms* and different variants of *time-local online algorithms*. These are summarized in Table 1.

We exploit this connection, and show how to lift many results from theory of distributed computing to establish *impossibility results* for time-local algorithms essentially for free. For example, it turns out that if an online problem has a component that is equivalent to a symmetry-breaking task or to a nontrivial distributed optimization problem, such as in online load balancing (Example 3.2), it follows that deterministic unclocked time-local algorithms cannot perform well, but randomization helps.

On the other hand, this suggests that problems where symmetry-breaking is not a critical component, such as in the online file migration problem (Example 3.1), deterministic unclocked time-local algorithms can perform well. This heuristic argument is corroborated by our case study on online file migration: the problem admits competitive time-local algorithms.

**The Power of Clocked Algorithms (Appendix B).** Second, we establish that deterministic *clocked* time-local algorithms are powerful. For the family of *bounded monotone minimization games*, we can turn any classic deterministic online algorithm into a deterministic clocked time-local algorithm with only a small increase in the competitive ratio.

► **Theorem 4.1.** *Let  $\mathcal{F}$  be a bounded monotone minimization game. If there exists an online algorithm  $A$  with competitive ratio  $c$  for  $\mathcal{F}$ , then for any constant  $\varepsilon > 0$  there exists some constant  $T$  and a clocked  $T$ -time-local algorithm  $B$  with competitive ratio  $(1 + \varepsilon)c$  for  $\mathcal{F}$ .*

We also show that there are problems outside this family that do not admit competitive clocked algorithms: for the classic *online caching problem* [42], no deterministic clocked time-local algorithm can achieve a finite competitive ratio, but competitive classic online algorithms exist.

**Randomization in Time-Local Algorithms (Appendix C).** In the classic online setting, randomized algorithms have two equivalent characterizations:

1. at the start, randomly sample a deterministic algorithm from the set of all algorithms, or
2. in each step, make a random decision based on e.g. a sequence of random coin flips.

The former corresponds to *mixed strategies*, where we sample all random bits used by the algorithm before seeing any of the input, whereas the latter corresponds to *behavioral strategies*, where the algorithm generates random bits along the way as it needs them. We show that these different types of randomized algorithms differ for time-local algorithms, and give a lower bound that holds for both types of algorithms (see below).

**Automated Synthesis of Time-Local Algorithms (Appendix D).** We formalize and implement a technique for the automated design of time-local algorithms for local optimization problems. This technique allows us to automatically obtain *tight* upper and lower bounds for unclocked time-local algorithms. First, we prove the following result.

► **Theorem 4.2** (informal). *Let  $\Pi$  be a local optimization problem and  $A$  an unclocked time-local algorithm with horizon  $T$ . Then there is a finite, dual-weighted graph  $G = G(\Pi, A)$  such that the competitive ratio of  $A$  is determined by the cycle with the heaviest weight ratio in  $G$ .*

Recall that each unclocked time-local algorithm with horizon  $T$  is given by some map  $A: X^T \rightarrow Y$ . For local optimization problems with finite input set  $X$  and output set  $Y$ , we can iterate through all of the  $|Y|^{|X|^T}$  maps to find an optimal algorithm for any given  $T$ . We illustrate the usefulness of this technique by synthesizing several optimal deterministic time-local algorithms for the online file migration problem. Moreover, we also synthesize *randomized* time-local algorithms with strictly better (expected) competitive ratio than the optimal deterministic algorithms under the oblivious adversary. See Figure 2 for a summary.

**Online File Migration: A Case Study (Appendix E).** Finally, as our analytical case study, we establish both lower and upper bounds for the online file migration problem in the time-local setting. The analytical and synthesized algorithms are summarized and compared against classic online algorithms in Figure 2. For example, we show the following results:

► **Theorem 4.3** (informal; see Theorem E.1). *For any  $\alpha \geq T$ , there is no randomized clocked time-local algorithm that achieves a competitive ratio better than  $2\alpha/T$ .*

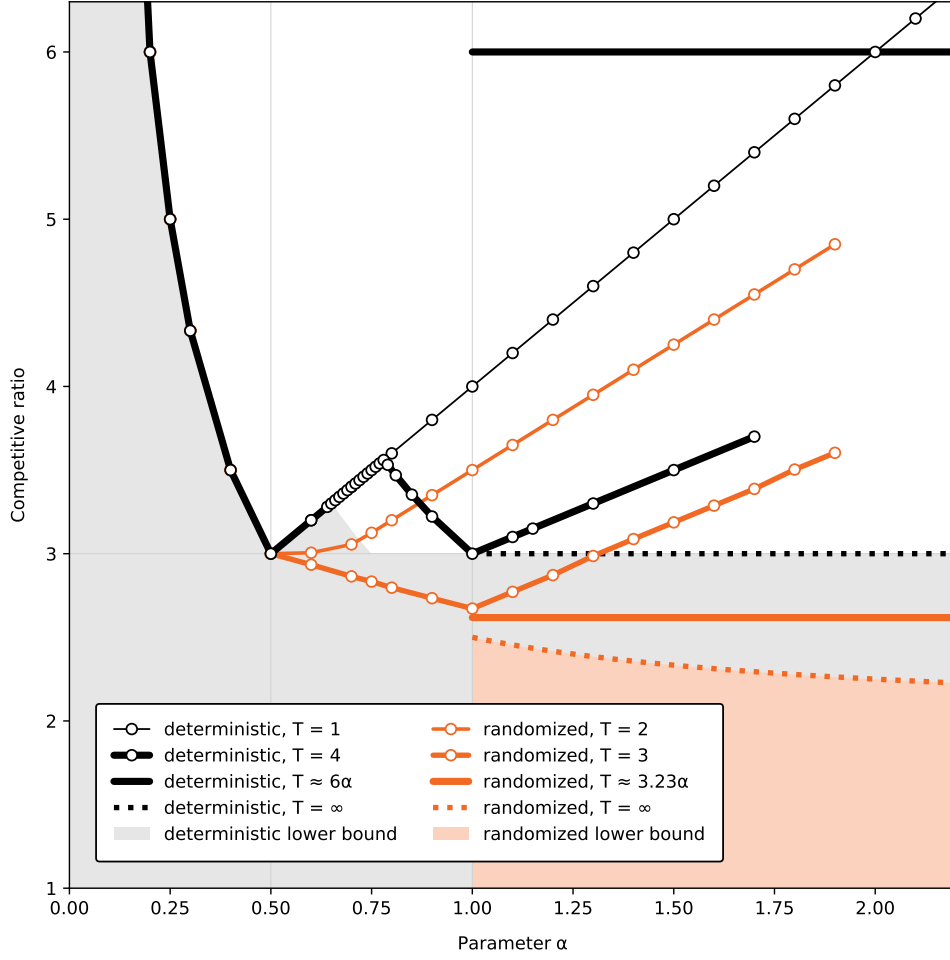
► **Theorem 4.4** (informal; see Theorem E.2). *For any  $\alpha \geq 1$ , there is a randomized clocked time-local algorithm that is  $(1 + \phi)$ -competitive, where  $\phi \approx 1.62$  is the golden ratio, for some  $T = \Theta(\alpha)$ . In general, it achieves a competitive ratio of  $\max\{2 + 2\alpha/T, 1 + (T + 1)/(2\alpha)\}$ .*

► **Theorem 4.5** (informal; see Corollary E.13). *For any  $\alpha \geq 1$ , there is a deterministic unclocked time-local algorithm that is 6-competitive for  $T \geq 6\alpha$ . Moreover, for  $1 \leq T < 6\alpha$  the algorithm is  $(4 + 12\alpha/T)$ -competitive.*

The above deterministic algorithm uses a simple sliding window rule. We say that the last  $T$  inputs contain a  $b$ -window for  $b \in \{0, 1\}$  if there is a subsequence of requests in which the number of  $b$ -requests is at least twice the number of  $(1 - b)$ -requests. The algorithm is:

1. Output  $b \in \{0, 1\}$  if the most recent window in the last  $T$  inputs is a  $b$ -window.
2. Otherwise, output 0.

However, despite the deceptive simplicity of the algorithm, its analysis is surprisingly challenging and illuminates aspects involved in the competitive analysis of time-local algorithms.



**Figure 2** Upper and lower bounds for the online file migration problem. The visualization includes the upper bounds from Table 2 for small values of  $T$ , as well as the upper bounds from Corollaries E.3 and E.13, and the lower bound from Corollary E.16.

## 5 Conclusions

In this work, we initiated the study of time-local online algorithms, and its connections with distributed graph algorithms. We used a special case of the online file migration problem as a running example. We saw that even this simple problem already exhibits a wide range of behaviors in the time-local setting, and identified new questions that need further study, also for classic online algorithms; among them is the analysis of the problem for the range  $1/2 < \alpha < 1$  and how different types of randomized time-local algorithms relate to each other and to the existence of competitive deterministic time-local algorithms.

Beyond these, there is a wide variety of online problems that would be a good fit with the framework of time-local algorithms, and where the approach of synthesizing optimal algorithms may similarly lead to new discoveries. Moreover, our work suggests also new directions for the study of local distributed graph algorithms. Clocked time-local algorithms are a natural application for distributed algorithms in the supported LOCAL model and its slightly weaker variant, numbered LOCAL model, introduced here, and the capabilities of these models in distributed optimization are not fully understood yet.

## References

- 1 Dana Angluin. Local and global properties in networks of processors. In *Proc. 12th Annual ACM Symposium on Theory of Computing (STOC 1980)*, 1980. doi:10.1145/800141.804655.
- 2 Hagit Attiya, Marc Snir, and Manfred K. Warmuth. Computing on an anonymous ring. *Journal of the ACM*, 35(4):845–875, 1988. doi:10.1145/48014.48247.
- 3 Yossi Azar, Andrei Z. Broder, and Anna R. Karlin. On-line load balancing. *Theor. Comput. Sci.*, 130(1):73–84, 1994. doi:10.1016/0304-3975(94)90153-8.
- 4 Alkida Balliu, Sebastian Brandt, Yi-Jun Chang, Dennis Olivetti, Mikael Rabie, and Jukka Suomela. The distributed complexity of locally checkable problems on paths is decidable. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 262–271, 2019.
- 5 Alkida Balliu, Sebastian Brandt, Yuval Efron, Juho Hirvonen, Yannic Maus, Dennis Olivetti, and Jukka Suomela. Classification of distributed binary labeling problems. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing (DISC 2020)*, volume 179 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:17. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.DISC.2020.17.
- 6 Yair Bartal, Amos Fiat, and Yuval Rabani. Competitive algorithms for distributed data management. *J. Comput. Syst. Sci.*, 51(3):341–358, 1995. doi:10.1006/jcss.1995.1073.
- 7 Sanjoy Baruah, Gilad Koren, Decao Mao, Bhuvaneshwar Mishra, Arvind Raghunathan, Louis Rosier, Dennis Shasha, and Fuxing Wang. On the competitiveness of on-line real-time task scheduling. *Real Time Syst.*, 4(2):125–144, 1992. doi:10.1007/BF00365406.
- 8 Shai Ben-David, Allan Borodin, Richard M. Karp, Gábor Tardos, and Avi Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, 1994. doi:10.1007/BF01294260.
- 9 Marcin Bienkowski. Migrating and replicating data in networks. *Computer Science-Research and Development*, 27(3):169–179, 2012.
- 10 Marcin Bienkowski, Jaroslaw Byrka, and Marcin Mucha. Dynamic beats fixed: On phase-based algorithms for file migration. *ACM Trans. Algorithms*, 15(4):46:1–46:21, 2019. doi:10.1145/3340296.
- 11 David L. Black and Daniel D. Sleator. Competitive algorithms for replication and migration problems. Technical Report CMU-CS-89-201, Carnegie Mellon University, 1989. URL: <https://www.cs.cmu.edu/~sleator/papers/migration-problems.pdf>.
- 12 Paolo Boldi and Sebastiano Vigna. An effective characterization of computability in anonymous networks. In *Proc. 15th International Symposium on Distributed Computing (DISC 2001)*, volume 2180 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 2001. doi:10.1007/3-540-45414-4\_3.
- 13 Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
- 14 Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, 1992. doi:10.1145/146585.146588.
- 15 Sebastian Brandt, Juho Hirvonen, Janne H Korhonen, Tuomo Lempiäinen, Patric RJ Östergård, Christopher Purcell, Joel Rybicki, Jukka Suomela, and Przemysław Uznański. Lcl problems on grids. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 101–110, 2017.
- 16 Yi-Jun Chang and Seth Pettie. A time hierarchy theorem for the local model. *SIAM Journal on Computing*, 48(1):33–69, 2019.
- 17 Yi-Jun Chang, Jan Studený, and Jukka Suomela. Distributed graph problems through an automata-theoretic lens, 2020. To appear in SIROCCO 2021. URL: <https://arxiv.org/abs/2002.07659>.
- 18 Krishnendu Chatterjee, Andreas Pavlogiannis, Alexander Kößler, and Ulrich Schmid. Automated competitive analysis of real-time scheduling with graph games. *Real Time Syst.*, 54(1):166–207, 2018. doi:10.1007/s11241-017-9293-4.

- 504   19   Marek Chrobak and Lawrence L. Larmore. The server problem and on-line games. In *On-Line Algorithms, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, February 11-13, 1991*, volume 7 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 11–64. DIMACS/AMS, 1991. doi:10.1090/dimacs/007/02.
- 505
- 506
- 507
- 508   20   Marek Chrobak, Lawrence L. Larmore, Nick Reingold, and Jeffery Westbrook. Page migration algorithms using work functions. pages 406–415, 1993.
- 509
- 510   21   Richard Cole and Uzi Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986. doi:10.1016/S0019-9958(86)80023-7.
- 511
- 512
- 513   22   Don Coppersmith, Peter Doyle, Prabhakar Raghavan, and Marc Snir. Random walks on weighted graphs and applications to on-line algorithms. *J. ACM*, 40(3):421–453, 1993. doi:10.1145/174130.174131.
- 514
- 515
- 516   23   Andrzej Czygrinow, Michał Hańćkowiak, and Wojciech Wawrzyniak. Fast distributed approximations in planar graphs. In *Proc. 22nd International Symposium on Distributed Computing (DISC 2008)*, volume 5218 of *Lecture Notes in Computer Science*, pages 78–92. Springer, 2008. doi:10.1007/978-3-540-87779-0\_6.
- 517
- 518
- 519
- 520   24   Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, 1974. doi:10.1145/361179.361202.
- 521
- 522   25   Shlomi Dolev. *Self-Stabilization*. MIT Press, 2000.
- 523   26   Klaus-Tycho Foerster, Juho Hirvonen, Jukka Suomela, and Stefan Schmid. On the power of preprocessing in decentralized network optimization. In *Proc. IEEE Conference on Computer Communications (INFOCOM 2019)*, 2019. doi:10.1109/INFOCOM.2019.8737382.
- 524
- 525
- 526   27   Mika Göös, Juho Hirvonen, and Jukka Suomela. Lower bounds for local approximation. *Journal of the ACM*, 60(5), 2013. doi:10.1145/2528405.
- 527
- 528   28   Juho Hirvonen, Joel Rybicki, Stefan Schmid, and Jukka Suomela. Large cuts with local algorithms on triangle-free graphs. In *The Electronic Journal of Combinatorics (EJC)*, 2017.
- 529
- 530   29   Takashi Horiyama, Kazuo Iwama, and Jun Kawahara. Finite-state online algorithms and their automated competitive analysis. In *Proceedings of the 17th International Conference on Algorithms and Computation*, ISAAC’06, pages 71–80, 2006. doi:10.1007/11940128\_9.
- 531
- 532
- 533   30   Sandy Irani and Steve Seiden. Randomized algorithms for metrical task systems. *Theoretical Computer Science*, 194(1):163–182, 1998. doi:10.1016/S0304-3975(97)00006-6.
- 534
- 535   31   Kazuo Iwama and Shiro Taketomi. Removable online knapsack problems. In *International Colloquium on Automata, Languages, and Programming*, pages 293–305, 2002.
- 536
- 537   32   Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel D. Sleator. Competitive snoopy caching. *Algorithmica*, 3:77–119, 1988. doi:10.1007/BF01762111.
- 538
- 539   33   Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992. doi:10.1137/0221015.
- 540
- 541   34   Akira Matsubayashi. A  $3 + \Omega(1)$  lower bound for page migration. *Algorithmica*, 82(9):2535–2563, 2020. doi:10.1007/s00453-020-00696-5.
- 542
- 543   35   Moni Naor. A lower bound on probabilistic algorithms for distributive ring coloring. *SIAM Journal on Discrete Mathematics*, 4(3):409–412, 1991. doi:10.1137/0404036.
- 544
- 545   36   Moni Naor and Larry Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995.
- 546
- 547   37   A. Pavlogiannis, N. Schaumberger, U. Schmid, and K. Chatterjee. Precedence-aware automated competitive analysis of real-time scheduling. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):3981–3992, 2020. doi:10.1109/TCAD.2020.3012803.
- 548
- 549
- 550   38   Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980. doi:10.1145/322186.322188.
- 551
- 552   39   David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, 2000. doi:10.1137/1.9780898719772.
- 553

- 554 40 Joel Rybicki and Jukka Suomela. Exact bounds for distributed graph colouring. In *International*  
555 *Colloquium on Structural Information and Communication Complexity*, pages 46–60. Springer,  
556 2015.
- 557 41 Stefan Schmid and Jukka Suomela. Exploiting locality in distributed SDN control. In *Proc.*  
558 *2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN*  
559 *2013)*, pages 121–126. ACM Press, 2013. doi:10.1145/2491185.2491198.
- 560 42 Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules.  
561 *Commun. ACM*, 28(2):202–208, 1985. doi:10.1145/2786.2793.
- 562 43 Jukka Suomela. Survey of local algorithms. *ACM Computing Surveys*, 45(2), 2013. doi:  
563 10.1145/2431211.2431223.
- 564 44 Jeffery Westbrook. Randomized algorithms for multiprocessor page migration. *SIAM J.*  
565 *Comput.*, 23(5):951–966, 1994. doi:10.1137/S0097539791199796.
- 566 45 Masafumi Yamashita and Tsunehiko Kameda. Computing on anonymous networks: part  
567 I—characterizing the solvable cases. *IEEE Transactions on Parallel and Distributed Systems*,  
568 7(1):69–89, 1996. doi:10.1109/71.481599.



## 569      **A      Time-Local Online Algorithms vs. Local Graph Algorithms**

570      In this section, we discuss the connection between time-local online algorithms and local  
 571      distributed graph algorithms on paths. Although the former deal with locality in the *temporal*  
 572      dimension and the latter in *spatial* dimension, we will see that these two worlds are closely  
 573      connected. In particular, we show how to transfer results from distributed computing to the  
 574      time-local online setting.

### 575      **A.1      Distributed Graph Algorithms**

576      Let  $G = (V, E)$  be a graph that represents the communication topology of a distributed  
 577      system consisting of  $n$  nodes  $V = \{v_1, \dots, v_n\}$ . Each node  $v_i \in V$  corresponds to a processor  
 578      and the edges denote direct communication links between processors, i.e., any pair of nodes  
 579      connected by an edge can directly communicate with each other. In this work,  $G$  will always  
 580      be a path of length  $n$  with the set of edges given by  $E = \{\{v_i, v_{i+1}\} : 1 \leq i < n\}$ .

