Truly Online Paging with Locality of Reference*

Amos Fiat†‡
Tel-Aviv University

Manor Mendel†‡
Tel-Aviv University

November 1, 2001

Abstract

The competitive analysis fails to model locality of reference in the online paging problem. To deal with it, Borodin *et al.* introduced the access graph model for the paging problem, which attempts to capture the locality of reference. However, the access graph model has a number of troubling aspects. The access graph has to be known in advance to the paging algorithm and the memory required to represent the access graph itself may be very large.

In this paper we present truly online strongly competitive paging algorithms in the access graph model that do not have any prior information on the access sequence. We present both deterministic and randomized algorithms. The algorithms need only $O(k \log n)$ bits of memory, where k is the number of page slots available and n is the size of the virtual address space. *I.e.*, asymptotically no more memory than needed to store the virtual address translation table.

We also observe that our truly online algorithms adapt themselves to temporal changes in the locality of reference. We model temporal changes in the locality of reference by extending the access graph model to the so called *extended access graph model*, in which many vertices of the graph can correspond to the same virtual page. We define a measure for the rate of change in the locality of reference in G denoted by $\Delta(G)$. We then show our truly online algorithms remain strongly competitive as long as $\Delta(G) \geq (1+\varepsilon)k$, where k is the cache's size, and no truly online algorithm can be strongly competitive on a class of extended access graphs that includes all graphs with $\Delta \geq k - o(k)$.

*An extended abstract of this paper appeared in [FM97]

†Department of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel. Email: {fiat,mendelma}@tau.ac.il

‡This work has been Supported by United States Israel Bi-national Science Foundation Grant 96-00247/1

1 Introduction

1.1 The Paging Problem and Competitive Analysis

The paging problem is a simplification of an optimization problem that appears in computers system with *virtual memory*. In the paging problem, memory is partitioned to two types: Small and fast memory, called *real memory* and large and slow memory, called *virtual memory*. The memory space is partitioned into equal sized regions, called *pages*: k real memory pages, and n virtual memory pages. Usually n is much larger than k.

Programs address the virtual memory, and the address translation mechanism translates it to real memory address. Whenever a requested (addressed) virtual page is not in the real memory, a *page fault* occurs and the requested page is brought into the real memory. *A page eviction strategy* decides what page is to be evicted from the real memory in order to make room for the requested page. The goal of the strategy is to minimize the number of page faults, and the decisions should be made *online*, i.e., without knowing the future requests. Such a strategy is called *paging algorithm.*

In case the paging algorithm can see the whole request sequence in advance, i.e. it is not an online algorithm, an optimal solution is known for long time [Bel66]: Evict the page whose next use is furthest in the future. This strategy is called OPT.

In their seminal paper, Sleator and Tarjan [ST85] suggest using *competitive analysis* in order to measure the performance of online algorithms. Let $\textsc{On}(k,\sigma)$ denote the number of page faults the online paging algorithm ON does on the sequence σ using k real memory pages, and let $\textsc{Opt}(k,\sigma)$ be the number of page faults done by OPT. ON is called *(strictly) r–competitive* if $\textsc{On}(k,\sigma) \le r \cdot \textsc{Opt}(k,\sigma)$ for every request sequence σ, and the infimum r for which ON is r–competitive is denoted by $r_{\textsc{On}}(k)$. As it is shown in [ST85], the best deterministic competitive ratio for paging is k.

In case the online algorithm is allowed to flip random coins, it is called *randomized online algorithm.* In this case $\textsc{On}(k,\sigma)$ is a random variable and we use $E[\textsc{On}(k,\sigma)]$, the expectation of $\textsc{On}(k,\sigma)$ over the random coin tosses of ON, as the measure for its performance. Hence, a randomized algorithm ON is r–competitive (against oblivious adversary [BDBK$^+$94]) if $E[\textsc{On}(k,\sigma)] \le r \cdot \textsc{Opt}(k,\sigma)$. Fiat *et al.* [FKL$^+$91] proved that the best competitive ratio for randomized paging algorithms is $\Theta(\ln k)$.

1.2 Locality of Reference

Competitive analysis of paging algorithms does not model reality well. It fails to distinguish between algorithms that perform very differently in practice. For example, both "Least Recently Used" (LRU) algorithm and "First In First Out" (FIFO) algorithm have optimal deterministic competitive ratio of k, but in practice LRU out-performs FIFO. Also the "observed competitive ratio" of LRU is usually a constant, i.e. on typical request sequences its performance is worse than OPT by only a constant multiplicative factor [FR97].

A partial explanation for these phenomena is that programs exhibit *locality of reference*. Informally, locality of reference means that pages requested in the near past are likely to be requested in the near future, and at any moment there is usually a small set of pages likely to be requested. The standard competitive analysis does not consider locality of reference, as it treats all possible request sequences the same. Thus, the competitive ratio is likely to be unrealistically high for algorithms that exploit better the locality of reference in the request sequence.

Motivated by this observation, Borodin, Irani, Raghavan and Schieber [BIRS95] suggest incorporating the locality of reference into the competitive analysis. In their model, the set of possible request sequences is limited to only those derived from adjacency relationship on an *access graph.*

An access graph $G = (V, E)$ for a program is a graph that has a vertex for each page in the virtual memory. Locality of reference is imposed by the adjacency relationship in the graph: A page v can be requested immediately after a page u only if there is an edge between u and v in the access graph. Hence, the possible request sequences is limited to those correspond to a path (walk) in the access graph. *Here we consider only undirected access graphs.*

The competitive ratio of a paging algorithm is now dependent on the access graph G, and is defined as

$$r_{\text{ON}}(G,k) = \inf\{r : \ E[\text{ON}(k,\sigma)] \le r \cdot \text{OPT}(k,\sigma) \text{ for every path } \sigma \text{ in } G\}.$$

The deterministic competitive ratio of a paging problem is defined as the competitive ratio of the best deterministic online algorithm, i.e. $r(G,k) = \inf_A r_A(G,k)$, where A range over the *deterministic* online paging algorithms. The randomized competitive ratio is defined as $c_{\mathsf{obl}}(G,k) = \inf_A r_A(G,k)$, where A range over the *randomized* online paging algorithms.

We are interested in *uniform* algorithms that are given the access graph as their input and work efficiently. We adapt the convention from [IKP96] and define a uniform paging algorithm ON to be strongly competitive if its competitive ratio for any paging problem is bounded above by constant times the competitive ratio of the paging problem. I.e., ON should satisfy for every k and G $r_{\text{ON}}(G,k) = O(r(G,k))$ if ON is deterministic, and $r_{\text{ON}}(G,k) = O(r_{\mathsf{obl}}(G,k))$ if ON is randomized.

1.3 Truly Online Algorithms

A practical problematic aspect in algorithms for the access graph model such as FAR, and Fiat and Karlin's algorithms, is the requirement that the access graph will be given in advance. It is problematic because of the following reasons:

- It might be hard to gather the information needed to construct the access graph in advance.
- The access graph requires memory in order of magnitude of between the virtual memory's size and the virtual memory's size squared!

In contrast, algorithms such as LRU, FIFO and RMARK that are "oblivious" to the underline access graph, do not have these problems. We call such algorithms *truly online algorithms.* Formally

Definition 1.1. A uniform paging algorithm ON is called *truly online* if for any k, two access graphs G and G', and request sequence σ consistent with both G and G', $\text{ON}(G,k)$ and $\text{ON}(G',k)$ behave the same on σ.

Classic paging algorithms such as LRU and FIFO are truly online but not strongly competitive, as demonstrated in [BIRS95]. We are interested in strongly competitive algorithms. The existence of truly online strongly competitive algorithms is not at all obvious. However, in this paper we present two paging algorithms, a deterministic algorithm and a randomized algorithm, with the following desirable properties:

1. *They are truly online and strongly competitive.* This implies that the knowledge of the access graph is not necessary for an "almost optimal" online algorithm.
2. Their space requirement is only $O(k \log n)$ bits, compared to naive implementations that stores the whole access graph and needs $\Omega(n^2 \log n)$ bits in the worst case. Using randomization, we can reduce the space requirement even further to $O(k \log k)$ bits.

3. They have rather efficient implementation in hardware. In fact, their hardware requirements is comparable to LRU's requirements.

4. They are adaptive. Once the locality of reference is changed, they adapt themselves to the changes. Unfortunately, the locality of reference in the access graph model is fixed, and therefore the access graph model does not reveal the full power of these algorithms. In the next section we consider a model for "changing locality of reference", which reveals the full power of our algorithms.

1.4 Refined Locality of Reference

We sought a model that allows one to deal with changing behavioral patterns of the access sequence over time. For example, a compiler may run in many stages, with entirely different local behavior in the different stages. Because much of the execution of software is performed in the operating system (I/O processing), a common access graph would show that certain pages are accessed from all over the address space, essentially losing much of the information about locality of reference.

To deal with this we allow multiple appearances of a page. The same page label may appear on many different vertices. The access sequence is constrained to obey the locality conditions imposed by the edge relations in the graph. *I.e.*, every access sequence is derived from a walk on the graph. We call this model the *extended access graph model*.

For a given extended access graph G we define a parameter $\Delta(G)$ to be the shortest path in G between two different vertices labeled with the same page. Intuitively, $\Delta(G)$ indicates "how quickly" the locality of reference changes.

As we shall see, our truly online algorithm are quite good performers with respect to Δ. Specifically, we show that our truly online algorithms are strongly competitive with respect to the family of all extended access graphs with $\Delta(G) \geq (1+\varepsilon)k$. We also prove an almost matching impossibility result: there exists a family of extended access graphs such that $\Delta(G) \geq k - o(k)$ such that no truly online algorithm can be strongly competitive with respect to it.

1.5 Related Work

Borodin *et al.* [BIRS95] introduce the access graph model. They present some basic facts about it and investigate popular algorithms like LRU and FIFO in the context of this model. They show the competitive ratio of LRU is at most twice as the competitive ratio of FIFO for the same problem. They also show that LRU performs badly on access graphs with cycles of size $k+1$. Recently Chrobak and Noga [CN99] proved that LRU is better than FIFO in this model, *i.e.*, $c_{\mathrm{LRU},k}(G) \leq c_{\mathrm{FIFO},k}(G)$ for any access graph G and k.

Borodin *et al.* also consider uniform paging algorithms. They prove the existence of an optimal uniform paging algorithm in PSPACE($|G|$). They give a natural *efficient* uniform paging algorithm, called FAR, and prove that FAR obtains a competitive ratio no worse than $O(\log k)$ times the competitive ratio for the graph. This result is improved in a paper by Irani, Karlin and Phillips [IKP96] in which it is shown that FAR is strongly competitive. The same paper also present a strongly competitive algorithm for a sub-class of directed access graphs, called tree connected directed cycles.

Fiat and Karlin [FK95] present a strongly competitive randomized algorithm against an oblivious adversary, and a strongly competitive algorithm for multi pointer paging (where the page requests come from more than one source). The later gives an alternative deterministic strongly competitive algorithm. The algorithms of [FK95], deterministic and randomized, are the basis of the work presented here.