581      **Synchronous distributed computation.** We start with the basic synchronous message-  
 582      passing model of computation. Let  $X$  and  $Y$  be the set of input and output labels, respectively.  
 583      The input is the vector  $\mathbf{x} = (x_1, \dots, x_n) \in X^n$ , where  $x_i$  is the local input of node  $v_i$ . Initially,  
 584      each node  $v_i$  only knows its local input  $x_i \in X$ .

585      The computation proceeds in synchronous rounds, where in each round  $t = 1, 2, \dots$ , all  
 586      nodes in parallel perform the following in lock-step:

- 587      1. send messages to their neighbors,
- 588      2. receive messages from their neighbors, and
- 589      3. update their local state.

590      An algorithm has running time  $T$  if at the end of round  $T$ , each node  $v_i$  halts and declares its  
 591      own local output value  $y_i$ . The output of the algorithm is the vector  $\mathbf{y} = (y_1, \dots, y_n) \in Y^n$ .

592      Note that—since there is no restriction on message sizes—every  $T$ -round algorithm can  
 593      be represented as a simple full-information algorithm: In every round, each node broadcasts  
 594      all the information it currently has, i.e., its own local input and inputs it has received from  
 595      others, to all of its neighbors. After executing this algorithm for  $T$  rounds, this algorithm  
 596      has obtained all the information *any*  $T$ -round algorithm can. Thus, every  $T$ -round algorithm  
 597      can be represented as map from radius- $T$  neighborhoods to output values.

### 598      **A.2      Distributed Algorithms vs. Time-Local Online Algorithms**

599      The distributed computing literature has extensively studied the computational power of  
 600      different variants of the above basic model of graph algorithms. The variants are obtained  
 601      by considering different types of *symmetry-breaking information*: in addition to the problem  
 602      specific local input  $x_i \in X$ , each node  $v_i$  also receives some input  $z_i$  that encodes additional  
 603      model-dependent symmetry-breaking information.

604      We will now discuss four such models in increasing order of computational power. The  
 605      correspondence between these models and time-local online algorithms is summarized by  
 606      Table 1.

607      **The port-numbering model PN on directed paths.** In the PN model [1, 2, 45] all nodes are  
 608      anonymous, but the edges of  $G$  are consistently oriented from  $v_i$  towards  $v_{i+1}$  for all  $1 \leq i < n$ .  
 609      The nodes know their degree and can distinguish between the incoming and outgoing edges.

The orientation only serves as symmetry-breaking information; the communication links are bidirectional.

Any deterministic algorithm in this model corresponds to a map  $A: X^{2T+1} \rightarrow Y$  such that the output of node  $v_i$  for  $1 \leq i \leq n$  is

$$y_i = A(x_{i-T}, \dots, x_i, \dots, x_{i+T}),$$

where we let  $x_j = \perp$  for any  $j < 0$  or  $j > n$  (the  $\perp$  values are used in the scenarios where nodes near the endpoints of the path observe these endpoints). Note that this is exactly the definition of an unlocked  $[T, T]$ -local algorithms (Definition 3.5).

**The LOCAL model on directed paths.** In the LOCAL model [33, 39] each node receives the same information as in the port-numbering model PN, but in addition, each node  $v_i$  is also given a unique identifier  $ID(v_i)$  from the set  $\{1, \dots, n^c\}$  for some constant  $c \geq 1$ ; the nodes do not know  $n$ . Lower bounds for this model also hold in the weaker PN model.

**The numbered LOCAL model on directed paths.** The numbered LOCAL model further assumes that the unique identifiers have a specific, ordered structure: node  $v_i$  is given the identifier  $ID(v_i) = i$  as local input in addition to the problem specific input  $x_i \in X$ . That is, each node knows its distance from the start of the path. Any deterministic algorithm in this model corresponds to a map  $A: X^{2T+1} \times \mathbb{N} \rightarrow Y$  such that the output of node  $v_i$  for  $1 \leq i \leq n$  is

$$y_i = A(x_{i-T}, \dots, x_i, \dots, x_{i+T}, i) = A_i(x_{i-T}, \dots, x_i, \dots, x_{i+T}).$$

Observe that this coincides with *clocked*  $[T, T]$ -local algorithms (Definition 3.5).

This model is *not* something that to our knowledge has been studied in the distributed computing literature; the name “numbered LOCAL model” is introduced here. However, it is very close to another model, so-called supported LOCAL model, which has been studied in the literature.

**The supported LOCAL model on directed paths.** The supported LOCAL model [26, 41] is the same as the numbered LOCAL model, but each node is also given the length  $n$  of the path as local input. This would correspond to *clocked*  $[T, T]$ -local algorithms that also know the length of the input in advance but do not see the full input. This is the most powerful model, and hence, all impossibility results in this model also hold for all the previous models.

### A.3 Transferring Results From Distributed Computing

#### A.3.1 Symmetry-Breaking Tasks in Distributed Computing

One of the key challenges in distributed graph algorithms is local symmetry breaking: two adjacent nodes in a graph (here: two consecutive nodes along the path) have got isomorphic local neighborhoods but are expected to produce different outputs.

In distributed computing, a canonical example is the *vertex coloring problem*. Consider, for example, the task of finding a proper coloring with  $k$  colors. This is trivial in the supported and numbered models (node number  $i$  can simply output e.g.  $i \bmod 2$  to produce a proper 2-coloring). However, the case of the PN model and the LOCAL model is a lot more interesting.

One can use simple arguments based on *local indistinguishability* [12, 45] to argue that such tasks are not solvable in  $o(n)$  rounds in the PN model. In brief, if two nodes have

identical radius- $T$  neighborhoods, then they will produce the same output in any deterministic PN-algorithm that runs in  $T$  rounds. For example, it immediately follows  $k$ -coloring for any  $k$  requires  $\Omega(n)$  rounds in the deterministic PN model.

Yet another idea one can exploit in the analysis of symmetry-breaking tasks is *rigidity* (or, put otherwise, the lack of *flexibility*); see e.g. [15, 17]. For example, 2-coloring is a rigid problem: once the output of one node is fixed, all other nodes have fixed their outputs. Informally, two nodes arbitrarily far from each other need to be able to coordinate their decisions—or otherwise there is at least one node between them that produces the wrong output. This idea can be used to quickly show that e.g. 2-coloring in the LOCAL model requires also  $\Omega(n)$  rounds, and this holds even if we consider *randomized* algorithms (say, Monte Carlo algorithms that are supposed to work w.h.p.).

This leaves us with the case of symmetry-breaking tasks that *are* flexible. A canonical example is the 3-coloring problem. Informally, one can fix the colors of any two nodes (sufficiently far from each other), and it is always possible to complete the coloring between them. While the 3-coloring problem requires  $\Omega(n)$  rounds in the deterministic PN model, it is a problem that can be solved much faster in the deterministic LOCAL model and also in the randomized PN model: the Cole–Vishkin technique [21] can be used to do it in only  $O(\log^* n)$  rounds. However, what is important for us in this work is that this is also known to be tight [33, 35]: 3-coloring is not possible in  $o(\log^* n)$  rounds, not even if we use both unique identifiers and randomness.

Moreover, the same holds for *all* problems in which the task is to label a path with some labels from a constant-sized set  $Y$ , and arbitrarily long sequences of the same label are forbidden: no such problem can be solved in constant time in the PN or LOCAL model, not even if one has got access to randomness [16, 33, 35, 36, 43].

We will soon see what all of this implies for us, but let us discuss one technicality first: symmetric vs. asymmetric horizons.

### A.3.2 Symmetric vs. Asymmetric Horizons

While the standard models in distributed computing correspond to *symmetric* horizons ( $[T, T]$ -local algorithms) and the study of online algorithms is typically interested in *asymmetric* horizons (e.g.  $[T, -1]$ -local algorithms), in many cases this distinction is inconsequential when one considers symmetry-breaking tasks.

Consider, for example, the vertex coloring problem  $\Pi$ . Assume one is given an  $[a, b]$ -local algorithm  $A$  for solving  $\Pi$ . Now for any constant  $c$  one can construct an  $[a + c, b - c]$ -local algorithm  $A'$  that solves the same problem. In essence, node  $x_i$  in algorithm  $A'$  simply outputs  $A(x_{i-a-c}, \dots, x_{i+b-c})$ . Now if one compares the outputs of  $A'$  and  $A$ , we produce the same sequence of colors but shifted by  $c$  steps. This is the standard trick one uses to convert algorithms for directed paths into algorithms for rooted trees and vice versa; see e.g. [17, 40]. The only caveat is that we need to worry about what to do near the boundaries, but for our purposes the very first and the very last outputs are usually inconsequential (can be handled by an ad hoc rule, or simply ignored thanks to the additive constant in the definition of the competitive ratio).

Hence, in essence everything that we know about symmetry-breaking tasks in the context of  $[T, T]$ -local algorithms can be easily translated into equivalent results for  $[T', -1]$ -local algorithms for  $T' = 2T + 1$ , and vice versa.

### A.3.3 Distributed Optimization and Approximation

So far we have discussed distributed graph problems in which the task is to find *any* feasible solution subject to some local constraints. However, especially in the context of online algorithms, we are usually interested in finding *good* solutions. Typical examples are problems such as the task of finding the minimum dominating set problem and the maximum independent set problem.

These are not, strictly speaking, symmetry-breaking tasks. Nevertheless, it turns out to be useful to look at also such tasks through the lens of symmetry breaking. In brief, the following picture emerges [23, 27, 36, 43]:

- Deterministic  $O(1)$ -round LOCAL-model algorithms are not any more powerful than deterministic  $O(1)$ -round PN-model algorithms.
- Randomized  $O(1)$ -round algorithms are strictly stronger than deterministic  $O(1)$ -round algorithms.

For example, if we look at the *minimum dominating set problem* in unweighted paths, the only possible deterministic  $O(1)$ -round PN-algorithm produces a constant output: all nodes (except possibly some nodes near the boundaries) are part of the solution. Deterministic  $O(1)$ -round LOCAL-algorithm can *try* to do something much more clever, with the help of unique identifiers, but a Ramsey-type argument [23, 27, 36] shows that it is futile: there always exists an adversarial assignment of unique identifiers such that the algorithm produces a near-constant output for all but  $\epsilon n$  many nodes, for an arbitrarily small  $\epsilon > 0$ . However, randomized algorithms can do much better (at least on average); to give a simple example, consider an algorithm that first takes each node with some fixed probability  $0 < p < 1$ , and then adds the nodes that were not yet dominated. Finally, in the numbered and supported models one can obviously do much better, even deterministically (simply pick every third node).

This is now enough background on the most relevant results related to  $T$ -round algorithms in deterministic and randomized PN and LOCAL models.

### A.3.4 Consequences: Time-Local Solvability

It turns out to be highly beneficial to try to classify online problems in the above terms: whether there is a component that is equivalent to a symmetry-breaking task or to a nontrivial distributed optimization problem. This is easiest to explain through examples:

**Online file migration (Example 3.1).** This problem is trivial to solve for a constant input; the same also holds for any input sequence that is strictly periodic. Indeed, if the adversary gives a long sequence of constant inputs (or follows a fixed periodic pattern), it only helps us. Hence none of the above obstacles are in our way; interesting inputs are sequences that already break symmetry locally. Furthermore, as we also know that this is a well-known online problem solvable with the full history, we would expect that there is also an unclocked time-local algorithm for solving the task, with a nontrivial competitive ratio. While this is a *heuristic* argument (based on the *lack* of specific obstacles), we will see in Appendix E that the argument works very well in this case.

**Online load balancing (Example 3.2).** This problem is fundamentally different from the file migration problem. Let us assume that the algorithm needs to output the action (on which machine to schedule the current job). Consider an input sequence that consists of the constant value 2. In such a case, there is an optimal solution that alternately assigns the

2-unit jobs to the two machines, ensuring that the load of any machine at any time is exactly 1. But this means that an optimal algorithm has to turn the constant input  $2, 2, 2, \dots$  into a strictly alternating sequence like  $1, 2, 1, 2, \dots$ . Any deviation from it will result at least momentarily in a load of 2. Hence in an *optimal* solution we need to *at least* solve the 2-coloring problem within each segment of such constant inputs. As we discussed, this is not possible in the PN or LOCAL model in  $O(1)$  rounds, not even with the help of randomness; it follows that there certainly is no optimal unlocked time-local online algorithm, with any constant horizon  $T$ . Optimal solutions have to resort to the clock.

However, this does not prevent us from solving the problem with a finite competitive ratio. Indeed, even the trivial solution that outputs always 1 will result in a maximum load that is at most 2 times as high as optimal.

Furthermore, if we were not interested in the *maximum* load but the *average* load, we arrive at a task that is, in essence, a distributed optimization problem. Unlocked *randomized* time-local algorithms may then have an advantage over unlocked *deterministic* time-local algorithms, and indeed this turns out to be the case here: simply choosing the machine at random is already better on average than assigning all tasks to the same machine.

### A.3.5 Consequences: Time-Local Models

On a more general level, the above discussion also leads to the following observation: the definition of *unlocked* time-local algorithms is *robust*. Now it coincides with the PN model, but even if one tried to strengthen it so that its expressive power was closer to the LOCAL model, very little would change in terms of the results.

Conversely, if one weakened the *clocked* model so that e.g. the clock values are not increasing by one but they are only a sequence of monotone, polynomially-bounded time stamps, we would arrive at a model very similar to the LOCAL model, and as we have seen above, time-local algorithms in such a model cannot solve symmetry-breaking tasks any better than in the unlocked model. Hence in order to capture the idea of a model that is strictly more powerful than the unlocked model, it is not sufficient to have a definition in which the clock values are merely monotone and polynomially bounded, but one has to further require e.g. that the clock values increase at each step at most by a constant. (Such a model with constant-bounded clock increments would indeed be a meaningful alternative, and it would fall in its expressive power strictly between our unlocked and clocked models. It would be strong enough to solve 3-coloring but not strong enough to solve 2-coloring in a time-local fashion. We do not explore this variant further, but it may be an interesting topic for further research, especially when comparing its power with randomized PN algorithms.)

## B Clocked Time-Local Algorithms

In this section, we examine the power of clocked time-local algorithms. First, we show that clocked time-local algorithms can be powerful, despite their limited access to the input: for many problems, competitive classic online algorithms can be converted into competitive clocked time-local algorithms. Second, we complement the first result by giving an example of a classic online problem that does admit a competitive clocked time-local algorithms, despite having competitive classic online algorithms.

## B.1 Clocked Time-Local Algorithms from Full-History Algorithms

We now show that for a large class of online problems the following result holds: if the problem admits a *deterministic* classic online algorithm with competitive ratio  $c$ , then for any given constant  $\varepsilon > 0$  there exists a deterministic clocked time-local algorithm with a competitive ratio of at most  $(1 + \varepsilon)c$  for some constant horizon  $T$ .

The proof follows a similar structure as the constructive derandomization proof of Ben-David et al. [8, Section 4] for classic online algorithms: we chop the input sequence into short segments and show that under certain assumptions both the offline and competitive online algorithms pay roughly the same cost. However, some care is needed to adapt the proof strategy, as in the case of time-local algorithms, we can only use constant-size segments.

**Bounded Minimization Games.** We now define the class of request-answer games for which we prove our result. A (minimization) game is *monotone* if for all  $n \in \mathbb{N}$

$$f_{n+1}(\mathbf{x} \cdot x, \mathbf{y} \cdot y) \geq f_n(\mathbf{x}, \mathbf{y}) \text{ for all } \mathbf{x} \in X^n, \mathbf{y} \in Y^n, x \in X, y \in Y.$$

That is, the cost cannot decrease when extending the input-output sequence. We say that a monotone game has *bounded delay* if for every  $h \in \mathbb{R}$  the set

$$L(h) = \left\{ \mathbf{x} \in \bigcup_{n=1}^{\infty} X^n : \text{OPT}(\mathbf{x}) \leq h \right\}$$

is finite (sometimes this property is called locality [8]). That is, there cannot be arbitrarily long sequences of the fixed cost: eventually the cost of any sequence must increase. Finally, the diameter of the game is

$$D = \sup \left\{ |f(\mathbf{x} \cdot \mathbf{x}', \mathbf{y} \cdot \mathbf{y}') - f(\mathbf{x}, \mathbf{y}) - f(\mathbf{x}', \mathbf{y}')| : (\mathbf{x}, \mathbf{y}), (\mathbf{x}', \mathbf{y}') \in \bigcup_{n \geq 0} X^n \times Y^n \right\}.$$

We define that a *bounded monotone minimization game* is a monotone minimization game that has bounded delay, finite diameter, and finite input set  $X$ . Note that bounded monotone minimization games are not necessarily local optimization problems as defined in Section 3.1. The latter are monotone games with bounded diameter, but they do not necessarily have bounded delay. The following result holds for deterministic algorithms:

► **Theorem B.1.** *Let  $\mathcal{F}$  be a bounded monotone minimization game. If there exists an online algorithm  $A$  with competitive ratio  $c$  for  $\mathcal{F}$ , then for any constant  $\varepsilon > 0$  there exists some constant  $T$  and a clocked  $T$ -time-local algorithm  $B$  with competitive ratio  $(1 + \varepsilon)c$  for  $\mathcal{F}$ .*

**Proof.** Since  $A$  has competitive ratio  $c$ , then there exists some constant  $d$  such that for every input  $\mathbf{x}$  the output  $\mathbf{y} = A(\mathbf{x})$  satisfies  $f(\mathbf{x}, \mathbf{y}) \leq c \cdot \text{OPT}(\mathbf{x}, \mathbf{y}) + d$ . Let  $D$  be the diameter of the game and fix

$$\delta = \frac{2\varepsilon}{3} \quad \text{and} \quad H = \left( \frac{2 + \delta}{\delta} \right) \cdot \max\{d + D\}.$$

Since the game has bounded delay, we have that

$$L(H) = \{ \mathbf{x} : \text{OPT}(\mathbf{x}) \leq H \} \quad \text{and} \quad T = \max\{k + 1 : (x_1, \dots, x_k) \in L(H)\}$$

are finite. Note that  $T$  is independent of  $n$ , as it only depends on  $H$ . Observe that since the cost functions are monotone, for all  $n \geq T$  any input sequence  $\mathbf{x} \in X^n$  satisfies  $\text{OPT}(\mathbf{x}) \geq H$ .

We can now construct the *clocked* time-local algorithm that only sees the  $T$  latest inputs and the total number of requests served so far. Let  $A = (A_i)_{i \geq 1}$  be the classic online algorithm. The clocked time-local algorithm  $B$  is given by sequence  $(B_j)_{j \geq 1}$ , where

$$B_{Tk+i}(z_1, \dots, z_T) = A_i(z_1, \dots, z_i) \text{ for } 1 \leq i \leq T \text{ and } k \geq 0.$$

That is, the clocked time local algorithm  $B$  simulates the classic online algorithm  $A$  by resetting it every time  $T$  inputs have been served since the last reset.