Karlin, Phillips and Raghavan [KPR92] consider a paging strategy where the input to the paging algorithm is a Markov chain with states correspond to pages, and probabilities $(p_{ij})_{ij}$ such that p_{ij} is the probability page j is referenced just after page i. They show a paging algorithm that is within a constant multiplicative factor of the optimal online algorithm when the page accesses are generated from the Markov chain. A simpler and better algorithm for Markov paging and generalizations was given by Lund, Phillips and Reingold [LPR94].

Fiat and Rosen [FR97] give an access graph based heuristic that is truly online and makes use of a (weighted) dynamic access graph. In this sense we emulate the [FR97] concept. While the [FR97] algorithm is experimentally interesting in that it seems to beat Lru, it is certainly not strongly competitive for any graph and is known to have a competitive ratio of $\Theta(k \log k)$.

Much of the above work is summarized in [BEY98, chap. 3–5].

2 Preliminaries

There is a weaker notion of competitiveness, called *asymptotic competitive ratio*. An online algorithm On is called asymptotically r-competitive if exists C such that for any request sequence σ, $E[\textsc{On}(k,\sigma)] \leq r \cdot \textsc{Opt}(k,\sigma) + C$. We emphasize that in this paper, upper bounds on the competitive ratio are in terms of strict competitive ratio, whereas lower bounds on the competitive ratio are in terms of asymptotic competitive ratio.

A *marking algorithm* [FKL+91] associates one bit with every page slot and works in *phases*. A phase is defined to be a sub-sequence of the page request sequence that includes reference to k different pages and ends just before the $k+1$'th different page is requested (except for the last phase). Marking algorithms obey the rule that when a page is referenced during the phase it is marked and is not evicted from real memory before the end of the phase, when all marks are removed. The only difference between different marking algorithms is the page eviction strategy used for unmarked pages. All the algorithms we consider in this paper are marking algorithms.

We use the term *hole* to denote a page requested during the previous phase, evicted during the current phase and not yet requested during the current phase. A page is called *stale* if it was requested in the previous phase, and not yet requested or evicted in the current phase. A *new page* is a page which has been requested in the current phase, but not in the previous phase. The following lemma clarify the importance of the phase partitioning of request sequences.

Lemma 2.1. *[FKL+91] Assume a phase has g new pages, then then number of page faults* Opt *does in that phase is no less than $g/2$ and no more than g, amortized.*

Borodin *et al.* [BIRS95] presented the following lower bounds. For any tree T let $\ell(T)$ denote the number of leaves in T, and for a graph G, let $\mathcal{T}_i(G)$ denote the set of i-vertex sub-trees of G.

Lemma 2.2 ([BIRS95]). *For any access graph G and cache size k,*

$$r(G,k) \geq \max\{\ell(T) - 1 \,|\, T \in \mathcal{T}_{k+1}(G)\}$$
$$r_{\mathsf{obl}}(G,k) \geq \max\{H_{\ell(T)-1} \,|\, T \in \mathcal{T}_{k+1}(G)\}$$

We get a good estimate of the number of leaves that can be found in a sub-tree of a given graph G by the following folklore proposition:

Proposition 2.3. *Let $G = (V, E)$ be a connected graph with $k < |V| \leq 2k$ and ℓ vertices with degree not equal two. Then there exists a sub-tree of G on $k+1$ vertices and with at least $\ell/30$ leaves.*

[BIRS95] presented another lower bound using the notion of *vine decomposition.*

Definition 2.1 ([BIRS95]). A *vine decomposition* $\mathcal{V} = (B, \mathcal{P})$ of a graph G is a connected sub-graph B together with a set $\mathcal{P} = \{p_1, p_2, \ldots\}$ of vertex disjoint simple paths such that (i) both endpoints of every path in $\mathcal{P}$ are adjacent to vertices in B; (ii) B, p_1, p_2, $\ldots$, are all pair-wise disjoint in their vertices. B is called the *backbone* of $\mathcal{V}$. For a path (vine) p denote by $|p|$ the number of vertices in p *plus one*, *i.e.*, the number of edges in p including those connecting them to B. Define the *value* of vine decomposition $\mathcal{V} = (B, \mathcal{P})$ to be $\nu(\mathcal{V}) \triangleq \sum_{p \in \mathcal{P}} \log |p|$.

Denote by $\mathcal{H}_x(G)$ denote the set of vine-decompositions over x vertices in G.

Lemma 2.4 ([BIRS95]). $r(G, k) \geq \max\{\nu(\mathcal{V}) \,|\, \mathcal{V} \in \mathcal{H}_{k+1}(G)\}$.

When the graph does not contain small cycles (of at most $k+1$ vertices), the following lower bound is useful.

Lemma 2.5 ([IKP96]). *If G contain a vine-decomposition $\mathcal{V} = (B, \mathcal{P})$ on $k+g$ vertices, then*

$$r(G, k) \geq \left\lfloor \max_{p \in \mathcal{P}} \log(|p| - 1) - \log g \right\rfloor /2.$$

An analog lower bound for randomized algorithms, due to Fiat and Karlin [FK95]:

Lemma 2.6 ([FK95]). *For any $(B, \mathcal{P}) \in \mathcal{H}_{k+g}(G)$ with at least $2g$ vertices on its vines,*

$$r_{\mathsf{obl}}(G, k) = \Omega\big(\log(\sum_{p \in \mathcal{P}} |p|) - \log g\big).$$

Also noted in [BIRS95] is

Lemma 2.7. *If G is sub-graph of G' then $r_{\mathsf{obl}}(G, k) \leq r_{\mathsf{obl}}(G', k)$ and $r(G, k) \leq r(G', k)$.*

Dynamic Access Graph

The algorithms presented here are similar to Fiat and Karlin's algorithms [FK95], but they do not have the access graph available. Instead, they will make use of a spanning tree of the pages requested during the previous phase, as the access graph.