We now analyze the clocked time local algorithm  $B$ . For any  $n \in \mathbb{N}$ , let  $\mathbf{x} \in X^n$  be some input sequence and  $\mathbf{y} \in Y^n$  be the output of  $B$  on the input sequence  $\mathbf{x}$ . Let  $\mathbf{x}(1), \dots, \mathbf{x}(k)$  be the subsequences of  $\mathbf{x}$ , where  $\mathbf{x}(1)$  denote the first  $T$  inputs,  $\mathbf{x}(2)$ , denote the next  $T$  inputs, and so on. Define the shorthand  $C(i) = \text{OPT}(\mathbf{x}(i))$  for each  $1 \leq i \leq k$ . Note that  $C(i) \geq H$  for each  $1 \leq i < k$ . The last subsequence  $\mathbf{x}(k)$  may consist of fewer than  $T$  inputs, so we have no lower bound for  $C(k)$ . For  $1 \leq i < k$ , we get that

$$C(i) - D \geq \left( \frac{2}{2 + \delta} \right) \cdot C(i) \quad \text{and} \quad D + d \leq \left( \frac{2\delta}{2 + \delta} \right) \cdot C(i)$$

by applying the fact that  $C(i) \geq H$  and the definition of  $H$ .

By repeatedly applying the definition of diameter, we get that the optimum offline solution is lower bounded by

$$\text{OPT}(\mathbf{x}) \geq C(1) + \sum_{i=2}^k (C(i) - D) \geq \sum_{i=1}^k (C(i) - D) \geq \left( \frac{2}{2 + \delta} \right) \sum_{i=1}^k C(i).$$

Since  $A$  has competitive ratio  $c$ , the output of  $B$  has cost

$$f(\mathbf{x}, B(\mathbf{x})) \leq c \cdot C(1) + d + \sum_{i=2}^k (c \cdot C(i) + d + D) \leq \left( c + \frac{2\delta}{2 + \delta} \right) \sum_{i=1}^k C(i) + d.$$

Now using the lower bound on  $\text{OPT}(\mathbf{x})$  and the definition of  $\delta$ , we get that the output of  $B$  has cost bounded by

$$\begin{aligned} f(\mathbf{x}, B(\mathbf{x})) &\leq \left( c + \frac{2\delta}{2 + \delta} \right) \sum_{i=1}^k C(i) + d \\ &\leq \left( c + \frac{2\delta}{2 + \delta} \right) \cdot \left( \frac{2 + \delta}{2} \right) \text{OPT}(\mathbf{x}) + d \\ &= c \left( 1 + \frac{\delta}{2} \right) + \delta + d = (1 + \varepsilon)c + d. \end{aligned}$$

## B.2 The Limitations of Clocked Time-Local Algorithms

If the assumptions of Theorem B.1 are not satisfied, how badly can clocked time-local algorithms perform? We now give an example of an online problem for which no time-local algorithm is competitive, despite the existence of competitive classic online algorithms: the *online caching problem* [42].

In this problem, the input set  $X = \{1, \dots, m\}$  coincides with the set of elements that can be stored in the cache of size  $k$ , and the set  $Y = \{A \subseteq X : |A| \leq k\}$  of outputs corresponds to the set of files currently stored in the cache. The result holds already for universe of size  $m = 3$  and cache of size  $k = 2$ .



852 ► **Theorem B.2.** *There is no deterministic clocked time-local algorithm for online caching*  
 853 *with cache size  $k = 2$  that has finite competitive ratio.*

854 **Proof.** Consider the caching problem with the cache size  $k = 2$  and an universe of  $m = 3$   
 855 elements  $X = \{1, 2, 3\}$ . Let  $A$  be any clocked time-local algorithm with horizon  $T = O(1)$ .  
 856 Fix an arbitrary element  $a \in X$ . We say that  $A$  is *indecisive* if for each  $t \in \mathbb{N}$  there exists  
 857 a time  $t' > t$  such that the algorithm faced with the input  $a^L$  for some  $L > t'$  outputs a  
 858 different output at  $t$  than at  $t'$ , and *decisive* otherwise. Intuitively, an indecisive algorithm  
 859 changes its output infinitely many times on the infinite sequence consisting only of requests  
 860 to  $a$ . Note that any indecisive algorithm  $A$  must be a clocked algorithm.

861 We distinguish two cases. First, suppose that  $A$  is indecisive and consider an input  
 862 sequence family  $\mathcal{I} := \{a^L : L \in \mathbb{N}\}$ . Clearly, the optimal offline solution on any input  $\mathbf{x} \in \mathcal{I}$   
 863 is constant. However, since  $A$  is indecisive,  $A$  may incur an arbitrarily large cost on requests  
 864 from this family, due to an arbitrary number of changes in the output, each causing a cache  
 865 reconfiguration. Thus, any indecisive algorithm has an unbounded competitive ratio.

866 Second, suppose that  $A$  is decisive. Note that its output on the input  $a^n$  for any step  
 867  $1 \leq \tau \leq n$  determines its output on other sequences that contain a sequence of  $T$  requests to  
 868  $a$  at time  $\tau$ . As  $A$  is decisive, there exists a time  $\tau_0$  so that  $A$  after time  $\tau_0$  always outputs a  
 869 fixed configuration when faced with  $T$  many requests to  $a$ . We refer to this configuration as  
 870 *default* configuration, and w.l.o.g. we assume that the default configuration is  $\{a, b\}$ .

871 Consider an input sequence family  $\mathcal{I}' := \{(a^T c)^L : L \in \mathbb{N}\}$ . Starting from  $\tau_0$ ,  $A$  outputs  
 872 the default configuration  $\{a, b\}$  after seeing any sequence of  $T$  requests to  $a$ . Hence, after  
 873  $\tau_0$ , each request to  $c$  incurs the cost at least 1, and this accumulated cost can be arbitrarily  
 874 large (it grows with the length of the sequence). The optimal solution for all inputs in  $\mathcal{I}'$  is  
 875 to move to the configuration  $\{a, c\}$  at the beginning, where all requests are free, hence the  
 876 optimal offline solution incurs a constant cost. Thus,  $A$  is not finitely competitive on  $\mathcal{I}'$ . ◀

877 We note that the above result relies upon a specific, yet natural, choice of the request  
 878 and answer set in our formulation, coinciding with the configuration set. However, other  
 879 encodings may admit competitive time-local algorithms. For a discussion on the choice of  
 880 the request and answer set and their influence on the competitive ratio, see Section E.4.

881 We note that similar arguments can establish the hardness of a wide range of metrical  
 882 task systems, and many  $k$ -server problem variants. Moreover, the reasoning can be improved  
 883 to show intractability for a wider class of randomized clocked time-local online algorithms;  
 884 the reasoning is equivalent to the proof Theorem E.1.

## 885 **C Power of Randomness**

886 In this section, we define randomized time-local algorithms. In the classic online setting,  
 887 there are two equivalent ways of describing randomized algorithms:

- 888 ■ at the start, randomly sample an algorithm from a set of deterministic algorithms, or
- 889 ■ at each step, make a random decision based on coin flips.

890 The former corresponds to *mixed strategies*, where we sample all random bits used by  
 891 the algorithm before seeing any of the input, whereas the latter corresponds to *behavioral*  
 892 *strategies*, where the algorithm generates random bits along the way as it needs them.

893 **Mixed vs. Behavioral Strategies in Time-Local Algorithms.** The above two characteri-  
 894 zations are equivalent in classic online algorithms [13]: to simulate a behavioral strategy  
 895 with a mixed strategy, we can generate an infinite sequence  $(r_i)_{i \geq 1}$  of random bit strings

in advance and use the random bits given by  $r_i$  in step  $i$ . Conversely, we can choose to flip coins only at the beginning and store the outcomes in memory and refer to them consistently at later steps.

In contrast, for time-local algorithms, the behavioral and mixed strategies differ in a way we can exploit randomness, and each type of strategy brings distinct advantages. If we use a behavioral strategy, at each step the algorithm can make coin flips that are independent of the previous coin flips. This enables algorithmic strategies that can e.g. break ties in an independent manner in successive steps. If we use a mixed strategy, we commit to a randomly chosen (consistent) strategy: the initial random choice influences all outputs. In a sense, in time-local algorithms, behavioral strategies correspond to *private randomness* available at each step  $i$ , whereas mixed strategies correspond to *shared randomness* across the whole sequence. Interestingly, in the time-local setting, it is also natural to consider a *combination of both*: we choose a behavioral time-local strategy at random.

With this in mind, we arrive at three natural definitions of randomized time-local algorithms:

- *behavioral strategy* time-local algorithms,
  - *mixed strategy* time-local algorithms, and
  - *general strategy* time-local algorithms that use a combination of both.
- We now give formal definitions for each class of randomized time-local algorithms.

► **Definition C.1** (behavioral local algorithms). *A behavioral  $[a, b]$ -local algorithm is given by the sequence of maps  $(A_i)_{i \geq 1}$  of the form  $A_i: X^{a+b} \times [0, 1) \rightarrow Y$ , where the output is given by*

$$y_i = A_i(x_{i-a}, \dots, x_{i+b}, r_i),$$

where  $(r_i)_{i \geq 1}$  is a sequence of i.i.d. real values sampled uniformly from the unit range. If  $A_i = A_j$  for all  $i, j$ , then the algorithm is unclocked. Otherwise, it is clocked.

► **Definition C.2** (mixed local algorithms). *Let  $\mathcal{D}$  be a nonempty set of (deterministic)  $[a, b]$ -local algorithms. A mixed  $[a, b]$ -local algorithm over  $\mathcal{D}$  is a probability measure  $A: \mathcal{D} \rightarrow [0, 1]$  over  $\mathcal{D}$ . The output of  $A$  on input  $\mathbf{x}$  is the random vector  $\mathbf{y} = P(\mathbf{x})$ , where  $P$  is a deterministic time-local algorithm sampled from  $\mathcal{D}$  according to  $A$ . If  $\mathcal{D}$  is a subset of all unclocked  $[a, b]$ -local algorithms, then  $A$  is unclocked. If  $\mathcal{D}$  is a subset of all clocked  $[a, b]$ -local algorithms, then  $A$  is clocked.*

► **Definition C.3** (general randomized local algorithms). *A general randomized unclocked  $[a, b]$ -local algorithm is a mixed  $[a, b]$ -local algorithm over the set of unclocked behavioral  $[a, b]$ -local algorithms.*

Observe that in the case of *clocked* algorithms, general randomized time-local algorithms coincide with mixed clocked time-local algorithms, as the former can be simulated by the latter. To this end, we can generate an infinite sequence  $(r_i)_{i \geq 1}$  of random bit strings in advance and store them in functions  $A_i$  of the deterministic clocked  $[a, b]$ -local algorithms, and use the random bits given by  $r_i$  in step  $i$ . However, this is not the case with unclocked algorithms: as time-local algorithms do not have memory to store past random outcomes, it is impossible to directly simulate mixed time-local algorithms by a behavioral time-local algorithm that flips coins only at the beginning.

**Adversaries and the Expected Competitive Ratio.** We naturally extend the notion of competitiveness to randomized algorithms. For randomized algorithms, the answer sequence

and the cost of an algorithm is a random variable. We will abuse the notation slightly to let  $\mathbf{y} = A(\mathbf{x})$  denote the random output generated by a randomized algorithm  $A$  on input  $\mathbf{x}$ .

We say that a randomized online algorithm  $A$  for a game defined with cost functions  $(f_n)_{n \geq 1}$  is *c-competitive* if

$$\mathbb{E}[f_n(\mathbf{x}, A(\mathbf{x}))] \leq c \cdot \text{OPT}(\mathbf{x}) + d$$

for any input sequence  $\mathbf{x}$  and a fixed constant  $d$ . The input sequence and the benchmark solution  $\text{OPT}$  is generated by an adversary. We distinguish between the notion of competitiveness against various adversaries, having different knowledge about  $A$  and different knowledge while producing the solution  $\text{OPT}$ . Competitive ratios for a given problem may vary depending on the power of the adversary.

An *oblivious offline adversary* must produce an input sequence in advance, merely knowing the description of the algorithm it competes against (in particular, it may have access to probability distributions that the algorithm uses, but not the random outcomes), and pays an optimal offline cost for the sequence. An *adaptive online adversary* produces an input sequence based on the actions of the algorithm, and serves this request sequence online. An *adaptive offline adversary* produces an input sequence based on the actions of the algorithm, and pays an optimal offline cost for the sequence. For a comprehensive overview of adversary types, see [13].

Later in this paper, we present time-local algorithms that are competitive against the oblivious offline (cf. Section E.2) and the adaptive online adversary (cf. Section E.4). We note an interesting question regarding the adaptive offline adversary in the time-local setting. A well-known result in classic online algorithms states that if there exists a  $c$ -competitive randomized algorithm against it, then there exists a deterministic  $c$ -competitive algorithm, for any  $c$  [8]. Does the existence of a competitive randomized time-local algorithm against the adaptive offline adversary imply the existence of any competitive deterministic *time-local* algorithm?

## D Automated Algorithm Synthesis

In this section, we describe a technique for automated design of time-local algorithms for local optimization problems. This technique allows us to automatically obtain both upper and lower bounds for unclocked time-local algorithms. In particular, for deterministic algorithms, we can synthesize *optimal* algorithms. We also discuss how to extend our approach to randomized algorithms. As our case study problem, we use the simplified variant of online file migration.

### D.1 Overview of the Approach

We now assume that the input and output sets  $X$  and  $Y$  are finite. Recall that an unclocked time-local algorithm that has access to last  $T$  inputs is given by a map  $A: X^T \rightarrow Y$ . The synthesis task is as follows: given the length  $T \in \mathbb{N}$  of the input horizon, find a map  $A$  that minimizes the competitive ratio. For simplicity of presentation, we will ignore short instances of length  $n < T$ , as short input sequences do not influence the competitive ratio.

**The Synthesis Method.** The high-level idea of our synthesis approach is simple:

1. Iterate through all of the algorithm candidates in the set  $\mathcal{A} = \{X^T \rightarrow Y\}$ .
2. Compute the competitive ratio  $c(A)$  for each algorithm  $A \in \mathcal{A}$ .

981    3. Choose the algorithm  $A$  that minimizes the competitive ratio.  
 982    Given that the input and output sets  $X$  and  $Y$  are finite, the set  $\mathcal{A}$  of algorithms is also  
 983    finite: there are exactly  $|Y|^{|X|^T}$  algorithms we need to check.

984    **Evaluating the Competitive Ratio.** Obviously, the challenging part is implementing the  
 985    second step, i.e., computing the competitive ratio of a given algorithm  $A$ . A priori it may  
 986    seem that we would need to consider infinitely many input strings in order to determine the  
 987    competitive ratio of the algorithm. However, for any local optimization problem  $\Pi$  with finite  
 988    input and output sets, it turns out that we can capture the competitive ratio by analyzing a  
 989    finite combinatorial object.

990    We show that for any time-local algorithm  $A$ , we can construct a (finite) weighted,  
 991    directed graph  $G(\Pi, A)$  that captures the costs of output sequences as walks in  $G(\Pi, A)$ . The  
 992    cost of any unlocked time-local algorithm on adversarial input sequences can be obtained  
 993    by evaluating the weight of all cycles defined in this graph  $G(\Pi, A)$ .

## 994    D.2    Evaluating the Competitive Ratio of an Algorithm

995    We will now describe how to construct the graph  $G(\Pi, A)$  for a given local optimization  
 996    problem  $\Pi$  and an unlocked local algorithm  $A$ . For the sake of simplicity, we only consider  
 997    the sum aggregation function; the construction for min aggregation is defined analogously.  
 998    Let  $r \in \mathbb{N}$  be the horizon of the local optimization problem  $\Pi$ ,  $v$  the valuation function of  $\Pi$ ,  
 999    and  $A: X^T \rightarrow Y$  be the time-local algorithm. To avoid unnecessary notational clutter, we  
 1000    describe the construction for  $r = 1$ ; however, the construction is straightforward to generalize.

1001    **The Dual de Bruijn Graph.** We construct a directed graph  $G = (V, E)$  on the set of vertices  
 1002     $V = X^T \times Y$ . For any  $\mathbf{x} = (x_1, \dots, x_k)$ , we define  $s(\mathbf{x}, a) = (x_2, \dots, x_k, a)$  to be the successor  
 1003    of  $\mathbf{x}$  on  $a$ . For each vertex  $(\mathbf{a}, y) \in V$ , there is a directed edge towards the vertex  $(\mathbf{a}', y') \in V$ ,  
 1004    where for all  $y' \in Y$ ,  $\mathbf{a}' = s(\mathbf{a}, x)$  and  $x \in X$ . Note that there are self-loops in this graph.

1005    The idea is that for any sufficiently long input  $n \geq T$ , an input sequence  $\mathbf{x} \in X^n$  and an  
 1006    output sequence  $\mathbf{y} \in Y^n$  define a walk  $\rho(\mathbf{x}, \mathbf{y})$  in the graph  $G$ . After the time step  $i \geq T$ , we are  
 1007    at vertex  $(x_{i-T-1}, \dots, x_i, y_i) \in V$  and the next vertex is given by  $(x_{i-T}, \dots, x_{i+1}, y_{i+1}) \in V$ .  
 1008    In particular, from any walk  $\rho$  we can obtain the following sequences:

- 1009    ■ an input sequence  $\mathbf{x}(\rho) = (x_1, \dots, x_n) \in X^n$ ,
  - 1010    ■ some (possibly optimal) solution  $\mathbf{y}^*(\rho) = (y_1, \dots, y_n)$  for  $\mathbf{x}(\rho)$ , and
  - 1011    ■ the output  $y(\rho) = A(\mathbf{x}(\rho))$  given by the algorithm on  $\mathbf{x}(\rho)$ .
- 1012    Vice versa, any pair of input  $\mathbf{x}$  and output  $\mathbf{y}^*$  sequences defines a walk  $\rho(\mathbf{x}, \mathbf{y}^*)$  in  $G$ .

1013    **Assigning the Costs.** For each edge  $e \in E$  in the graph, we assign *two* costs for the edge: the  
 1014    first describes the cost paid by some (possibly optimal) output, and the second, the cost paid  
 1015    by the algorithm  $A$ . Recall that for a local optimization problem  $\Pi$ , the costs are given by the  
 1016    valuation function  $v: X^{r+1} \times Y^{r+1} \rightarrow \mathbb{R} \cup \{\infty\}$ . Consider an edge  $e = ((\mathbf{a}, \mathbf{b}), (\mathbf{a}', \mathbf{b}')) \in E$ ,  
 1017    where  $\mathbf{a}' = (a_2, \dots, a_{T+r-1}, x)$  and  $\mathbf{b}' = (b_2, \dots, b_r, y)$  for some  $x \in X$  and  $y \in Y$ .

1018    We now define the *adversary cost*  $w(e)$  and *algorithm cost*  $q(e)$  of the edge  $e$ . For the  
 1019    sake of simplicity, we now only describe the simpler case of  $r = 1$ ; the costs generalize to  
 1020    general  $r > 1$  by applying the definitions of value functions given in Section 3.1. We define

1021     $w(e) = v(a_T, x, b'_1, b'_2)$  is the cost paid on some output  $b'_r = y$  on input  $x$ ,

1022     $q(e) = v(a_T, x, b'_1, A(\mathbf{a}))$  is the cost paid by the algorithm on input  $x$ ,

1023    where  $v: X \times Y \rightarrow \mathbb{R} \cup \{\infty\}$  is the valuation function of the problem  $\Pi$ .