Let P denote the pages requested in the previous phase. Let $G_P = (P, E)$, where $E = \{uv | u, v \in P$ requested successively in the previous phase, and $u \neq v\}$. Let $G_0 = (V_0{=}P, E_0)$ be a spanning tree of G_P. Let r_0 denote the last page requested in the previous phase and let r_i, $i \geq 1$, denote the ith page request in the current phase. Define $G_{i+1} = (V_{i+1}, E_{i+1})$, where $V_{i+1} = V_i \cup \{r_{i+1}\}$, and $E_{i+1} = E_i \cup \{r_i r_{i+1}\} \setminus \{r_i r_i\}$.

Like Fiat and Karlin' algorithms, our algorithms are marking algorithms and have three sub-phases with a different page eviction strategy in each one of them. Let G_{II} denote G_i at the end of sub-phase II, and G_{III} denote G_i at the end of the phase.

For any vertex $v \in V_0$ $(= P)$, We define $N(v)$ to be the sequence of the vertices in V_0 ordered in non-decreasing distance from v in G_0. We also denote by c_v, at any given time, the maximum index such that $\{N(v)_1, \ldots, N(v)_{c_v}\}$ are all evicted and unmarked pages (holes).

3 Randomized Algorithms

We remark that if one uses asymptotic competitive ratio then there is a straightforward truly online algorithm as follows: Take any marking strongly competitive algorithm and execute it on "the observed access graph" so far *i.e.* the graph that has all the edges used so far in the request sequence. On phases in which the access graph did not changed these algorithms faults at most g times the competitive ratio of the observed graph. On phases in which the access graph do change they fault at most k times, and there are at most n^2 such phases. Therefore, using $C = n^2k$ as the constant additive term, we get "strongly competitive algorithm". Such a solution, besides not being strictly competitive and having a large additive term, would require $\Omega(n^2)$ memory.

In this section we present and analyze RTO, a truly online randomized paging algorithm. In the first two sub-phases of RTO a vine-decomposition is constructed in $G_{II} \subseteq G$, such that requested and evicted vertices are the backbone of the vine-decomposition. The third sub-phase evicts vertices randomly from the vines.

Algorithm Rto(k)**.** The algorithm is a marking algorithm that partition the phase into three consecutive sub-phases. In each sub-phase it does as below. We emphasize that the graph referred to in the following discussion is G_0, *i.e.*, a spanning tree of G_P.

Subphase I: On a fault, evict a random unmarked unevicted page v such that v has degree $\neq 2$. If there is no such page v, and the phase is not over, proceed to sub-phase II.

Subphase II: All unmarked, unevicted pages in the real memory lie on degree-2 paths in G_0. Choose v such that

1. v has been evicted in sub-phase I, and
2. $N(v)_{c_v+1}$ exists and has not yet been marked, and
3. v minimizes c_u amongst all candidate vertices u meeting conditions 1-2 above.

If no v meets the criteria above, proceed to the next sub-phase. Otherwise, evict $N(v)_{c_v+1}$.

Subphase III: On a fault, evict a random unmarked page.

Competitive Analysis

The analysis here follows the analysis of the randomized algorithm from [FK95].

Assume that g new pages are requested during the entire phase, and let f_i denote the expected number of pages evicted during sub-phase i that will be requested (later) during the phase. We note that the expected total number of faults in the phase is at most $f_1 + f_2 + f_3 + g$. We will show that $f_i = O(g \cdot r_{\mathsf{obl}}(G_{II}, k))$. As G_{II} is a sub-graph of the underlying access graph G, it follows that RTO is strongly competitive on G.

Sub-phase I faults: Let $C \subseteq P$ be the set of vertices in P with degree $\neq 2$ in G_0. Let $S \subseteq C$ be the set of vertices evicted in sub-phase I. Suppose the adversary requested vertices in S, during the phase in the order $s_1, s_2, \ldots, s_m$.

Proposition 3.1. *The probability that* RTO *has a hole at s_i at the time it is requested, is at most $g/(|C| - i + 1)$.*

Proof. RTO has, at any point in time, at most g holes. Inductively, when s_i was requested, the holes were evenly distributed among the vertices in $C \setminus \{s_1, \dots, s_{i-1}\}$. When s_{i+1} is requested the holes are evenly distributed in $C \setminus \{s_1, \dots, s_i\}$. If s_{i+1} was requested during sub-phase I itself, this follows from the page eviction strategy of sub-phase I. If s_{i+1} was requested after sub-phase I, then the vertices in $C \setminus \{s_1, \dots, s_i\}$ are all holes. In any case we have that the probability for a hole in s_i is at most $g/(|C| - i + 1)$. □

The expected number of faults on C is therefore no more than $gH_{|C|}$. G_1 is a tree with $\Omega(|C|)$ leaves and by Lemma 2.2, $r_{\mathsf{obl}}(G_1, k) = \Omega(H_{|C|})$. Thus, $f_1 = O(g \cdot r_{\mathsf{obl}}(G_1, k))$.