1025 **The Cost Ratio of a Walk.** Finally, for any walk  $\rho = (v_1, \dots, v_k)$  in  $G$ , we define

$$1026 \quad w(\rho) = \sum_{i=1}^{k-1} w(v_i, v_{i+1}), \quad q(\rho) = \sum_{i=1}^{k-1} q(v_i, v_{i+1}).$$

1027 Here  $w(\rho)$  and  $q(\rho)$  define the *total* adversary and algorithm costs for the walk  $\rho$ . The *cost*  
1028 *ratio* of a walk  $\rho$  is defined as

$$1029 \quad r(\rho) = \begin{cases} q(\rho)/w(\rho) & \text{if } w(\rho) > 0 \\ 1 & \text{if } q(\rho) = w(\rho) = 0 \\ \infty & \text{otherwise.} \end{cases}$$

1030 That is, on input  $\mathbf{x}(\rho)$  the algorithm  $A$  will pay a cost of  $q(\rho) + O(1)$ , whereas the optimum  
1031 solution has cost at most  $w(\rho) + O(1)$ ; there is a constant overhead on the costs since we  
1032 ignore the costs incurred during the first  $T - 1 + r = O(1)$  inputs.

1033 **Bounding the Competitive Ratio.** We now show that we can compute the competitive  
1034 ratio of the algorithm  $A$  using the graph  $G = G(\Pi, A)$ . We say a walk  $\rho = (v_1, \dots, v_k) \in V^k$   
1035 is *closed* if its starts and ends in the same vertex  $v_1 = v_k$ . A *directed cycle* is a closed walk  
1036 that is non-repeating, i.e.,  $v_i \neq v_j$  for all  $1 \leq i < j < k$ .

1037 **► Theorem D.1.** *The competitive ratio of algorithm  $A$  is  $\max\{r(\rho) : \rho \text{ is a directed cycle of } G\}$ .*

1038 Figure 4 gives an example of the dual de Bruijn graph for the online file migration problem  
1039 (Example 3.1) and an algorithm with local horizon  $T = 2$ .

1040 To prove the above theorem, we introduce three lemmas and the following definitions. A  
1041 *closed extension* of a walk  $\rho$  is a closed walk  $\rho'$  that contains  $\rho$  as a prefix. A *subwalk*  $\rho'$  of a  
1042 walk  $\rho = (v_1, \dots, v_k)$  is a subsequence  $(v_i, \dots, v_j)$  for some  $1 \leq i \leq j \leq k$ . A *decomposition*  
1043 *of*  $\rho$  into  $L$  subwalks is a sequence of subwalks  $\rho_1, \dots, \rho_L$  of  $\rho$  such that their concatenation  
1044  $\rho = \rho_1 \cdots \rho_L$ .

1045 **► Lemma D.2.** *Let  $\rho$  be a walk in  $G$ . For any decomposition of  $\rho$  into  $L$  subwalks  $\rho_1 \cdots \rho_L$ ,  
1046 there exists some  $1 \leq i \leq L$  such that  $r(\rho_i) \geq r(\rho)$ .*

1047 **Proof.** Let  $\pi$  be a permutation on  $\{1, \dots, L\}$  and  $\tau_i = \rho_{\pi(i)}$  such that

$$1048 \quad r(\tau_1) \leq r(\tau_2) \leq \dots \leq r(\tau_L).$$

1049 Moreover, for  $1 \leq i \leq L$  we define  $r(\tau_i) = q_i/w_i$ , where  $q_i = q(\tau_i)$  and  $w_i = w(\tau_i)$ . We use  
1050 the shorthands  $Q(i) = \sum_{j=1}^i q_j$  and  $W(i) = \sum_{j=1}^i w_j$ . Note that for the aggregate cost ratio  
1051 for a local optimization problem using the sum as its aggregation function gives that

$$1052 \quad r(\rho) = \frac{Q(L)}{W(L)} = \frac{\sum_{j=1}^L q_j}{\sum_{j=1}^L w_j}.$$

1053 We now show by induction that for all  $1 \leq i \leq L$  we have that

$$1054 \quad r(\tau_i) = \frac{q_i}{w_i} \geq \frac{Q(i)}{W(i)}.$$

1055 Observe that this implies that  $r(\rho_{\pi(L)}) = r(\tau_L) \geq Q(L)/W(L) = r(\rho)$ .

1056 The base case  $i = 1$  is vacuous. For the inductive step, assume that the claim holds for  
 1057 some  $1 \leq i < L$ . For the sake of contradiction, assume that claim does not hold for  $i + 1$ , i.e.,

$$1058 \quad r(\tau_{i+1}) = \frac{q_{i+1}}{w_{i+1}} < \frac{Q(i+1)}{W(i+1)}.$$

1059 By rearranging the terms, we get

$$1060 \quad \frac{Q(i+1)w_{i+1} - W(i+1)q_{i+1}}{W(i+1)} > 0,$$

1061 which in turn implies that  $Q(i+1) \cdot w_{i+1} > W(i+1) \cdot q_{i+1}$  holds. Now observing that

$$1062 \quad Q(i)w_{i+1} + q_{i+1}w_{i+1} = Q(i+1)w_{i+1} > W(i+1) \cdot q_{i+1} = W(i)q_{i+1} + w_{i+1}q_{i+1},$$

1063 we get that

$$1064 \quad r(\tau_{i+1}) = \frac{q_{i+1}}{w_{i+1}} < \frac{Q(i)}{W(i)} \leq \frac{q_i}{w_i} = r(\tau_i),$$

1065 where the second inequality follows from the induction assumption. However, this contradicts  
 1066 the fact that  $\tau_1, \dots, \tau_L$  were ordered according to increasing cost ratio.  $\blacktriangleleft$

1067 **► Lemma D.3.** *Let  $\rho$  be a directed cycle in  $G$ . The competitive ratio of  $A$  is at least  $r(\rho)$ .*

1068 **Proof.** Recall that the cycle defines an input sequence  $\mathbf{x} = \mathbf{x}(\rho)$ . By definition, the algorithm  
 1069 has cost at least  $q(\rho)$  on this input sequence, whereas the optimum solution has cost at  
 1070 most  $w(\rho) + d$  for some constant  $d$ . Thus, the algorithm has a cost of at least  $q(\rho) \geq$   
 1071  $r(\rho)(w(\rho) + d) \geq r(\rho) \cdot \text{OPT}(\mathbf{x}) + O(1)$ .  $\blacktriangleleft$

1072 **► Lemma D.4.** *If the competitive ratio of  $A$  is greater than  $c + \varepsilon$  for some  $\varepsilon > 0$ , then there  
 1073 exists a directed cycle  $\rho$  in  $G$  with cost ratio  $r(\rho) > c$ .*

1074 **Proof.** For any given walk  $\rho$  in  $G$ , let  $\hat{\rho}$  be the shortest closed extension of  $\rho$  that minimizes  
 1075 the cost of  $\mathbf{y}^*(\hat{\rho})$ . Note there may be multiple shortest closed extensions, so we pick the  
 1076 one with the cheapest adversarial cost. We let  $\hat{\rho} \setminus \rho$  denote the suffix of  $\hat{\rho}$  that satisfies  
 1077  $\hat{\rho} = \rho \cdot (\hat{\rho} \setminus \rho)$ . Define

$$1078 \quad \delta = \max\{w(\hat{\rho} \setminus \rho) : \rho \text{ is a walk in } G\}.$$

1079 Note that  $\delta$  is a constant, since  $\hat{\rho}$  is a minimal closed extension of  $\rho$  and  $G$  is finite.

1080 Let  $\mathbf{x}$  be an input sequence and  $\mathbf{y}^*$  an *optimal* output sequence. For the walk  $\rho = \rho(\mathbf{x}, \mathbf{y}^*)$ ,  
 1081 we have that

$$1082 \quad r(\hat{\rho}) = \frac{q(\hat{\rho})}{w(\hat{\rho})} \geq \frac{q(\rho) + q(\hat{\rho} \setminus \rho)}{w(\rho) + w(\hat{\rho} \setminus \rho)} \geq \frac{q(\rho) + \delta}{w(\rho) + \delta},$$

1083 since  $w(\rho) \leq q(\rho)$ , as the cost of the algorithm is never less than the cost of the optimal solution  
 1084  $\mathbf{y}^*$ . Asymptotically, as the length of the walk goes to infinity, we have that  $r(\hat{\rho}) = r(\rho) - o(1)$ .  
 1085 In particular, for any constant  $\varepsilon_0 > 0$  we can find  $n_0$  such that all input sequences  $\mathbf{x}$  of length  
 1086  $n \geq n_0$ , the walk  $\rho = \rho(\mathbf{x}, \mathbf{y}^*)$  given by  $\mathbf{x}$  and the optimal output sequence  $\mathbf{y}^*$ , satisfies

$$1087 \quad r(\hat{\rho}) \geq r(\rho) - \varepsilon_0.$$

1088 By assumption  $A$  had a competitive ratio of at least  $c + \varepsilon$ . We can pick a sufficiently long  
 1089 input sequence  $\mathbf{x}$  and an optimal solution  $\mathbf{y}^*$  such that  $\rho = \rho(\mathbf{x}, \mathbf{y}^*)$  satisfies

$$1090 \quad r(\hat{\rho}) \geq r(\rho) - \varepsilon_0 \geq \frac{f_n(\mathbf{x}, A(\mathbf{x}))}{\text{OPT}(\mathbf{x})} - \varepsilon' - \varepsilon_0 \geq c - \varepsilon' - \varepsilon_0 + \varepsilon > c,$$

1092 where  $f_n(\mathbf{x}, A(\mathbf{x}))$  denotes the cost of the algorithm  $A$  on input  $\mathbf{x}$  and  $\varepsilon_0$  and  $\varepsilon'$  are appro-  
 1093 priately chosen constants. Thus, we have now obtained a closed walk  $\hat{\rho}$  with  $r(\hat{\rho}) \geq c$ . Since  
 1094 we can decompose  $\hat{\rho}$  into a sequence  $\hat{\rho}_1, \dots, \hat{\rho}_K$  of directed cycles, by applying Lemma D.2  
 1095 we get that some directed cycle  $\hat{\rho}_i$  satisfies  $r(\hat{\rho}_i) > c$ , as claimed. ◀

1096 **Proof of Theorem D.1.** The above two lemmas yield that the competitive ratio of  $A$  is  
 1097 ■ at least as large as the cost-ratio of some directed cycle in  $G$  (Lemma D.3), and  
 1098 ■ at most as large as the cost-ratio of some directed cycle in  $G$  (Lemma D.4).  
 1099 Thus, the directed cycle with the highest cost-ratio determines the competitive ratio of the  
 1100 algorithm  $A$ . Since the graph  $G$  is finite, it suffices to check all directed cycles of  $G$  to  
 1101 determine the competitive ratio of  $A$ .

## 1102 D.3 Optimizations

1103 We now overview few techniques for optimizing the synthesis for our case study problem of  
 1104 online file migration.

1105 We can reduce the amount of computation needed to find the best algorithm  $A$  for a  
 1106 fixed  $T$  and  $\alpha$ , by eliminating some algorithms. For example, we can often quickly identify  
 1107 some simple property of  $G$  that immediately disqualifies an algorithm candidate.

1108 **The Role of Self-Loops.** If the competitive ratio of  $A$  is  $K$ , then the cost-ratio of any  
 1109 directed cycle has to be at most  $K$ . In particular, the cost-ratio of any directed cycle has to  
 1110 be finite. So we can directly eliminate all cases in which there is a cycle  $\rho$  with adversary-cost  
 1111  $w(\rho) = 0$  and a positive algorithm-cost  $q(\rho) > 0$ . For example, we can apply this reasoning to  
 1112 self-loops in the graph  $G$ . If the adversary-cost of a self-loop is zero, then the algorithm-cost  
 1113 of the same loop has to be also zero. It follows that e.g. we must have  $A(0, \dots, 0) = 0$   
 1114 and  $A(1, \dots, 1) = 1$  for any algorithm  $A$ . In the case of  $T = 3$ , this reduces the number of  
 1115 algorithms that need to be checked from  $2^8 = 256$  to only  $2^{2^3-2} = 2^6 = 64$  instead.

1116 **Detecting Heavy Cycles.** When searching for algorithms with best competitive ratio, it  
 1117 is useful to keep track of the best cost-ratio found so far: when checking a new algorithm  
 1118 candidate  $A$  and its corresponding graph  $G$ , we can first check small cycles of length at most  
 1119  $L$  to see if any such cycle has cost-ratio larger than the best found cost-ratio for any other  
 1120 algorithm so far. If we encounter a cycle  $\rho$  with cost-ratio  $r(\rho)$  that is larger or equal than  
 1121 the competitive ratio of some previously considered algorithm  $A'$ , then we know that the  
 1122 competitive ratio of  $A$  is larger or equal than that of  $A'$ . Thus, we can immediately disregard  
 1123  $A$  and move on to check the next possible algorithm candidate.

1124 Indeed, it turns out that in many cases, cycles with large cost-ratio are already found  
 1125 when examining only short cycles. However, if high-cost short cycles are not found, we can  
 1126 always fall back to an exhaustive search that checks all cycles.



## 1127 D.4 On the Synthesis of Randomized Algorithms

1128 We note that we can extend our approach to the synthesis of randomized algorithms as  
 1129 well. Here, the synthesis bounds the *expected* competitive ratio of the algorithm against an  
 1130 oblivious randomized adversary.

1131 We model randomized algorithms as follows. Instead of considering maps  $A: \{0, 1\}^T \rightarrow$   
 1132  $\{0, 1\}$ , we consider maps  $A: \{0, 1\}^T \rightarrow [0, 1]$ , where  $A(\mathbf{a})$  gives the *probability* that  $A$  outputs 1  
 1133 upon seeing the sequence  $\mathbf{a} \in X^T$  of last  $T$  inputs. Thus,

$$\begin{aligned} 1134 \quad A(\mathbf{a}) &= \Pr[A \text{ outputs } 1 \text{ on input } \mathbf{a} \in X^T] \\ 1135 \quad 1 - A(\mathbf{a}) &= \Pr[A \text{ outputs } 0 \text{ on input } \mathbf{a} \in X^T]. \end{aligned}$$

1137 We assign the algorithm cost  $q(e)$  for any edge  $(\mathbf{a}, y)$  to  $(s(\mathbf{a}, x), y')$  as follows:

1138 ■ On a mismatch, the algorithm pays the cost

$$1139 \quad q_{\text{mismatch}}(e) = \begin{cases} 1 - A(\mathbf{a}) & \text{if } x = 1 \\ A(\mathbf{a}) & \text{otherwise.} \end{cases}$$

1140 ■ The switching cost is given by

$$1141 \quad q_{\text{switch}}(e) = \alpha \cdot [A(\mathbf{a}) \cdot (1 - A(s(\mathbf{a}, a))) + (1 - A(\mathbf{a})) \cdot A(s(\mathbf{a}, x))].$$

1142 ■ The total cost is  $q(e) = q_{\text{mismatch}}(e) + q_{\text{switch}}(e)$ .

1143 We calculate the adversary-cost in the same manner as we do in the deterministic model.  
 1144 That is our adversary always outputs 0 or 1 (but not, e.g. 0.6). Thus, the graph  $G$  will have  
 1145 the same structure as in the deterministic case.

1146 Since there are uncountably many possible randomized algorithms  $A$  for any  $T$ , we instead  
 1147 discretize the probability space into finitely many segments. Thus, we cannot guarantee that  
 1148 we find optimal randomized algorithms. Nevertheless, this method can be used to obtain  
 1149 synthesized algorithms that beat the deterministic algorithms.

## 1150 D.5 Results

1151 We now give some results for the online file migration problem obtained using the synthesis  
 1152 approach. First, we discuss algorithms with small values of  $T = 1, 2, 3$  and then provide  
 1153 observations for  $T = 4$  and  $T = 5$ .

### 1154 D.5.1 Synthesized Algorithms for $T = 1, 2, 3$

1155 Table 2 summarizes results for  $T = 1, 2, 3$  and  $0.1 \leq \alpha \leq 1.6$ . For deterministic algorithms,  
 1156 we list the competitive ratios of the *optimal* deterministic algorithms for the given values of  
 1157 parameters  $T$  and  $\alpha$ .

1158 For randomized algorithms, we list the best competitive ratios found by the synthesis  
 1159 method for the given values of  $T$  and  $\alpha$ . As discussed, the search for randomized algorithms  
 1160 was conducted in a discretized search space, so it is possible that some randomized algorithms  
 1161 with better competitive ratios may have been missed by the search method.

1162 **The Power of Randomness.** Note that already with  $T = 2$  we can obtain algorithms with  
 1163 strictly better competitive ratios when randomness is used. Moreover, with only  $T = 3$ , we  
 1164 are able to obtain randomized algorithms with competitive ratio  $< 3$  (e.g., when  $\alpha = 1.0$ ).  
 1165 This is strictly better than *any* (non-time-local) deterministic algorithm for  $\alpha = 1$ . Table 3

gives an example of such an algorithm that achieve competitive ratio of roughly 2.67 for  $T = 3$  and  $\alpha = 1$ .

After checking all cycles in the constructed dual de Bruijn graph, the cycle with the maximum cost-ratio (cost-ratio of about 2.67) happens to be the following:

- Last  $T$  inputs: 000, adversary output: 0.
- Last  $T$  inputs: 001, adversary output: 0.
- Last  $T$  inputs: 011, adversary output: 0.
- Last  $T$  inputs: 110, adversary output: 0.
- Last  $T$  inputs: 100, adversary output: 0.

### D.5.2 The Case of $T = 4$

For  $T = 4$  we can obtain better deterministic algorithms than with  $T = 3$ . Interestingly, we can find several optimal algorithms for the case  $\alpha = 1$ : even a full-history deterministic online algorithm cannot achieve a better competitive ratio. Table 4 lists all the 3-competitive algorithms that exist for parameter values of  $T = 4$ ,  $\alpha = 1$ . This shows that even very simple time-local algorithms can perform well compared to classic online algorithms. Table 2 contains some of the results for  $T = 4$  and  $0.1 \leq \alpha \leq 1.5$ .

### D.5.3 Negative Results for $T = 5$

Since the number of cycles to be checked increases exponentially in  $T$ , we were not able to obtain any positive results for the case of  $T = 5$ . However, negative results could still be obtained, since verifying for a certain lower bound does not require to check all the cycles for all the algorithms. Instead, it is sufficient to find at least one cycle with a large enough cost-ratio to disregard a certain algorithm and move on to the next one. We were able to obtain the following results:

► **Observation D.5.** *With parameter values of  $\alpha = 1$  and  $T = 5$ , the best competitive ratio remains 3. That is, for each deterministic algorithm, after the dual de Bruijn graph has been constructed, there is a cycle with a cost-ratio of at least 3.*

► **Observation D.6.** *No algorithm with ratio  $< 3.1$  exists for parameter values of  $T = 5$  and  $\alpha = 1.1$ .*

► **Observation D.7.** *No algorithm with ratio  $< 3.2$  exists for parameter values of  $T = 5$  and  $\alpha = 1.2$ .*

## E Case Study: Online File Migration

We study a variant of online file migration (defined in Section 1.2) in the time-local setting. The case study serves three purposes:

- to show an example of a problem that admits competitive algorithms in the time-local setting,
- to highlight the challenges of algorithm design and analysis present in the time-local setting and propose techniques used to deal with them,
- to study the influence of limiting the visible horizon size on degradation of the competitive ratio.