Sub-phase II faults: Let C' denotes the holes in C at the beginning of sub-phase II. $|C'| \leq g - 1$. We say that a vertex $v \in C'$ *dies* when $N(v)_{c_v}$ is marked. Let c'_v denotes c_v at the time immediately before $v \in C'$ dies. Note that $f_2 \leq \sum_{v \in C'} c'_v$. Note also that at any moment during sub-phase II, (i) $\sum c_v < g$ where the sum ranges over the "live" vertices in C', and (ii) $|c_v - c_u| \leq 1$ for any two "live" vertices in C'. Denote by v_i the ith vertex that dies in C'. We conclude that $c'_{v_i} \leq \lceil g/(|C| - i + 1) \rceil$. Hence $f_2 \leq g + gH_{|C|} = O(f_1) = O(g \cdot r_{\mathsf{obl}}(G_1, k))$.

Sub-phase III faults: At the beginning of this sub-phase, all unmarked, unevicted vertices lie on degree-2 paths Π in G_0, which are also degree-2 paths in G_{II} (any u s.t. $\deg_{G_{II}}(u) \neq \deg_{G_0}(u)$ must have been marked by now). We denote by B the sub-graph of G_{II} spanned by $(P \cup P') \setminus \Pi$, where P are the vertices requested in the previous phase, and P' are the vertices requested in sub-phases I and II. The endpoints of the degree-2 paths in Π are adjacent to B.

Proposition 3.2. *B is connected.*

Proof. P' are the marked vertices. Clearly, the sub-graph spanned by P' is connected, and Π contains no vertex from P', so P' is contained in some connected component of B.

Assume, for the sake of contradiction, that B has more than one connected component. There must be another connected component X, and a vertex $v \in X$ such that (i) v was evicted during sub-phase I, (ii) $N(v)_{c_v+1}$ exists (because there are marked vertices) and (iii) $N(v)_{c_v+1}$ is unmarked, otherwise v is in the connected component of P'. This implies that sub-phase II is not done yet, because $N(v)_{c_v+1}$ could be evicted, a contradiction. □

Thus (B, Π) is a vine-decomposition in G_{II} on at most $k + g$ vertices. Let L denote the number of vertices in Π. If $L \leq 3g$ then $f_3 = O(g)$ because RTO can have at most one fault on the vertices in Π. Otherwise, denote by g' the number of holes on Π at the end of the phase. Clearly, $g' \leq g$, and at most $L - g'$ vertices from Π can be requested during this sub-phase. As in sub-phase I, the probability that the ith requested vertex in Π ($1 \leq i \leq L - g'$) has a hole is at most $g'/(L - i + 1)$. Thus,

$$f_3 \leq g'(H_L - H_{g'}) \leq g(H_L - H_g).$$

By Lemma 2.6, $r_{\mathsf{obl}}(G_{II}, k) = \Omega(\log L - \log g)$. Therefore $f_3 = O(g \cdot r_{\mathsf{obl}}(G_{II}, k))$.

We conclude,

Theorem 1. RTO *is strongly competitive on any underlying access graph.*

4 Deterministic Algorithms

DTO is similar to the deterministic algrotihm from [FK95], but instead of using a known access graph G, it makes use of the dynamic tree G_0. Again, the first two phases construct a vine-decomposition in G_{II} such that requested and evicted vertices form the backbone of the vine-decomposition.

Subphase III evicts vertices from the middle of unmarked degree–2 *segments* in these paths. There may be more than one unmarked degree–2 segment in these paths, as the adversary may request an *arbitrary* vertex in an unmarked degree–2 segment, effectively decomposing the segment into two.

As in RTO, all graph relations described herein relate to G_0.

Subphase I: On a fault, evict any unmarked page p such that p has degree $\neq 2$. If there is no such page p, and the phase is not over, proceed to sub-phase II.

Subphase II: All unmarked, unevicted pages in the real memory lie on degree-2 paths Π in G_0. On a fault, choose v such that

1. v has been evicted in sub-phase I, and
2. $N(v)_{c_v+1}$ exists and has not yet been marked.

If no v meets the criteria above, proceed to the next sub-phase. Otherwise, evict $N(v)_{c_v+1}$.

Subphase III: On a page fault DTO chooses a maximal (w.r.t. containment) unmarked degree 2 segment in the paths of Π, and evicts an unmarked page closest to the middle of the segment.

Competitive Analysis

Let g be the number of new vertices requested during the entire phase, and let f_i denote the number of pages evicted during sub-phase i. As in the case of RTO, the total number of faults in the phase is at most $f_1 + f_2 + f_3 + g$. We will show that $f_i = O(g \cdot r(G_{III}, k))$.

Sub-phases I & II. Let $C \subseteq P$ be the set of vertices in P with degree $\neq 2$ in G_0. Note that for each $v \in C$, at most g vertices associated with v were evicted during subphases I and II. This so since after g evictions one of the evicted vertices must have been requested (otherwise there were more than g new pages in the phase). Thus $f_1 + f_2 \leq g \cdot |C|$. From Proposition 2.3, G_1 is a tree on $k+1$ vertices with $\Omega(|C|)$ leaves and by Lemma 2.2 $r(G_1, k) = \Omega(|C|)$, and therefore $f_1 + f_2 = O(gr(G, k))$.

Sub-phase III faults: At the beginning of this sub-phase, all unmarked, unevicted vertices lie on degree two paths Π in G_0, which are also degree two paths in G_{II}. We denote by B the sub-graph of G_{II} spanned by $(P \cup P') \setminus \Pi$, where P are the vertices requested in the previous phase, and P' are the vertices requested in sub-phase I and II.

Proposition 4.1. *B is connected.*

Proof. Similar to the proof of Proposition 3.2. □

The endpoints of the degree-2 paths in Π are adjacent to B, hence (B, Π) is a vine decomposition in G_{II}. Unlike the case in [FK95], where the algorithm faults at most $g \log |p|$ on every vine $p \in \Pi$, DTO might fault on *every* vertex on *every* $p \in \Pi$. Nonetheless, we prove in Section 5 the following:

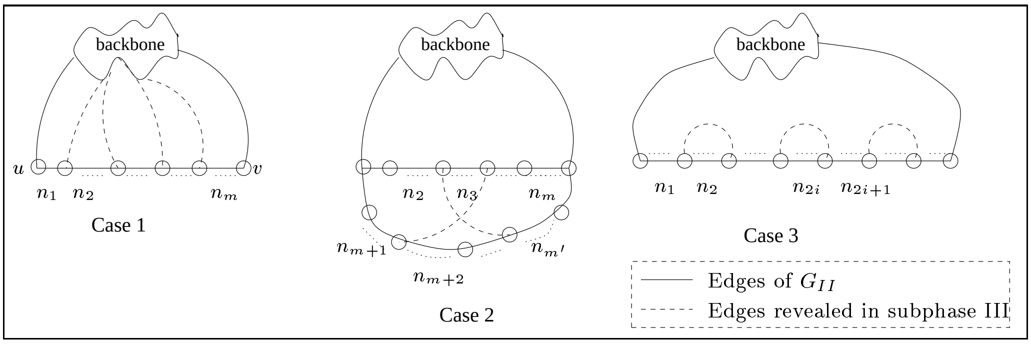

Figure 1: Possible scenarios during sub-phase III.

Lemma 4.2. $f_3 = O(g \cdot r(G_{III}, k))$.

We conclude:

Theorem 2. Dto *is strongly competitive on any underlying access graph.*

5 Proof of Lemma 4.2

Our proof of Lemma 4.2 is quite lengthy. To make the exposition simpler it is partitioned as follows: Section 5.1 presents the complications in proving the lemma and gives some intuition. Section 5.2 introduces the notation used throughout the proof. Section 5.3 provides the proof, leaving out some combinatorial lemmata. Section 5.4 ends the exposition by providing the missing proofs.

5.1 Informal Exposition

First we should note that the situation here is quite different from the randomized case. In the randomized case the upper bound on the number of faults is *not influenced* by the new edges revealed in sub-phase III. In contrast, in the deterministic case, the added edges can increase the number of faults. For example, in case 1 in Fig. 1, at the end of sub-phase II we have a vine $u, \ldots, v$ in the vine decomposition Π in G_{II}. Hence, the naive lower bound for the number of faults in this vine is $g(\log \sum_i n_i)$, whereas Dto might fault there almost $g(\sum_i \log n_i)$, which can be much higher. In this example, the solution is clear — we should construct a new vine decomposition that uses the new edges as part of the backbone and has a value of $\Omega(\sum_i \log n_i)$. *I.e.*, we improve the lower bound on the number of faults of Opt to match the upper bound.

Case 2 in Fig. 1 is more complex. Here the construction of a new vine decomposition is not obvious. The scenario addressed here includes cases where new edges connect one G_{II} vine to another. These new edges split the vines into sub-paths. If the lengths of the resulting sub-paths were $(n_i)_i$, then one upper bound on the number of faults for Dto during sub-phase III would be $g \sum_i \log n_i$. We will prove that in this scenario it is possible to build a new vine decomposition with a value of $c \sum_i \log n_i$ for some global constant $c > 0$. Again, we have found matching upper and lower bounds.

The situation becomes more complicated when the new edges do not cross path boundaries, as in case 3 in Fig. 1. In this case we can not hope to construct a vine decomposition with value $\Omega(\sum_i \log n_i)$, such a vine decomposition simply does not exist. Here we will have to show that the upper bound on the number of faults for Dto is indeed $O(\log(\sum_i n_i) + \sum_i \log n_{2i})$, which is smaller than $\sum_i \log n_i$.

The difference between cases 1,2 in Fig. 1 and case 3 is that in cases 1 and 2 we used a simple upper bound on the number of faults and could devise an appropriate vine decomposition for the lower bound on the competitive ratio. In case 3 we need a more sophisticated upper bound as well as more complicated way of building the vine decomposition for the lower bound.

5.2 Preliminaries

During sub-phase III new pages might be requested (at most $g-1$ new pages). As we can associate an amortized cost of $\Omega(1)$ to any offline algorithm for every new page, we would like to "ignore" them, but we need to consider the connectivity relations they add.

Definition 5.1. The *simplification* of G_{III} is a graph denoted by $G^s = (V, E)$, such that V is set set of vertices of G_{II} and E includes the edges of G_{II} *and* an edge uv if there exists a simple path between $u, v \in V$ with all its internal vertices are of vertices *not in* V, i.e. new vertices requested during sub-phase III.

As mentioned before, the vine decomposition at the end of sub-phase II in G_{II} may not give us a sufficiently high lower bound. In order to differentiate it from the vine-decomposition that we will construct, we call it the *backbone bi-connected path complex* in G^s, or simply *the complex.*

Given a graph G we denote the set of vertices of G as $V[G]$. Given $G = (V, E)$ and $U \subseteq V$ we denote the sub-graph induced by G on U as $G|_U$. Given a simple path $p = \langle v_1, \ldots, v_k \rangle$ we define the inner subpath $I(p) = \langle v_2, v_3, \ldots, v_{k-1} \rangle$.

Definition 5.2. A *proper path* p is a path in G^s such that edges with one endpoint in $V[I(p)]$ have their other endpoint in $V[p]$.

Note that the new edges added to G_{II} during the course of sub-phase III decompose the vines in the complex into disjoint sub-paths. We view this decomposition as an hierarchical process as follows:

1. We "add" to the decomposition all the new edges that cross path boundaries, which results in a decomposition to proper sub-paths.