■ **Table 2** The best competitive ratios for some values of  $\alpha$  and  $T$ ; see also Figure 2.

$\alpha$	$T = 1$	$T = 2$		$T = 3$		$T = 4$
	deterministic	deterministic	randomized	randomized	randomized	deterministic
0.1	11	11	11	11	11	11
0.2	6	6	6	6	6	6
0.3	4.333	4.333	4.333	4.333	4.333	4.333
0.4	3.5	3.5	3.5	3.5	3.5	3.5
0.5	3	3	3	3	3	3
0.6	3.2	3.2	3.006	3.2	2.934	3.2
0.7	3.4	3.4	3.055	3.4	2.864	3.4
0.8	3.6	3.6	3.2	3.6	2.797	
0.9	3.8	3.8	3.35	3.8	2.734	3.222
1.0	4	4	3.5	4	2.672	3
1.1	4.2	4.2	3.65	4.2	2.772	3.1
1.2	4.4	4.4	3.8	4.4	2.872	
1.3	4.6	4.6	3.95	4.6	2.986	3.3
1.4	4.8	4.8	4.1	4.8	3.088	
1.5	5	5	4.25	5	3.188	3.5
1.6	5.2	5.2	4.4	5.2	3.288	

■ **Table 3** A randomized algorithm for  $T = 3$  and  $\alpha = 1$  with expected competitive ratio  $\approx 2.67$ .

Last $T$ inputs	The probability to output 1
000	0
001	0.3309
010	0.2711
011	1
100	0
101	0.7289
110	0.6691
111	1

■ **Table 4** Three 3-competitive algorithms for  $T = 4$ ,  $\alpha = 1$ .

Last $T$ inputs	Output		
	$A_1$	$A_2$	$A_3$
...0000	0	0	0
...0001	0	0	0
...0010	0	0	0
...0011	1	1	1
...0100	0	0	0
...0101	0	0	1
...0110	1	1	1
...0111	1	1	1
...1000	0	0	0
...1001	0	0	0
...1010	1	0	1
...1011	1	1	1
...1100	0	0	0
...1101	1	1	1
...1110	1	1	1
...1111	1	1	1

We assume the following encoding of the problem: on each request, a time-local algorithm takes the current visible horizon as input and outputs the next location of the file. For a discussion on alternative encodings of this problem, see Appendix E.4. Unless stated explicitly, we assume that the migration cost  $\alpha$  is at least 1.

We start by providing insights into techniques which are useful for studying online problems in time-local setting. Next, we present a lower bound showing that the degradation of the competitive ratio is inevitable with the decrease of  $T$ , even with access to a global clock and using randomization. Then, we discuss an adaptation of a well-known randomized algorithm to the time-local setting, and design a competitive deterministic algorithm for 2-node networks.

**Techniques for designing time-local algorithms.** The challenges in designing time-local algorithms come from two sources: (1) the algorithm can make decisions based on the most recent input history only, and (2) the algorithm is unaware of its current configuration. Note that the latter challenge is not present in *memoryless* online algorithms [20]. To tackle these challenges, we highlight a useful technique of *tracking* distinguished subsequences of the input as they recede in the visible horizon. Implementing consistent tracking is simpler in the clocked setting; we study an example where we track requests issued at certain points in time, initially chosen uniformly at random. Tracking is significantly more challenging to implement in non-clocked setting. Without knowing the temporal position of requests, requests from the same node are indistinguishable. We overcome this limitation by tracking distinguishable subsequences of requests instead of single requests, and we specify it in Section E.3.

**Lower bounds under time-local setting.** We are able to reason about a time-local algorithm's performance on different sequences that share the same subsequences of  $T$  requests. The algorithm is a function of the last  $T$  requests, hence its output on one input is identical

to the output on the second. A common lower bound design technique for classic online algorithms involves reasoning about distributions over inputs with Yao's principle. Instead, a simpler reasoning about a time-local algorithm may be sufficient in certain situations. We may argue about the performance of the algorithm on various relevant inputs independently, and come to conclusions about its performance on every input.

## E.1 A Lower Bound

In this section, we present a lower bound for time-local algorithms for online file migration that shows an inevitable degradation of the competitive ratio when the visible horizon is limited. The following lower bound assumes the length of the visible horizon is given. Hardness for simpler settings is implied, in particular for non-clocked time-local algorithms. In the subsequent Sections E.2 and E.3, we present randomized and deterministic algorithms that asymptotically match this lower bound.

► **Theorem E.1.** *Fix any randomized  $T$ -time-local clocked algorithm  $A$  for online file migration for networks with at least 2 nodes. Assume that the file size is  $\alpha$ , and  $\alpha \geq T$ . If  $A$  is  $c$ -competitive against an oblivious offline adversary, then  $c \geq 2\alpha/T$ .*

**Proof.** To state  $A$ 's properties, we consider two infinite sequence of requests,  $0^*$  and  $1^*$ . Later we will reason about  $A$ 's performance on finite sequences. We say that a deterministic algorithm is *resisting* if for each  $t$  there exists a time  $t' > t$  such that the algorithm either outputs 1 at  $t'$  when faced with  $0^*$ , or outputs 0 at  $t'$  when faced with  $1^*$ . A time-local algorithm may be resisting if it has access to a global clock.

Recall that  $A$  is a distribution over deterministic clocked algorithms. First, assume that  $A$  has a resisting strategy in its support. Consider two input sequence families,  $\mathcal{I}_0 := \{0^L : L \in \mathbb{N}\}$  and  $\mathcal{I}_1 := \{1^L : L \in \mathbb{N}\}$ . Note that  $A$  may incur an arbitrarily large cost on requests from either of these families of inputs, say  $\mathcal{I}_0$ , due to an arbitrary number of 0-requests served in the configuration 1. The cost of an optimal offline solution on such sequences is constant: an offline algorithm moves the file at the beginning to the only node that requests the file. We conclude that the competitive ratio of  $A$  can be arbitrarily large on inputs from  $\mathcal{I}_0$  or  $\mathcal{I}_1$ .

For the rest of this proof, we assume that  $A$  does not have any resisting strategy in its support. A crucial observation is that for a fixed deterministic time-local algorithm, its output on  $0^*$  (resp.  $1^*$ ) for any time  $\tau$  determines its output on other sequences that contain a sequence of  $T$  requests to 0 (resp. 1) at time  $\tau$ . As  $A$  does not have a resisting strategy in its support, there exists a time  $\tau_{det}$  such that after  $\tau_{det}$ , all strategies in  $A$ 's support always output  $b$  when faced with  $T$  many requests to  $b$ , for  $b \in \{0, 1\}$ .

Consider an input  $\sigma = (1^T 0^T)^L$  for some  $L$  to be determined. Let  $\sigma'$  be the subsequence of  $\sigma$  starting from the first 0-request that comes after  $\tau_{det}$ . Fix any optimal offline algorithm  $\text{OPT}$  for  $\sigma$ . For  $\sigma \setminus \sigma'$ , we simply claim  $A(\sigma \setminus \sigma') \geq \text{OPT}(\sigma \setminus \sigma')$ , where  $A(\cdot)$  and  $\text{OPT}(\cdot)$  denotes the cost of the algorithm  $A$  and an optimal offline algorithm, respectively. To analyze  $\sigma'$ , we split it into subsegments of  $(1^T 0^T)$  that we refer to as *phases*. As no strategy in  $A$ 's support is resisting, and requests from  $\sigma'$  come after  $\tau_{det}$ ,  $A$ 's behavior on  $\sigma'$  is determined: it must output  $b$  faced with  $b$ -uniform sequence, for  $b \in \{0, 1\}$ . Hence, in each phase  $A$  incurs cost  $2\alpha$ . On the other hand, in each phase  $\text{OPT}$  incurs cost at least  $T$  — recall that  $T \leq \alpha$ , thus either it migrated during the phase and paid  $\alpha \geq T$  already, or did not migrate and paid for either 0-requests or 1-requests. Summing up the above observations and assuming  $\sigma'$  has

length  $2T \cdot L'$ , we obtain

$$\frac{A(\sigma)}{\text{OPT}(\sigma)} = \frac{A(\sigma \setminus \sigma') + A(\sigma')}{\text{OPT}(\sigma \setminus \sigma') + \text{OPT}(\sigma')} \geq \frac{\text{OPT}(\sigma \setminus \sigma') + 2\alpha \cdot L'}{\text{OPT}(\sigma \setminus \sigma') + T \cdot L'}.$$

By choosing a long enough sequence  $\sigma$  (and consequently a large enough  $L'$ ), the competitive ratio can be arbitrarily close to  $2\alpha/T$ . ◀

Note that the result presented in this section implies the lower bound for online file migration on general networks (not necessarily consisting of two vertices).

## E.2 A Randomized Algorithm

In this section we discuss a randomized time-local algorithm for online file migration in general networks. The algorithm is an adaptation of an elegant randomized algorithm by Westbrook [44] as a function of the local time horizon and the global clock. We discuss the competitive ratio tradeoffs related to limited view of past requests, and we show that these tradeoffs are asymptotically optimal (cf. Section E.1).

We assume that the feasible set of answers of any algorithm is coincident with the set of nodes of the network. For a discussion on possible extensions of such setting, see Section E.4.

**Algorithm Mixed Resetting.** Fix any network  $N$ , size of the file  $\alpha$  and the size of visible  $T$ -horizon. We describe a set of deterministic strategies, each parameterized by an integer  $k \in [1, T]$ . The Mixed Resetting algorithm chooses uniformly at random one of these strategies at the beginning.

Now we describe the deterministic strategy for a fixed  $k$ . In the classic setting with stateful algorithms, the algorithm may be described as follows. It maintains a counter, initially set to  $k$ . Each time it encounters a request, the counter decreases. When the counter drops to 0, the algorithm moves the file to the node where the request comes from at the moment, and resets the counter to  $T$ . While the counter is positive, the algorithm does not change the file placement.

This deterministic strategy can be implemented in the clocked time-local setting with the last  $T$  requests visible. Let  $\tau$  be the current value of the clock, and let  $p, q$  be integers such that  $\tau = T \cdot p + q$  and  $q < T$ . The algorithm's output at  $\tau$  is  $x_{T-q}$ , the node that requested a file with the index  $T - q$  in the current visible  $T$ -horizon.

► **Theorem E.2.** *The competitive ratio of Mixed Resetting against the oblivious offline adversary is  $\max\{2 + \frac{2\alpha}{T}, 1 + \frac{T+1}{2\alpha}\}$ .*

► **Corollary E.3.** *The competitive ratio of Mixed Resetting against the oblivious offline adversary is  $1 + \phi \approx 2.62$  for  $T = \alpha + \frac{1}{2}\sqrt{20\alpha^2 - 4\alpha + 1} - 1$ , where  $\phi$  is the Golden Ratio.*

An elegant proof of this theorem was given by Westbrook [44], using a potential function argument. This yields the best currently known algorithm in general networks against the oblivious adversary; the result is not tight, and the best known lower bound against the oblivious adversary is  $2 + 1/(2\alpha)$  [20].

For larger  $T$  the competitive ratio would increase, thus in such case we simply truncate the visible horizon to its optimal value. Note that the ratio of Mixed Resetting is  $O(\alpha/T)$ , and it asymptotically matches the lower bound of  $\Omega(\alpha/T)$  from Theorem E.1. The result in this section is more general: the network may be arbitrary (more than two vertices), and the competitive ratio of the time-local algorithm remains unchanged and matches the lower bound even in arbitrary networks.

### 1315 E.3 A Deterministic Algorithm

1316 In this section, we introduce a constant competitive time-local algorithm, the *Sliding Win-*  
 1317 *dow Algorithm* (ALG for short), for the online file migration problem restricted to two nodes,  
 1318 identified by 0 and 1. The algorithm takes the last  $T$  requests as input, and it outputs a  
 1319 value in  $\{0, 1\}$ , the (new) location of the file. For each request 0 or 1 (the node requesting  
 1320 the file), ALG pays a unit cost if its last output does not match the request (i.e., if the file is  
 1321 not located at the node). In such cases, we say ALG incurs a *mismatch*. After serving the  
 1322 request, ALG may choose to migrate the file to the other node (by switching the output) at  
 1323 the cost  $\alpha$ , and in such case we say ALG *flips* its output.

1324 The algorithm scans the visible horizon looking for a distinguished subsequence of requests,  
 1325 called a *relevant window*. It decides its output based on the existence of any relevant window,  
 1326 and an invariable property of the relevant window (if any detected). After a window enters  
 1327 the visible horizon, ALG maintains its latest output as long as (i) the window is contained  
 1328 in the visible horizon (while *sliding*), and (ii) it is not succeeded by a more recent relevant  
 1329 window. ALG may flip its output once either of (i) or (ii) is no longer the case. Intuitively,  
 1330 a relevant window serves as a short-lived memory, enabling ALG to maintain the same output  
 1331 for as long as the window is contained in the visible horizon.

1332 **Sliding Window Algorithm.** We define a *b-window* for  $b \in \{0, 1\}$  as a subsequence of  
 1333 requests in which the number of  $b$ -requests is at least twice the number of  $\neg b$ -requests, where  
 1334  $\neg b = 1 - b$ . ALG outputs 1 only if the most recent  $b$ -window in the visible horizon is a  
 1335 1-window. Therefore, it outputs 1 as long as the visible horizon contains a 1-window that  
 1336 is not succeeded by a (more recent) 0-window. The 1-window slides further to the past as  
 1337 new requests arrive, until it is no longer contained in the visible horizon. At this time, ALG  
 1338 flips back to 0 (as the default output) unless there is a more recent 1-window in the visible  
 1339 horizon.

1340 We describe the sliding window algorithm formally as follows. Given any  $T \geq 6$ , let  
 1341  $\lambda := \min\{\lceil T/6 \rceil, \alpha\} \geq 1$  be a parameter. We denote a  $b$ -window of length  $3\lambda$  that enters the  
 1342 visible horizon at time  $t$  by  $\mathbf{W}_t := \sigma(t - 3\lambda, t]$ . At any time  $t$ , the algorithm ALG takes the  
 1343 visible horizon (the past  $T$  requests) as input and outputs either 0 or 1 according to the  
 1344 following rules.

1345 **Rule 1.** ALG outputs  $b \in \{0, 1\}$  if the most recent window in the visible horizon is a  $b$ -window.

1346 **Rule 2.** ALG outputs 0 if the visible horizon contains no  $b$ -window for  $b \in \{0, 1\}$ .

1347 Note that whenever ALG flips to 0, it is either because the visible horizon contains  
 1348 a 0-window (Rule 1), or because there is no  $b$ -window in the visible horizon (Rule 2).

#### 1349 E.3.1 Analysis

1350 Our analysis is based on the observation that ALG neither flips too frequently, incurring  
 1351 excessive reconfiguration cost, nor too conservatively, incurring excessive mismatches. To  
 1352 this end, we focus our attention on individual subsequences between two consecutive flips  
 1353 to 1. We show that any of these subsequences contains sufficiently many 0 and 1-request, so  
 1354 that an optimal offline algorithm incurs a cost within a factor  $O(\alpha/T)$  of ALG's cost.

1355 **Preliminaries.** We introduce auxiliary definitions and notations that we use in our analysis.  
 1356 ALG starts serving requests by outputting 0 and flips for the first time at  $t_{\text{first}}$ , which is a flip  
 1357 to 1. Let  $(l_i, r_i)$  denote the  $i$ th pair of time indexes after  $t_{\text{first}}$ , s.t. ALG flips to 0 at  $l_i$  and  
 1358 to 1 at  $r_i$ , and let  $m$  denote the number of these pairs. For the case  $m = 0$ , we let  $r_0 := t_{\text{first}}$ .



After the last pair, ALG may perform a last flip to 0 at a time denoted by  $t_{\text{last}}$ . We denote the set of times at which ALG flips its output by  $F := \{t_{\text{first}}, l_1, r_1, \dots, l_m, r_m, t_{\text{last}}\}$ . We partition the  $i$ th phase into two parts as  $\mathbf{P}_i = \mathbf{L}_i \mathbf{R}_i$ , where  $\mathbf{L}_i := \sigma(r_{i-1}, l_i]$  is the *left* part and  $\mathbf{R}_i := \sigma(l_i, r_i]$  is the *right* part.

A (sub)segment of  $\sigma$  between times  $i$  and  $j > i$  is a contiguous subsequence of  $\sigma$  specified by  $\sigma(i, j]$ . We denote the concatenation of any two consecutive segments  $\mathbf{S}_1$  and  $\mathbf{S}_2$  by  $\mathbf{S}_1 \mathbf{S}_2$ . We say a segment  $\mathbf{S}$  is *short* if  $|\mathbf{S}| < 3\lambda$ , otherwise  $|\mathbf{S}| \geq 3\lambda$  and it is *long*. For  $b \in \{0, 1\}$ , we denote the number of  $b$ -requests in a segment  $\mathbf{S}$  by  $n_b(\mathbf{S})$ .

We compare the cost of our algorithm to the cost of an optimal offline algorithm denoted by OPT. The total cost incurred by ALG and OPT while serving a segment  $\mathbf{S}$  is denoted by  $\text{ALG}(\mathbf{S})$  and  $\text{OPT}(\mathbf{S})$  respectively. We denote the cost of mismatches to ALG for a segment  $\mathbf{S}$  by  $\text{mis}(\mathbf{S})$ . Therefore,  $\text{ALG}(\mathbf{W}_t) = \text{mis}(\mathbf{W}_t) + \alpha \leq 2\lambda + \alpha$  for  $t \in F$ .

**Charging Scheme.** We partition the input  $\sigma$  into *phases*, separated by flips to 1. Precisely  $\sigma = \mathbf{P}_{\text{first}} \mathbf{P}_1 \dots \mathbf{P}_k \mathbf{P}_{\text{last}}$ , where  $\mathbf{P}_{\text{first}} := \sigma(0, t_{\text{first}}]$  is the subsequence until the first flip,  $\mathbf{P}_{\text{last}} := \sigma(r_m, |\sigma|]$  is the subsequence from the last flip to the last request, and each  $\mathbf{P}_i := \sigma(r_{i-1}, r_i]$ ,  $1 \leq i \leq m$ . In a series of lemmas, we analyze the total cost to OPT and eventually the competitive ratio for each part separately. Then, we aggregate all individual ratios into one competitive ratio in Theorem E.12.

For any phase  $\mathbf{P}_i$ , Lemma E.4 lower-bounds the number of 0-requests in  $\mathbf{W}_{l_i}$  and shows the window is contained in  $\mathbf{P}_i$ , given the flip to 0 occurs by Rule 2.