2. For every resulting sub-path we recursively construct a new decomposition.

We now formally define the concepts *decomposition* and *recursive decomposition*:

Definition 5.3. Given a complex $\mathcal{C} = (B, \mathcal{Q})$, a *separating set* for $\mathcal{C}$ is a set S of vertices, $S \subseteq \cup_{q \in \mathcal{Q}} V[q]$, where $\deg_{G^s}(v) \geq 3$ for all $v \in S$.

Definition 5.4. Given a complex $\mathcal{C} = (B, \mathcal{Q})$ and a separating set S for $\mathcal{C}$, we define the *decomposition* $\mathcal{D} = (S, \mathcal{P})$ of complex $\mathcal{C}$ as follows:

Choose $q \in \mathcal{Q}$, let $q = \langle v_1, v_2, \ldots, v_k \rangle$. Let $S \cap V[q] = \{v_{i_1}, v_{i_2}, \ldots, v_{i_j}\}$, where $i_\ell > i_{\ell-1}$, $1 \leq \ell \leq j$. Define the paths

$$\begin{aligned}
p_1 &= \langle v_1, v_2, \ldots, v_{i_1-1} \rangle, \\
p_2 &= \langle v_{i_1+1}, v_{i_1+2}, \ldots, v_{i_2-1} \rangle, \\
&\vdots \\
p_j &= \langle v_{i_{j-1}+1}, v_{i_{j-1}+2}, \ldots, v_{i_j-1} \rangle, \\
p_{j+1} &= \langle v_{i_j+1}, v_{i_j+2}, \ldots, v_k \rangle.
\end{aligned}$$

Let $\mathcal{P}(q)$ be the set of all the non-empty p_ℓ paths, $1 \le \ell \le j+1$. Let $\mathcal{P}$ be the union of all $\mathcal{P}(q)$, $q \in \mathcal{Q}$. S is also called the separating set of $\mathcal{D}$, and it is denoted by $S[\mathcal{D}]$.

A *proper decomposition* is a decomposition such that all paths $p \in \mathcal{P}$ are proper paths.

Definition 5.5. Given a proper path $p = \langle v_1, v_2 \ldots, v_k\rangle$, we define a *separating set* for p to be *non-empty* set S of vertices, $S \subseteq p$, $S = \{v_{i_1}, v_{i_2}, \ldots, v_{i_j}\}$, where for all $1 \le \ell \le j$ it holds that

1. (a) $\deg_{G^s}(v_{i_\ell}) \ge 3$, or
   (b) $\deg_{G^s}(v_{i_\ell}) = 2$ and there is no edge between the sets $\{v_1, \ldots, v_{i_\ell - 1}\}$ and $\{v_{i_\ell+1}, \ldots, v_k\}$ in G^s.
2. For $1 \le \ell \le j-1$, If $i_\ell < i_{\ell+1} - 1$, then both $\deg_{G^s}(v_{i_\ell}) \ge 3$ and $\deg_{G^s}(v_{i_{\ell+1}}) \ge 3$.

Additionally, we require that if $i_1 > 1$ then $\deg_{G^s}(v_{i_1}) \ge 3$, and if $i_j < k$ then $\deg_{G^s}(v_j) \ge 3$.

Definition 5.6. Given a proper path $p = \langle v_1, v_2 \ldots, v_k\rangle$, and a separating set for p, $S = \{v_{i_1}, v_{i_2}, \ldots, v_{i_j}\}$, the *decomposition* $\mathcal{D} = (S, \mathcal{P})$ of a proper path p is defined as follows:

Let $p_1, \ldots, p_{j+1}$ denote the paths

$$\begin{aligned} p_1 &= \langle v_1, v_2, \ldots, v_{i_1 - 1}\rangle, \\ p_2 &= \langle v_{i_1+1}, v_{i_1+2}, \ldots, v_{i_2-1}\rangle, \\ &\vdots \\ p_j &= \langle v_{i_{j-1}+1}, v_{i_{j-1}+2}, \ldots, v_{i_j - 1}\rangle, \\ p_{j+1} &= \langle v_{i_j+1}, v_{i_j+2}, \ldots, v_k\rangle. \end{aligned}$$

Let $\mathcal{P}$ be the set of all the non-empty p_ℓ paths, $1 \le \ell \le j+1$. S is also called the separating set of $\mathcal{D}$, and it is denoted by $S[\mathcal{D}]$ or $S[p]$ if $\mathcal{D}$ is clear from the context.

A *proper decomposition* is a decomposition such that all paths $p \in \mathcal{P}$ are proper paths.

Definition 5.7. A *recursive decomposition* $\mathcal{D}^R(q)$ of a proper path q is a proper decomposition $\mathcal{D} = (S, \mathcal{P})$ of q, along with recursive decompositions $\mathcal{D}^R(p)$, for each $p \in \mathcal{P}$. We define the *value* of $\mathcal{D}^R(q)$ recursively as

$$\tilde{v}(\mathcal{D}^R(q)) = \log|q| + \sum_{p\in\mathcal{P}} \tilde{v}(\mathcal{D}^R(p)).$$

Definition 5.8. A *recursive decomposition* $\mathcal{D}^R(\mathcal{C})$ *of a complex* $\mathcal{C}$ is a proper decomposition $\mathcal{D} = (S, \mathcal{P})$ of the complex $\mathcal{C} = (B, \mathcal{Q})$ along with recursive decompositions $\mathcal{D}^R(p)$, for each proper path $p \in \mathcal{P}$. The *value* of $\mathcal{D}^R(\mathcal{C})$ is defined as

$$\tilde{v}(\mathcal{D}^R(\mathcal{C})) = \sum_{p\in\mathcal{P}} \tilde{v}(\mathcal{D}^R(p)).$$

We use the shorthand $\mathcal{D}^R$ when the complex $\mathcal{C}$ or the proper path q is implicitly understood.