► **Lemma E.4.** *For any phase  $\mathbf{P}_i$ , if the flip to 0 at  $l_i$  occurs by Rule 2 then  $\mathbf{P}_{i-1} \cap \mathbf{W}_{l_i} = \emptyset$  and  $n_0(L_i) > \lambda$ .*

**Proof.** By definition, ALG flips to 1 at  $r_{i-1} < l_i$  in the phase  $\mathbf{P}_{i-1}$ . Since  $T \geq 6\lambda$  (by definition), the segment  $\mathbf{W}_{r_{i-1}}$  is contained in the visible horizon at each time  $r_{i-1}, \dots, r_{i-1} + 3\lambda$ . Thus, ALG outputs 1 at every time step  $r_{i-1}, \dots, r_{i-1} + 3\lambda$ . Therefore, the flip to 0 by Rule 2 occurs earliest at  $r_{i-1} + 3\lambda + 1$ , that is, at  $l_i \geq r_{i-1} + 3\lambda + 1$ . Hence,

$$|\mathbf{L}_i| = l_i - r_{i-1} \geq 3\lambda + 1 = |\mathbf{W}_{l_i}| + 1.$$

That is,  $\mathbf{W}_{l_i}$  is contained in  $\mathbf{L}_i$ , which implies  $\mathbf{P}_{i-1} \cap \mathbf{W}_{l_i} = \emptyset$ . Moreover, we have

$$n_1(\mathbf{W}_{l_i}) \leq 2\lambda - 1,$$

as otherwise ALG would output 1 at  $l_i$ , contradicting our assumption. Thus,

$$n_0(\mathbf{L}_i) \geq n_0(\mathbf{W}_{l_i}) = |\mathbf{W}_{l_i}| - n_1(\mathbf{W}_{l_i}) \leq 3\lambda - (2\lambda - 1) \geq \lambda + 1 > \lambda. \quad \blacktriangleleft$$

Lemma E.4 implies that if the flip to 0 in  $\mathbf{P}_i$  occurs by Rule 2 then both windows  $\mathbf{W}_{l_i}$  and  $\mathbf{W}_{r_i}$  are contained in  $\mathbf{P}_i$ , i.e., they do not overlap  $\mathbf{P}_{i-1}$ .

► **Corollary E.5.** *For any phase  $\mathbf{P}_i$ , if the flip to 0 at  $l_i$  occurs by Rule 2 then both windows  $\mathbf{W}_{l_i}$  and  $\mathbf{W}_{r_i}$  are contained in  $\mathbf{P}_i$ .*

The next lemma bounds the number of 0-requests in  $\mathbf{L}_i$  and the number of 1-requests in  $\mathbf{R}_i$ , given the flip at  $l_i$  occurs by Rule 1.

► **Lemma E.6.** *For any phase  $\mathbf{P}_i$ , if the flip at  $l_i$  occurs by Rule 1 then  $n_0(\mathbf{L}_i) \geq \lambda$  and  $n_1(\mathbf{R}_i) \geq \lambda$ .*

**Proof.** A flip to 1 always occurs by Rule 1 and therefore  $\mathbf{W}_{r_i}$  contains exactly  $2\lambda$  many 1-requests. Assume for contradiction that  $x := n_1(\mathbf{W}_{r_i} \setminus \mathbf{W}_{l_i}) < \lambda$ . Then the remaining  $2\lambda - x$  many 1-requests must be in  $\mathbf{W}_{l_i}$  which implies  $n_1(\mathbf{W}_{l_i}) \geq 2\lambda - x > \lambda$ . Since the flip to 0 is assumed to occur by Rule 1,  $\mathbf{W}_{l_i}$  must contain exactly  $2\lambda$  many 0-requests. However, the number of 0-requests in  $\mathbf{W}_{l_i}$  is at most  $3\lambda - n_1(\mathbf{W}_{l_i}) \leq 3\lambda - (2\lambda - x) < 2\lambda$ , which implies either there is no flip to 0 at  $l_i$ , or the flip to 0 occurs by Rule 2, contradicting our assumption. Therefore,  $n_1(\mathbf{R}_i) \geq n_1(\mathbf{W}_{r_i} \setminus \mathbf{W}_{l_i}) \geq \lambda$ .

The flip to 0 at  $l_i$  follows the flip to 1 in the previous phase  $\mathbf{P}_{i-1}$ , which occurs by Rule 1. Then, following a similar argument as used for the flip at  $r_i$  (involving  $\mathbf{W}_{r_{i-1}}$ ), we conclude  $n_0(\mathbf{L}_i) \geq n_0(\mathbf{W}_{l_i} \setminus \mathbf{P}_{i-1}) \geq \lambda$ .  $\blacktriangleleft$

A segment  $\mathbf{S}$  is a *block* segment if  $|\mathbf{S}| = 3\lambda$ . The following lemma lower-bounds costs to OPT for any block segment in which ALG never flips, or it flips only at the end of the block, i.e., immediately *after* serving the last request in the block.

► **Lemma E.7.** *For any block segment  $\mathbf{S}$  where ALG pays the mismatch cost  $\text{mis}(\mathbf{S})$ , if ALG does not flip in  $\mathbf{S}$  then  $\text{OPT}(\mathbf{S}) \geq \min\{n_0(\mathbf{S}), n_1(\mathbf{S}), \alpha\} \geq \text{mis}(\mathbf{S})/2$ . If ALG flips only at end the of  $\mathbf{S}$  then  $\text{OPT}(\mathbf{S}) \geq \lambda$ .*

**Proof.** Assume ALG outputs  $b \in \{0, 1\}$  in  $\mathbf{S}$ . Then it must hold  $\text{mis}(\mathbf{S}) = n_{\neg b}(\mathbf{S}) \leq 2\lambda - 1$  (otherwise it would flip to  $\neg b$ ) and hence  $n_b(\mathbf{S}) \geq \lambda + 1 \geq n_{\neg b}(\mathbf{S})/2 = \text{mis}(\mathbf{S})/2$ . OPT pays mismatches to  $b$ -requests or to  $\neg b$ -requests, or it performs a flip and possibly serves some of them for free. Thus, regardless of OPT's choices in  $\mathbf{S}$ ,

$$\text{OPT}(\mathbf{S}) \geq \min\{n_b(\mathbf{S}), n_{\neg b}(\mathbf{S}), \alpha\} \geq \min\{\text{mis}(\mathbf{S})/2, \text{mis}(\mathbf{S}), \alpha\}$$

and the claim follows since  $\text{mis}(\mathbf{S}) \leq 2\lambda - 1$  and thus  $\text{mis}(\mathbf{S})/2 < \lambda \leq \alpha$ .

Assume that ALG flips to 0 at the end of  $\mathbf{S}$ . Then  $\mathbf{S}$  contains at most  $2\lambda$  many 0-requests as otherwise the flip to 0 would occur earlier in  $\mathbf{S}$ . Moreover,  $\mathbf{S}$  contains at most  $2\lambda - 1$  many 1-requests, otherwise ALG would output 1 at the end of  $\mathbf{S}$  (by Rule 1). Therefore,  $\mathbf{S}$  contains at least  $\lambda$  many of either 0- and 1-requests, that is,  $n_b(\mathbf{S}), n_{\neg b}(\mathbf{S}) \geq \lambda$ . Regardless of OPT's actions, we have

$$\text{OPT}(\mathbf{S}) \geq \min\{n_b(\mathbf{S}), n_{\neg b}(\mathbf{S}), \alpha\} \geq \min\{\lambda, \alpha\} = \lambda.$$

The case where ALG flips to 1 at the end of the segment is analogous (up to swapping 0's and 1's).  $\blacktriangleleft$

Consider any segment  $\mathbf{U}$  that can be partitioned into a set of blocks  $\mathcal{B}$ . If ALG does not flip in  $\mathbf{U}$  then applying Lemma E.7 to each block separately yields

$$\text{OPT}(\mathbf{U}) = \sum_{\mathbf{B} \in \mathcal{B}} \text{OPT}(\mathbf{B}) \geq \sum_{\mathbf{B} \in \mathcal{B}} \text{mis}(\mathbf{B})/2 = \text{mis}(\mathbf{U})/2.$$

► **Corollary E.8.** *For any segment  $\mathbf{U}$  such that  $|\mathbf{U}|$  is a multiple of  $3\lambda$ , if ALG does not flip in  $\mathbf{U}$  then  $\text{OPT}(\mathbf{U}) \geq \text{mis}(\mathbf{U})/2$ .*

The following lemma lower-bounds costs to OPT for a segment in which OPT does not flip, and ALG flips immediately after serving the segment, i.e., at the *end* of the segment.

► **Lemma E.9.** *Consider any phase  $\mathbf{P}_i$  and a segment  $\mathbf{S} \in \{\mathbf{L}_i, \mathbf{R}_i\}$  s.t.  $|\mathbf{S}| \geq 3\lambda$ . ALG outputs  $b \in \{0, 1\}$  for the entire  $\mathbf{S}$  and flips to  $\neg b$  at  $t \in \{l_i, r_i\}$ . Consider the partitioning  $\mathbf{S} = \mathbf{U}\mathbf{V}\mathbf{W}_t$ , where  $|\mathbf{U}|$  is a multiple of  $3\lambda$ ,  $|\mathbf{V}| < 3\lambda$ . If OPT does not flip in  $\mathbf{V}\mathbf{W}_t$  then one of the two cases holds:*

- 1439 i) OPT serves  $\mathbf{VW}_t$  in state  $b$  and  $\text{OPT}(\mathbf{VW}_t) = \text{mis}(\mathbf{VW}_t)$   
 1440 ii) OPT serves  $\mathbf{VW}_t$  in state  $\neg b$  and  $\text{OPT}(\mathbf{VW}_t) \geq \lambda$ .

1441 **Proof.** We show the claim for the case  $b = 1$ . The argument for  $b = 1$  is analogous subject  
 1442 to swapping 0's and 1's. If OPT serves the entire  $\mathbf{S}$  in state  $b = 1$  then both OPT and ALG  
 1443 pay mismatches to  $\neg b$ 's in the entire  $\mathbf{S}$  and  $\text{OPT}(\mathbf{S}) = \text{mis}(\mathbf{S})$  which concludes Lemma E.9.i.  
 1444 Otherwise, OPT serves the entire  $\mathbf{S}$  in state  $\neg b = 0$  and possibly pays no cost for  $\mathbf{V}$ .  
 1445 Therefore,  $\text{OPT}(\mathbf{S}) \geq \text{OPT}(\mathbf{W}_t) \geq \lambda$ , where the last inequality follows from Lemma E.7,  
 1446 concluding Lemma E.9.ii.  $\blacktriangleleft$

1447 The following lemma lower-bounds OPT's cost for a segment in which ALG does not flip  
 1448 until the end of the segment.

1449 **► Lemma E.10.** Consider any phase  $P_i$  and a segment  $\mathbf{S} \in \{L_i, R_i\}$  s.t.  $|\mathbf{S}| \geq 3\lambda$ . ALG flips  
 1450 to  $b \in \{0, 1\}$  at  $t \in \{l_i, r_i\}$  after it outputs  $\neg b$  for the entire  $\mathbf{S}$ . Consider the partitioning  
 1451  $\mathbf{S} = \mathbf{UVW}_t$ , where  $|\mathbf{U}|$  is a multiple of  $3\lambda$ ,  $|\mathbf{V}| < 3\lambda$ . If OPT flips in  $\mathbf{VW}_t$  then one of the two  
 1452 cases applies:

- 1453 i) OPT flips to  $b$  in  $\mathbf{VW}_t$  and  $\text{OPT}(\mathbf{VW}_t) \geq \min\{\text{mis}(\mathbf{V}), \lambda\} + \alpha$ ,  
 1454 ii) OPT flips to  $\neg b$  in  $\mathbf{VW}_t$  and  $\text{OPT}(\mathbf{VW}_t) \geq \max\{\text{mis}(\mathbf{VW}_t)/2 - \lambda, 0\} + \alpha$ .

1455 **Proof.** We analyze the case where ALG flips to  $b = 0$  at  $t$ . The case for  $b = 1$  follows in a  
 1456 similar way subject to swapping 0's and 1's.

1457 If OPT flips more than once in  $\mathbf{VW}_t$  then  $\text{OPT}(\mathbf{VW}_t) \geq 2\alpha$ . Since  $\min\{\text{mis}(\mathbf{V}), \lambda\} \leq \lambda \leq \alpha$ ,  
 1458 Lemma E.10.i follows from  $\text{OPT}(\mathbf{VW}_t) \geq 2\alpha \geq \min\{\text{mis}(\mathbf{V}), \lambda\} + \alpha$ . Since  $\text{mis}(\mathbf{VW}_t) < 4\lambda$ ,  
 1459 we have  $\text{mis}(\mathbf{VW}_t)/2 - \lambda < \lambda \leq \alpha$ . Thus, if OPT flips more than once in  $\mathbf{VW}_t$  then  
 1460  $\text{OPT}(\mathbf{VW}_t) \geq 2\alpha \geq \max\{\text{mis}(\mathbf{VW}_t)/2 - \lambda, 0\} + \alpha$  and Lemma E.10.ii holds. Hence, in the  
 1461 remainder, we assume OPT flips only once in  $\mathbf{VW}_t$ .

1462 If OPT flips in  $\mathbf{V}$  then it does not flip in  $\mathbf{W}_t$  and by Lemma E.7,  $\text{OPT}(\mathbf{W}_t) \geq \lambda$ . Therefore,  
 1463  $\text{OPT}(\mathbf{VW}_t) = \text{OPT}(\mathbf{V}) + \text{OPT}(\mathbf{W}_t) \geq \alpha + \lambda$ . Next, we provide lower bounds for the cost of  
 1464 OPT in  $\mathbf{VW}_t$  by distinguishing the two cases of OPT's flip, to 0 and to 1.

1465 **OPT flips to  $b = 0$  in  $\mathbf{VW}_t$ .** If OPT flips in  $\mathbf{V}$  then

$$1466 \quad \text{OPT}(\mathbf{VW}_t) \geq \lambda + \alpha \geq \min\{\text{mis}(\mathbf{V}), \lambda\} + \alpha.$$

1467 Otherwise, OPT serves  $\mathbf{V}$  in state 1,  $\text{OPT}(\mathbf{V}) = n_0(\mathbf{V}) = \text{mis}(\mathbf{V})$ , and it flips to 0 in  $\mathbf{W}_t$ .  
 1468 Therefore,

$$1469 \quad \text{OPT}(\mathbf{VW}_t) = \text{OPT}(\mathbf{V}) + \text{OPT}(\mathbf{W}_t) \geq \text{mis}(\mathbf{V}) + \alpha \geq \min\{\text{mis}(\mathbf{V}), \lambda\} + \alpha,$$

1470 which concludes Lemma E.10.i.

1471 **OPT flips to  $\neg b = 1$  in  $\mathbf{VW}_t$ .** If OPT flips in  $\mathbf{V}$  then

$$1472 \quad \text{OPT}(\mathbf{VW}_t) \geq \lambda + \alpha \geq \max\{\text{mis}(\mathbf{VW}_t)/2 - \lambda, 0\} + \alpha.$$

1473 Otherwise, OPT serves the entire  $\mathbf{V}$  in state 0. We lower-bound the cost of mismatches  
 1474 in  $\mathbf{W}_t$  after OPT flips to 1, using the following observation. If  $n_0(\mathbf{V}) \geq 1$  then there are  
 1475 at least  $\lambda + 1$  many 1-requests between the last 0-request in  $\mathbf{V}$  and ALG's flip to 0 at  $t$ .  
 1476 Assume this is not the case and  $n_1(\mathbf{W}_t) \leq \lambda$ . Therefore  $n_0(\mathbf{W}_t) = 3\lambda - n_1(\mathbf{W}_t) \geq 2\lambda$   
 1477 and  $n_0(\mathbf{V}) + n_0(\mathbf{W}_t) \geq 2\lambda + 1$ , that is, ALG flips to 0 earlier than  $t$  in  $\mathbf{W}_t$ , contradicting  
 1478 our assumption. Therefore, at least  $\lambda + 1$  many 1-requests in  $\mathbf{VW}_t$  occur after the last  
 1479 0-request in  $\mathbf{V}$ . Let  $\sigma_p = 1$  be the  $(\lambda + 1)$ th 1-request in  $\mathbf{VW}_t$ .

Next, we lower-bound the number of mismatches incurred by OPT after it flips to 1. Either OPT flips to 1 after  $\sigma_p$ , that is, after paying at least  $\lambda + 1$  mismatches to 1-requests in  $\mathbf{VW}_t$ , or it flips to 1 before serving  $\sigma_p$ . In the latter case, OPT pays mismatches to the remaining 0-requests (occurring after  $\sigma_p$ ) in  $\mathbf{W}_t$ . Let  $x$  be the number of 0-requests in  $\mathbf{VW}_t$  after  $\sigma_p$ . Then the number of 0-requests in  $\mathbf{VW}_t$  and before  $\sigma_p$  is  $n_0(\mathbf{VW}_t) - x < 2\lambda$ , otherwise the flip to 0 would occur earlier than  $p < t$ , contradicting our assumption.

Summing up all the considered cases, OPT pays at least

$$x > n_0(\mathbf{VW}_t) - 2\lambda = \text{mis}(\mathbf{VW}_t) - 2\lambda$$

mismatches to 0-requests after it flips to 1, and

$$\text{OPT}(\mathbf{VW}_t) \geq \max\{\text{mis}(\mathbf{VW}_t) - 2\lambda, 0\} + \alpha \geq \max\{\text{mis}(\mathbf{VW}_t)/2 - \lambda, 0\} + \alpha,$$

which concludes Lemma E.10.ii.  $\blacktriangleleft$

► **Lemma E.11.** *For any phase  $\mathbf{P}_i$ , we have  $\text{ALG}(\mathbf{P}_i)/\text{OPT}(\mathbf{P}_i) \leq 4 + \frac{2\alpha}{\lambda}$ .*

**Proof.** Recall that in the phase  $\mathbf{P}_i$ , ALG flips to 0 at  $l := l_i$  and later to 1 at  $r := r_i$ . We bound the costs to OPT and ALG in  $\mathbf{L} := \mathbf{L}_i$  and  $\mathbf{R} := \mathbf{R}_i$  separately, which is followed by the competitive ratio for  $\mathbf{P}_i = \mathbf{LR}$ . Thus,  $\text{ALG}(\mathbf{P}_i) = \text{ALG}(\mathbf{L}) + \text{ALG}(\mathbf{R})$  and  $\text{OPT}(\mathbf{P}_i) = \text{OPT}(\mathbf{L}) + \text{OPT}(\mathbf{R})$ . If  $L$  is long, that is  $|\mathbf{L}| \geq 3\lambda$ , we consider the partitioning  $\mathbf{L} = \mathbf{U}_l \mathbf{V}_l \mathbf{W}_l$  where  $|\mathbf{U}_l|$  is a multiple of  $3\lambda$  and  $|\mathbf{V}_l| < 3\lambda$ . Similarly,  $\mathbf{R} = \mathbf{U}_r \mathbf{V}_r \mathbf{W}_r$  where  $|\mathbf{U}_r|$  is a multiple of  $3\lambda$  and  $|\mathbf{V}_r| < 3\lambda$ .

To obtain our upper bounds, we use the fact  $\sum_j a_j / \sum_j b_j \leq \max_j a_j / b_j$  for  $a_j, b_j \geq 0$ . We bound the costs under four major cases of  $|\mathbf{L}|$  and  $|\mathbf{R}|$ .

**Both parts are short.** Since  $|\mathbf{L}| < 3\lambda = |\mathbf{W}_l|$ ,  $\mathbf{L}$  is a subsegment of  $\mathbf{W}_l$  which implies  $\mathbf{P}_{i-1} \cap \mathbf{W}_l \neq \emptyset$ . The latter, together with Lemma E.4 implies that the flip to 0 at  $l$  occurs by Rule 1. Then, Lemma E.6 guarantees at least  $\lambda$  many 0-requests in  $\mathbf{L}$  and at least  $\lambda$  many 1-requests in  $\mathbf{R}$  are contained. Therefore, regardless of whether OPT performs a flip in  $\mathbf{P}_i$  or not, it pays  $\text{OPT}(\mathbf{P}_i) \geq \min\{\lambda, \alpha\} = \lambda$ . Using  $\text{mis}(\mathbf{W}_l), \text{mis}(\mathbf{W}_r) \leq 2\lambda + \alpha$ , we obtain

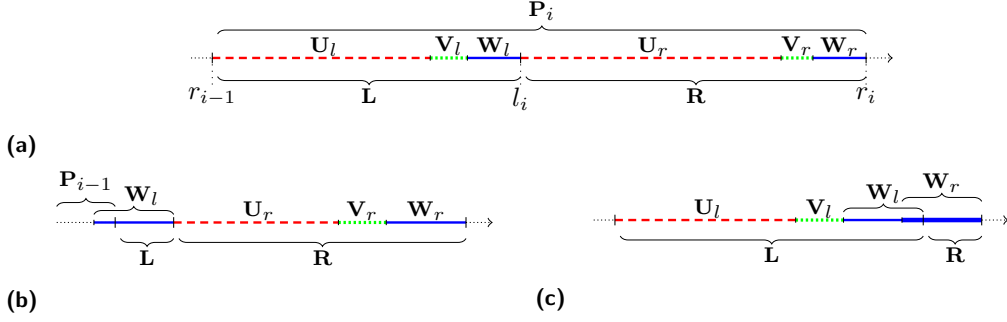
$$\frac{\text{ALG}(\mathbf{P}_i)}{\text{OPT}(\mathbf{P}_i)} = \frac{\text{ALG}(\mathbf{L}) + \text{ALG}(\mathbf{R})}{\text{OPT}(\mathbf{P}_i)} \leq \frac{\text{mis}(\mathbf{W}_l) + \text{mis}(\mathbf{W}_r) + 2\alpha}{\text{OPT}(\mathbf{P}_i)} \leq \frac{4\lambda + 2\alpha}{\lambda}. \quad (1)$$

**Both parts are long.** See Figure 3a for an illustration. Regardless of OPT's actions in  $\mathbf{L}$ , we have  $\text{OPT}(\mathbf{U}_l) \geq \text{mis}(\mathbf{U}_l)/2$  from Corollary E.8, and  $\text{OPT}(\mathbf{W}_l) \geq \lambda$  from Lemma E.7. Therefore,  $\text{OPT}(\mathbf{L}) = \text{OPT}(\mathbf{U}_l) + \text{OPT}(\mathbf{V}_l \mathbf{W}_l) \geq \text{mis}(\mathbf{U}_l)/2 + \lambda$ . Similarly for  $\mathbf{R}$ , we have  $\text{OPT}(\mathbf{R}) \geq \text{mis}(\mathbf{U}_r)/2 + \lambda$ . Thus,

$$\text{OPT}(\mathbf{P}_i) = \text{OPT}(\mathbf{L}) + \text{OPT}(\mathbf{R}) \geq \text{mis}(\mathbf{U}_l)/2 + \text{mis}(\mathbf{U}_r)/2 + 2\lambda.$$

For ALG's mismatch cost, we have  $\text{mis}(\mathbf{V}_l \mathbf{W}_l) = \text{mis}(\mathbf{V}_l) + \text{mis}(\mathbf{W}_l) \leq 2\lambda - 1 + 2\lambda < 4\lambda$  and similarly  $\text{mis}(\mathbf{V}_r \mathbf{W}_r) < 4\lambda$ . Then,  $\text{ALG}(\mathbf{P}_i) = \text{ALG}(\mathbf{L}) + \text{ALG}(\mathbf{R}) = \text{mis}(\mathbf{U}_l) + \text{mis}(\mathbf{V}_l \mathbf{W}_l) + \alpha + \text{mis}(\mathbf{U}_r) + \text{mis}(\mathbf{V}_r \mathbf{W}_r) + \alpha \leq \text{mis}(\mathbf{U}_l) + \text{mis}(\mathbf{U}_r) + 8\lambda + 2\alpha$ , and

$$\frac{\text{ALG}(\mathbf{P}_i)}{\text{OPT}(\mathbf{P}_i)} \leq \frac{\text{mis}(\mathbf{U}_l) + \text{mis}(\mathbf{U}_r) + 8\lambda + 2\alpha}{\text{mis}(\mathbf{U}_l)/2 + \text{mis}(\mathbf{U}_r)/2 + 2\lambda} \leq \frac{8\lambda + 2\alpha}{2\lambda} = 4 + \frac{\alpha}{\lambda}. \quad (2)$$



**Figure 3** ALG flips to 0 at the end of the segment  $L$  and flips to 1 at the end of the segment  $R$ . In 3a, both segments  $L$  and  $R$  are long. As a result, both windows  $W_l$  and  $W_r$  are contained in the phase. In 3b and 3c, one part is short and the other is long. The window  $W_l$  overlaps the phase  $P_{i-1}$  in 3b, and it overlaps  $W_r$  (in thick blue) in 3c.

1517 **Only the right part is long.** Then,  $L$  is a subsegment of  $W_l$  (see Figure 3b) and therefore  
 1518  $\text{ALG}(L) = \text{mis}(W_l) + \alpha \leq 2\lambda + \alpha$  For  $R$ , we distinguish several cases.

1519 **Case 1.1.** OPT enter  $V_r W_r$  in state 0 and serves it in state 0. In this case, Lemma E.9.i applies  
 1520 which together with Corollary E.8 yields  $\text{OPT}(R) = \text{OPT}(U_r) + \text{mis}(V_r) + \text{mis}(W_r) \geq$   
 1521  $\text{mis}(U_r)/2 + \text{mis}(V_r) + \lambda$ . Thus, we have,  $\text{ALG}(P_i) = \text{ALG}(L) + \text{ALG}(R) = (2\lambda + \alpha) +$   
 1522  $\text{mis}(U_r) + \text{mis}(V_r) + \text{mis}(W_r) + \alpha \leq \text{mis}(U_r) + \text{mis}(V_r) + 4\lambda + 2\alpha$  and thereby

$$\frac{\text{ALG}(P_i)}{\text{OPT}(P_i)} \leq \frac{\text{mis}(U_r) + \text{mis}(V_r) + 4\lambda + 2\alpha}{\text{mis}(U_r)/2 + \text{mis}(V_r) + \lambda} \leq \frac{4\lambda + 2\alpha}{\lambda} = 4 + \frac{2\alpha}{\lambda}. \quad (3)$$

1525 **Case 1.2.** OPT enters  $V_r W_r$  in state 0 and flips to 1 in  $V_r W_r$ . In this case, Lemma E.10.i  
 1526 applies which together with Corollary E.8 yields  $\text{OPT}(R) = \text{OPT}(U_r) + \text{OPT}(V_r W_r) \geq$   
 1527  $\text{mis}(U_r)/2 + \min\{\text{mis}(V_r), \lambda\} + \alpha$ . By distinguishing the two cases  $\text{mis}(V_r) < \lambda$  and  
 1528  $\text{mis}(V_r) \geq \lambda$ , and using  $\text{ALG}(P_i) = \text{ALG}(L) + \text{ALG}(R) \leq (2\lambda + \alpha) + \text{mis}(U_r) + \text{mis}(V_r) +$   
 1529  $2\lambda + \alpha$ , we obtain

$$\begin{aligned} \frac{\text{ALG}(P_i)}{\text{OPT}(P_i)} &\leq \frac{\text{mis}(U_r) + \text{mis}(V_r) + 4\lambda + 2\alpha}{\text{mis}(U_r)/2 + \min\{\text{mis}(V_r), \lambda\} + \alpha} \\ &\leq \max\left\{\frac{4\lambda + 2\alpha}{\lambda}, \frac{6\lambda + 2\alpha}{\lambda + \alpha}\right\} = 4 + \frac{2\alpha}{\lambda}. \end{aligned} \quad (4)$$

1531 **Case 1.3.** OPT enters  $V_r W_r$  in state 1. If OPT serves it in state 1 then from Lemma E.7, we  
 1532 have  $\text{OPT}(V_r W_r) \geq \text{OPT}(W_r) \geq \lambda$ . Else, it flips to 0 and  $\text{OPT}(V_r W_r) \geq \alpha \geq \lambda$ . Therefore,  
 1533 in either case,  $\text{OPT}(V_r W_r) \geq \lambda$ . Since  $P_{i-1} \cap W_l \neq \emptyset$ , Lemma E.4 implies that the flip  
 1534 to 0 at  $l$  occurs by Rule 1. Then, from Lemma E.6, we have  $n_0(L) = n_0(W_l \setminus P_{i-1}) \geq \lambda$ .

1535 Next, we lower-bound OPT's cost in  $P_i$  before entering  $V_r W_r$  (i.e. in  $LU_r$ ). Either OPT  
 1536 serves the entire  $L$  in state 1 and  $\text{OPT}(L) = n_0(L) \geq \lambda$ , or it flips in  $L$  at cost  $\alpha \geq \lambda$ . From  
 1537 Lemma E.6, we have  $\text{OPT}(U_r) \geq \text{mis}(U_r)/2$ , which yields

$$\text{OPT}(P_i) \geq \text{OPT}(L) + \text{OPT}(U_r) + \text{OPT}(V_r W_r) \geq \lambda + \text{mis}(U_r)/2 + \lambda.$$

1539 Using

$$\text{ALG}(P_i) = \text{ALG}(L) + \text{ALG}(R) \leq (2\lambda + \alpha) + \text{mis}(U_r) + \text{mis}(V_r W_r) + \alpha,$$

1541 we obtain

$$\frac{\text{ALG}(P_i)}{\text{OPT}(P_i)} \leq \frac{\text{mis}(U_r) + 6\lambda + 2\alpha}{\text{mis}(U_r)/2 + 2\lambda} \leq \frac{6\lambda + 2\alpha}{2\lambda} = 3 + \frac{\alpha}{\lambda}. \quad (5)$$

1544 **Only the left part is long.** See Figure 3c for an illustration. We distinguish cases of OPT's  
 1545 state when it enters  $\mathbf{V}_l\mathbf{W}_l$ . Note that ALG's flip to 0 at  $l$  possibly occurs by Rule 2.

1546 **Case 2.1.** OPT enters  $\mathbf{V}_l\mathbf{W}_l$  in state 1 and serves it in state 1. This case is symmetric to  
 1547 Case 1.1 and the upper bound (3) holds analogously, after swapping the usage of  $\mathbf{R}$  and  
 1548  $\mathbf{L}$ , as well as 0's and 1's.

1549 **Case 2.2.** OPT enters  $\mathbf{V}_r\mathbf{W}_r$  in state 1 and flips to 0 later in this segment. This case is  
 1550 symmetric to Case 1.2 and the upper bound (4) holds analogously, after swapping the  
 1551 usage of  $\mathbf{R}$  and  $\mathbf{L}$ , as well as 0's and 1's.

1552 **Case 2.3.** OPT enters  $\mathbf{V}_l\mathbf{W}_l$  in state 0. Recall that if the flip to 0 is by Rule 2 then Lemma E.6  
 1553 does not apply and possibly  $n_1(\mathbf{R}) = n_1(\mathbf{W}_r \setminus \mathbf{W}_l) < \lambda$ . However, since the flip to 1 at  
 1554  $r$  is by Rule 1, we have  $n_1(\mathbf{W}_r) = 2\lambda$ , and since  $\mathbf{W}_r$  is a subsegment of  $\mathbf{W}_l\mathbf{R}$ , we have  
 1555  $n_1(\mathbf{W}_l\mathbf{R}) \geq n_1(\mathbf{W}_r) \geq 2\lambda$ . Since  $\text{mis}(\mathbf{V}_l\mathbf{W}_l) < 4\lambda$ , we have  $\text{mis}(\mathbf{V}_l\mathbf{W}_l)/2 - \lambda < \lambda$  and  
 1556 hence

$$1557 \quad \max\{\text{mis}(\mathbf{V}_l\mathbf{W}_l)/2 - \lambda, 0\} + \lambda < 2\lambda \leq n_1(\mathbf{W}_l\mathbf{R}).$$

1558 Either OPT serves the entire segment  $\mathbf{V}_l\mathbf{W}_l\mathbf{R}$  in state 0 and

$$1559 \quad \text{OPT}(\mathbf{V}_l\mathbf{W}_l\mathbf{R}) \geq n_1(\mathbf{W}_l\mathbf{R}) \geq 2\lambda \geq \max\{\text{mis}(\mathbf{V}_l\mathbf{W}_l)/2 - \lambda, 0\} + \lambda,$$

1560 or it flips to 1 in this segment. If OPT flips to 1 in  $\mathbf{R}$  then by Lemma E.6,

$$1561 \quad \text{OPT}(\mathbf{V}_l\mathbf{W}_l\mathbf{R}) \geq \text{OPT}(\mathbf{W}_l) + \text{OPT}(\mathbf{R}) \geq \lambda + \alpha \geq 2\lambda \geq \max\{\text{mis}(\mathbf{V}_l\mathbf{W}_l)/2 - \lambda, 0\}.$$

1562 Else, it flips in  $\mathbf{V}_l\mathbf{W}_l$ ; by Lemma E.10.ii,

$$1563 \quad \text{OPT}(\mathbf{V}_l\mathbf{W}_l) \geq \max\{\text{mis}(\mathbf{V}_l\mathbf{W}_l)/2 - \lambda, 0\} + \lambda.$$

1564 Thus, in any case where OPT enters  $\mathbf{V}_l\mathbf{W}_l$  in state 0, we have

$$1565 \quad \text{OPT}(\mathbf{V}_l\mathbf{W}_l) \geq \max\{\text{mis}(\mathbf{V}_l\mathbf{W}_l)/2 - \lambda, 0\} + \lambda.$$

1566 By applying Corollary E.8 to  $\mathbf{U}_l$ , we obtain

$$1567 \quad \text{OPT}(\mathbf{P}_i) \geq \text{OPT}(\mathbf{U}_l) + \text{OPT}(\mathbf{V}_l\mathbf{W}_l\mathbf{R}) \geq \text{mis}(\mathbf{U}_l)/2 + \max\{\text{mis}(\mathbf{V}_l\mathbf{W}_l)/2 - \lambda, 0\} + \lambda. \quad (6)$$

1568 If  $\text{mis}(\mathbf{V}_l\mathbf{W}_l) < 2\lambda$  then (6) reduces to  $\text{OPT}(\mathbf{P}_i) \geq \text{mis}(\mathbf{U}_r)/2 + \lambda$ . Using

$$1569 \quad \text{ALG}(\mathbf{P}_i) \leq \text{mis}(\mathbf{U}_l) + \text{mis}(\mathbf{V}_l\mathbf{W}_l) + \alpha + (2\lambda + \alpha) \leq \text{mis}(\mathbf{U}_l) + 4\lambda + 2\alpha,$$

1570 we obtain

$$1571 \quad \frac{\text{ALG}(\mathbf{P}_i)}{\text{OPT}(\mathbf{P}_i)} \leq \frac{\text{mis}(\mathbf{U}_r) + 4\lambda + 2\alpha}{\text{mis}(\mathbf{U}_r)/2 + \lambda} \leq \frac{4\lambda + 2\alpha}{\lambda} = 4 + \frac{2\alpha}{\lambda}. \quad (7)$$

1573 Else,  $\text{mis}(\mathbf{V}_l\mathbf{W}_l) \geq 2\lambda$  holds and (6) reduces to

$$1574 \quad \text{OPT}(\mathbf{V}_l\mathbf{W}_l) \geq (\text{mis}(\mathbf{V}_l\mathbf{W}_l)/2 - \lambda) + \lambda.$$

1575 Let  $z := \text{mis}(\mathbf{V}_l\mathbf{W}_l)/2 - \lambda$ . Then,  $\text{mis}(\mathbf{V}_l\mathbf{W}_l) = 2z + 2\lambda$ ,

$$1576 \quad \text{ALG}(\mathbf{P}_i) \leq \text{mis}(\mathbf{U}_l) + \text{mis}(\mathbf{V}_l\mathbf{W}_l) + 2\lambda + 2\alpha = \text{mis}(\mathbf{U}_l) + (2z + 2\lambda) + 2\lambda + 2\alpha,$$

1577 and

$$1578 \quad \frac{\text{ALG}(\mathbf{P}_i)}{\text{OPT}(\mathbf{P}_i)} \leq \frac{\text{mis}(\mathbf{U}_r) + 2z + 4\lambda + 2\alpha}{\text{mis}(\mathbf{U}_r)/2 + z + \lambda} \leq \frac{4\lambda + 2\alpha}{\lambda} = 4 + \frac{2\alpha}{\lambda}. \quad (8)$$

1579

1580 From all upper bounds (1)–(8), we conclude  $\text{ALG}(\mathbf{P}_i)/\text{OPT}(\mathbf{P}_i) \leq 4 + \frac{2\alpha}{\lambda}$ . ◀

► **Theorem E.12.** *For any input sequence  $\sigma$ , any  $T \geq 6$  and  $1 \leq \lambda \leq \alpha$ , we have*

$$\text{ALG}(\sigma) \leq \left(4 + \frac{2\alpha}{\lambda}\right) \text{OPT}(\sigma) + 6\alpha.$$

1581 **Proof.** Assume ALG performs at least one flip in  $\sigma$ . Recall that in this case the input sequence  
1582  $\sigma$  is partitioned as  $\sigma = \mathbf{P}_{\text{first}} \mathbf{P}_1 \dots \mathbf{P}_m \mathbf{P}_{\text{last}}$ , where  $\mathbf{P}_{\text{first}}$  is the subsequence until the first flip  
1583 to 1,  $\mathbf{P}_{\text{last}}$  is the subsequence between the last flip to 1 and the end of the sequence, and  
1584 each  $\mathbf{P}_i$  is the subsequence between two consecutive flips to 1. From Lemma E.11, we have  
1585  $\text{ALG}(\mathbf{P}_i) \leq (4 + 2\alpha/\lambda) \text{OPT}(\mathbf{P}_i)$ . In the remainder, we upper bound the ratio separately for  
1586  $\mathbf{P}_{\text{first}}$  and  $\mathbf{P}_{\text{last}}$ , as well as the case where ALG never flips.

1587 We begin with the first part of the input,  $\mathbf{P}_{\text{first}} = \sigma(0, t_{\text{first}}]$ . Recall that ALG starts  
1588 serving  $\sigma$  by outputting 0 until it flips to 1 at  $t_{\text{first}}$  for the first time. We distinguish two  
1589 cases for  $\mathbf{P}_{\text{first}}$ .

1590  **$\mathbf{P}_{\text{first}}$  is short.** In this case,  $\text{ALG}(\mathbf{P}_{\text{first}}) \leq 2\lambda + \alpha$ . OPT begins in state 0 and either pays  
1591  $\text{mis}(\mathbf{W}_{\text{first}}) = 2\lambda$  mismatches to 1-requests in  $\mathbf{W}_{t_{\text{first}}}$  or it performs a flip to 1. In any case of  
1592 OPT's actions in  $\mathbf{P}_{\text{first}}$ , by distinguishing the two cases  $\alpha < 2\lambda$  and  $\alpha \geq 2\lambda$ , we obtain

$$\frac{\text{ALG}(\mathbf{P}_{\text{first}})}{\text{OPT}(\mathbf{P}_{\text{first}})} \leq \frac{\text{mis}(\mathbf{W}_{t_{\text{first}}}) + \alpha}{\min\{2\lambda, \alpha\}} \leq \frac{2\lambda + \alpha}{\min\{2\lambda, \alpha\}} \leq \max\left\{4, \frac{\alpha}{\lambda}\right\}. \quad (9)$$

1595  **$\mathbf{P}_{\text{first}}$  is long.** Consider the partitioning  $\mathbf{P}_{\text{first}} = \mathbf{U}\mathbf{V}\mathbf{W}$ , where  $|\mathbf{U}|$  is a multiple of  $3\lambda$ ,  $|\mathbf{V}| <$   
1596  $3\lambda$  and  $\mathbf{W} = \mathbf{W}_{t_{\text{first}}}$ . If OPT serves the entire  $\mathbf{V}\mathbf{W}$  in one state (either 0 or 1) then  $\text{OPT}(\mathbf{V}\mathbf{W}) \geq$   
1597  $\text{OPT}(\mathbf{W}) \geq \lambda$  (from Lemma E.7). Otherwise, OPT flips in  $\mathbf{V}\mathbf{W}$  and  $\text{OPT}(\mathbf{V}\mathbf{W}) \geq \alpha \geq \lambda$ . After  
1598 applying Lemma E.8 to  $\mathbf{U}$  and using  $\text{ALG}(\mathbf{P}_{\text{first}}) = \text{mis}(\mathbf{U}) + \text{mis}(\mathbf{V}\mathbf{W}) + \alpha \leq \text{mis}(\mathbf{U}) + 4\lambda + \alpha$ ,  
1599 we obtain

$$\frac{\text{ALG}(\mathbf{P}_{\text{first}})}{\text{OPT}(\mathbf{P}_{\text{first}})} \leq \frac{\text{mis}(\mathbf{U}) + \text{mis}(\mathbf{V}) + 4\lambda + \alpha}{\text{mis}(\mathbf{U})/2 + \lambda} \leq \frac{4\lambda + \alpha}{\lambda} = 4 + \frac{\alpha}{\lambda}. \quad (10)$$

1602 Lastly, we bound costs for  $\mathbf{P}_{\text{last}} = \sigma(r_m, n]$  as follows. If  $|\mathbf{P}_{\text{last}}| < 3\lambda$  then ALG pays up to  
1603  $3\lambda$  mismatches and possibly performs a last flip (to 0) at  $t_m \leq |\sigma|$ . Using  $\lambda \leq \alpha$ , we obtain  
1604  $\text{ALG}(\mathbf{P}_{\text{last}}) \leq |\mathbf{P}_{\text{last}}| + \alpha \leq 3\lambda + \alpha \leq 4\alpha$ ,

1605 Else,  $|\mathbf{P}_{\text{last}}| \geq 3\lambda$ . Consider the partitioning  $\mathbf{P}_{\text{last}} = \mathbf{U}'\mathbf{V}'$  where  $|\mathbf{U}'|$  is a multiple of  $3\lambda$  and  
1606  $|\mathbf{V}'| < 3\lambda$ . ALG possibly flips to 0 one last time at  $t_{\text{last}}$ , in a segment  $\mathbf{S}$  that is either the segment  
1607  $\mathbf{V}'$  or a block of  $\mathbf{U}'$ . In either case,  $\text{mis}(\mathbf{S}) \leq |\mathbf{S}| \leq 3\lambda$  and  $\text{ALG}(\mathbf{S}) = \text{mis}(\mathbf{S}) + \alpha \leq 3\lambda + \alpha \leq 4\alpha$ .  
1608 Using  $\text{mis}(\mathbf{V}') < 2\lambda \leq 2\alpha$  and Corollary E.8, we obtain

$$\text{ALG}(\mathbf{P}_{\text{last}}) = \text{ALG}(\mathbf{U}' \setminus \mathbf{S}) + \text{ALG}(\mathbf{S}) + \text{mis}(\mathbf{V}') \leq \text{ALG}(\mathbf{U}' \setminus \mathbf{S}) + 6\alpha. \quad (11)$$

1611 By an argument similar to that of  $\mathbf{P}_{\text{last}}$ , the bound (11) holds also for the case ALG never  
1612 flips in  $\sigma$ , as ALG does not incur any flipping cost.

1613 **Combining our bounds.** From the upper bounds (9), (10), (11), and by applying Lemma E.11  
1614 to each phase  $\mathbf{P}_i$ , we conclude  $\text{ALG}(\sigma) \leq (4 + \frac{2\alpha}{\lambda}) \text{OPT}(\sigma) + 6\alpha$ , where the additive is a conse-  
1615 quence of (11). ◀

1616 Since  $\lambda = \min\{T/6, \alpha\}$ , we obtain the following ratios for small and large values of  $T$   
1617 separately.

1618 ► **Corollary E.13.** *The time-local algorithm ALG is  $c$ -competitive, where  $c = 6$  for  $T \geq 6\alpha$ ,  
1619  $c = 4 + \frac{12\alpha}{T}$  for  $6 \leq T \leq 6\alpha$ , and  $c = 4 + 2\alpha$  for  $1 \leq T \leq 6$ .*



## 1620 E.4 Discussion on the Answer Set Choice

1621 Alternatively to using request-answer games, some online problems are often more convenient  
 1622 to formulate as *task systems* [14]. A task system consists of a set of configurations, transition  
 1623 costs between configurations and the processing cost of each type of request in every possible  
 1624 configuration. In this case, the output is a sequence of configurations chosen by the algorithm,  
 1625 and the cost of a solution is the sum of all transition and processing costs incurred. An  
 1626 online problem given by a task system does not in general have a unique encoding as a  
 1627 request-answer game, and vice versa. In this work we operate in the request-answer game  
 1628 framework, and unless otherwise specified, we typically assume that algorithms output the  
 1629 current configuration, that is, the set  $Y$  of output values coincides with (some encoding of)  
 1630 the configurations.

1631 The lower bound discussed in Section E.1 assumes a particular encoding of the online file  
 1632 migration as a request-answer game. Namely, the answer set is coincident with the set of  
 1633 nodes in the network, and is equivalent to the algorithm's configuration (placement of the  
 1634 file).

1635 Although this encoding is natural, it causes certain types of problems — when faced  
 1636 with a visible horizon with requests scattered over various nodes, without a clear majority,  
 1637 it still must uniquely determine the file location. Dealing with such situations involves e.g.  
 1638 distinguishing a default configuration in case there's no clear majority in the visible horizon,  
 1639 to avoid excessive file migration.

1640 If we consider a different set of answers, the competitive ratio of the problem may be  
 1641 improved. This is in contrast to the classic algorithms setting, where the competitive ratio is  
 1642 indifferent to the problem encoding.

1643 Consider an additional answer “do not move the file”, denoted SKIP, that instructs the  
 1644 file to stay in its current location. Note that this answer is configuration-dependent, and  
 1645 we may need to track an arbitrarily long sequence of answers to determine the actual file  
 1646 location. For this reason, it is impossible to encode the online problem as a local problem  
 1647 (cf. Section 3.1).

1648 With this answer set, the lower bound from Section E.1 no longer holds. In the remainder  
 1649 of this section, we adapt a classic randomized algorithm by Westbrook [44] to the time-local  
 1650 setting, and we show that it is 3-competitive against the adaptive online adversary even with  
 1651  $T = 1$ , i.e., the access to the last request is sufficient.

1652 Let the Behavioral Coin Flip algorithm be defined as follows. Upon receiving a request  
 1653 from any node, we move the file to this node with probability  $1/(2\alpha)$ , and with probability  
 1654  $1 - 1/(2\alpha)$  we keep the file in its previous location.

1655 ► **Theorem E.14.** *The Behavioral Coin Flip algorithm with  $T = 1$  is 3-competitive against*  
 1656 *the adaptive online adversary for online file migration encoded with the answer set including*  
 1657 *SKIP. Furthermore, no algorithm can obtain a competitive ratio below 3 against the adaptive*  
 1658 *online adversary.*

1659 An elegant proof of the first part of the theorem was given by Westbrook [44]. The second  
 1660 part follows by adapting the lower bound of 3 for deterministic algorithms [6] to the adaptive  
 1661 online setting, and uses an important technique of averaging the cost of 3 offline algorithms.

## 1662 E.5 Transferring Results from Classic Online Algorithms

1663 The lower bounds for classic online algorithms imply lower bounds for time-local algorithms.  
 1664 We first review classic results and algorithms for online file migration, and then provide  
 1665 insights into the case  $\alpha < 1$  in the classic online setting (it is usually assumed that  $\alpha \geq 1$ ).

We study a variant of online file migration in networks consisting of 2 nodes [9]. For this problem, a 3-competitive deterministic algorithm can be obtained using a work function algorithm for metrical task system [14]. The result is tight: a lower bound of 3 holds even for 2 node networks [11], and it uses the technique of averaging costs of multiple offline algorithms, introduced in [32]. In the randomized setting, the threshold work function algorithm obtains the competitive ratio that approaches  $\frac{2e-1}{e-1} \approx 2.581$  as the length of the input sequence grows [30].

More generally, it is known that online file migration in arbitrary networks admits a 4-competitive deterministic algorithm [10]. The best known lower bound for deterministic algorithms is  $3 + \Omega(1)$  and requires 4 nodes [34]. Randomized  $(1 + \phi)$ -competitive and 3-competitive algorithms exist against the oblivious offline adversary and the adaptive online adversary, respectively [44], where  $\phi \approx 1.6$  is the golden ratio. The result against the adaptive online adversary is tight: there is a lower bound of 3 [6]. Against the oblivious offline adversary, a lower bound of  $2 + \frac{1}{2\alpha}$  exists.

Next, we present a lower bound that holds for the classic variant, and hence for the time-local setting as well.

► **Theorem E.15.** *Consider any deterministic online algorithm A for online file migration with file size  $\alpha$ . If A is  $c$ -competitive, then  $c \geq 1 + 1/\alpha$  for  $\alpha \in (0, 1/2]$ , and  $c \geq \min\{2 + 2\alpha, 1 + 3/(2\alpha)\}$  for  $\alpha \in (1/2, 1)$ .*

**Proof.** Consider an input sequence  $\sigma_L$  for any  $L \in \mathbb{N}$ , constructed in the following way. We start by issuing 1-requests until A migrates the file to node 1. Then, we proceed by issuing 0-requests until A migrates the file to the node 0. We repeat these steps  $L$  times. Note that A must eventually perform a migration, otherwise it is not competitive (an optimal offline algorithm pays at most  $2\alpha \cdot L$  for  $\sigma_L$ ). In the remainder of the proof, we assume that A eventually performs a migration, and consequently  $\sigma_L$  is finite.

We partition  $\sigma_L$  into phases  $P_1, \dots, P_L$  in the following way. The first phase begins with the first request and each phase ends when ALG migrates the file to 0. We analyze the ratio of A to OPT on each phase separately. For any  $i \leq L$ , consider the  $i$ th phase  $P := P_i$ . Let  $x$  be the number of 1-requests and  $y$  be the number of 0-requests in  $P$ . Then  $x, y \geq 1$  and  $x + y \geq 2$ . Recall that A first serves a request and then decides whether to migrate the file or not. Consequently, it incurs the cost 2 in each phase for serving requests remotely, and performs two migrations, and its total cost is  $x + y + 2\alpha \geq 2 + 2\alpha$ .

Let OPT be any optimal offline solution. Note that OPT never pays more than  $2\alpha$  in any phase: it can always migrate the file to 1 prior to serving all 1-requests for free, and then to 0 prior to serving all 0-requests for free. Thus, for any  $\alpha > 0$ , we have

$$\frac{\text{ALG}(\sigma_L)}{\text{OPT}(\sigma_L)} \geq \min_i \frac{\text{ALG}(P_i)}{\text{OPT}(P_i)} \geq (2 + 2\alpha)/2\alpha = 1/\alpha + 1.$$

Next, we provide a stronger bound when  $1/2 < \alpha < 1$ , by distinguishing two cases.

**Case 1.** OPT has the file at the node 0 when it enters the phase. If  $x = 1$  then OPT does not benefit by migrating the file, as otherwise, it would incur for 0-requests in addition to  $\alpha$ . Therefore, it pays 1 for serving the (single) 1-request remotely, and the ratio is  $(2 + 2\alpha)/1$ . Else,  $x \geq 2 > 2\alpha$ , and ALG pays  $x + y + 2\alpha \geq 3 + 2\alpha$ . Since OPT never pays more than  $2\alpha$  for any phase, we have  $\text{ALG}(P)/\text{OPT}(P) \geq (3 + 2\alpha)/2\alpha = 3/2\alpha + 1$ .

**Case 2.** OPT has the file at the node 1 when it enters the phase. OPT serves all 1-requests in the phase for free. If  $y = 1$ , then either OPT serves the 0-request remotely without migrating the file, paying 1, or it migrates the file and pays  $\alpha < 1$ . In either case, it pays

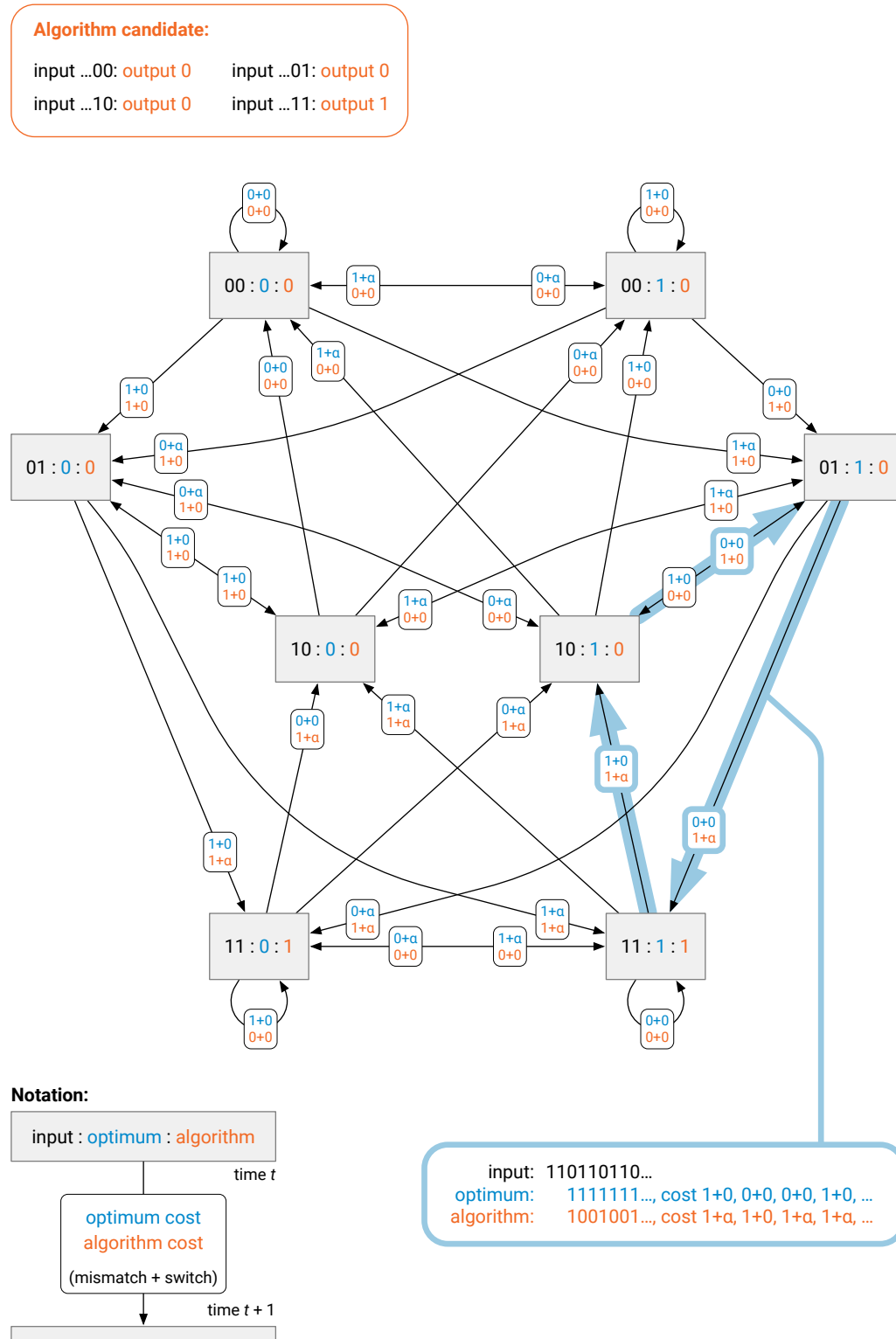
1707      at most 1 and  $\text{ALG}(P)/\text{OPT}(P) \geq (2+2\alpha)/1$ . Else,  $y \geq 2$ , and  $\text{OPT}$  migrates the file to the  
 1708      node 0 before serving the 0-requests; as otherwise it would incur  $y \geq 2 > 2\alpha$ , more than  
 1709      migrating the file twice. Since  $x+y \geq 3$ , we have  $\text{ALG}(P)/\text{OPT}(P) \geq (3+2\alpha)/\alpha = 3/\alpha + 2$ .  
 Hence, in all cases for  $\alpha \in (1/2, 1)$ , we have  $\text{ALG}(P)/\text{OPT}(P) \geq \min\{2+2\alpha, 1+3/(2\alpha)\}$ , and  
 consequently for all inputs  $\sigma_L$  for any  $L \in \mathbb{N}$  we have

$$\frac{\text{ALG}(\sigma_L)}{\text{OPT}(\sigma_L)} \geq \min_i \frac{\text{ALG}(P_i)}{\text{OPT}(P_i)} \geq \min\{2+2\alpha, 1+3/(2\alpha)\}.$$

1710      By combining the results for  $\alpha \in (0, 1/2]$  and  $\alpha \in (1/2, 1)$ , we conclude the lemma.      ◀

1711      A lower bound of 3 is presented in [11] for  $\alpha \geq 1$ , and we note that it holds also for  $\alpha < 1$ .  
 1712      We summarize all known lower bounds in the following corollary.

1713      ► **Corollary E.16.** *No deterministic classic online algorithm for online file migration can*  
 1714      *achieve a competitive ratio less than  $\max\{3, 1+1/\alpha\}$ , for  $\alpha > 0$ , or less than  $\min\{2+2\alpha, 1+$*   
 1715       *$3/(2\alpha)\}$  for  $\alpha \in (0.5, 1)$ .*



■ **Figure 4** Dual de Bruijn graph for  $T = 2$ . The highlighted cycle shows how an adversary can force the candidate algorithm to pay  $3 + 2\alpha$  when optimum pays only 1; hence this specific time-local algorithm cannot be better than  $(3 + 2\alpha)$ -competitive.