Definition 5.9. We define $\mathcal{P}[\mathcal{D}^R]$ to be the set of all sub-paths in the recursive decomposition $\mathcal{D}^R$, including sub-paths defined recursively. We also define $\mathcal{P}_g[\mathcal{D}^R] = \{p \in \mathcal{P}[\mathcal{D}^R] : \ |p| \ge g\}$, and

$$\tilde{v}_g(\mathcal{D}^R) = \sum_{p\in\mathcal{P}_g[\mathcal{D}^R]} (\log|p| - \log g)\,.$$

Proposition 5.1. *Let $\mathcal{D} = (S_0, \mathcal{P}_0)$ be a proper decomposition of a complex $\mathcal{C} = (B, \mathcal{Q})$, and let $\mathcal{D}^R$ be a recursive decomposition induced by it. Then the following holds:*

1. *For all $p, q \in \mathcal{P}[\mathcal{D}^R]$, $S_0 \cap S[p] = \emptyset$ and $S[p] \cap S[q] = \emptyset$.*
2. *$S_0 \cup \bigcup_{p \in \mathcal{P}[\mathcal{D}^R]} S[p] = \bigcup_{q \in \mathcal{Q}} V[q]$.*
3. *$\tilde{v}(\mathcal{D}^R) = \sum_{p \in \mathcal{P}[\mathcal{D}^R]} \log |p|$.*
4. *$\tilde{v}_1(\mathcal{D}^R) = \tilde{v}(\mathcal{D}^R)$.*
5. *$|\mathcal{P}[\mathcal{D}^R]| \leq l + t$, where $l = |\{v \in \bigcup_{q \in \mathcal{Q}} V[q] : \deg_{G^s}(v) \geq 3\}|$, and $t = |\mathcal{Q}|$.*

Proof. Items 1–4 follow immediately from the definitions. To prove 5 we first argue that for any proper path p, $|\mathcal{P}(\mathcal{D}^R(p))| \leq \ell + 1$, where ℓ is the number of degree 3 or more vertices in p. Next, we sum up over all $p \in \mathcal{P}_0$, getting an extra t. □

Example 5.1. Consider the following recursive decomposition: the first level consist of a separating set the includes *all* the vertices on the paths with degree at least 3. The rest are degree-2 sub-paths (which are proper). Each degree-2 sub-path is decomposed so that all its vertices are in the separating set.

5.3 The Proof

The Upper Bound

Lemma 5.2. *Let $\mathcal{C} = (B, \mathcal{Q})$, $|\mathcal{Q}| = t$, be the complex in G^s induced by* Dto *at the end of sub-phase II. Given $\mathcal{D}^R$, a recursive decomposition of the complex $\mathcal{C}$, then*

$$f_3 = O\left(g(\ell + t + \tilde{v}_g(\mathcal{D}^R))\right),$$

where g is the number of new pages in the phase, and ℓ is the number of vertices in $\bigcup_{q \in \mathcal{Q}} q$ with degree at least 3 in G^s.

Proof. There are at most g faults not on the paths (on new vertices requested for the first time during sub-phase III).

We count the number of faults on the paths by charging faults to vertices of degree ≥ 3 in G^s or to paths in the recursive decomposition of the complex.

If the fault is on a vertex of degree ≥ 3 in G^s, then we charge this fault to the vertex. There are at most ℓ such faults.

Otherwise, the fault is on a vertex v of degree 2 in G^s. From claim 5.1, v must be in some $S[p]$ for some path $p \in \mathcal{P}(\mathcal{D}^R)$. It cannot be in the separating set of the complex itself because all vertices in the separating set of the complex are of degree ≥ 3 in G^s. p is a subpath of some $q \in \mathcal{Q}$, this follows from the definition of the decomposition.

We charge path p for the fault on v. We want to show that there are at most $O(g \cdot \max\{\log |p| - \log g, 1\})$ faults associated with p. If this is true then the proof of the lemma follows by summing up all faults associated with paths and all faults associated with vertices of degree ≥ 3 or new vertices.

Let $S^2(p)$ denote the set of all vertices of degree 2 in $S[p]$. Let $U(p) \subseteq S^2(p)$ denote the set of unmarked vertices in $S^2(p)$, over time, when unmarked vertices are requested, they are removed from $U(p)$.

For $X \subseteq q$, let $C(X)$ denotes the minimal sized subpath of q that contains all vertices in X. We use the notation $C(U(p))$ to denote a set whose size may decrease over time (as $U(p)$ itself is a set whose size may decrease over time).

Proposition 5.3. *All vertices in $C(U(p))$ are unmarked.*

Proof. The proof follows from the fact that p is a proper path. Choose arbitrary $u, v \in S^2(p)$, $u \neq v$, if $w \in C(\{u,v\})$ and $x \notin C(\{u,v\})$ then any path from x to w must pass through either u or v.

Thus, if any vertex in $C(\{u,v\})$ was requested this implies that either u or v was requested, *i.e.*, either u or v is marked.

Taking u and v to be the extreme points of $C(U(p))$ (which must also be in $S^2(p)$) concludes the proof of the claim. ☐

Let $M(U(p))$ denote the longest unmarked subpath of $q \in \mathcal{Q}$ containing $C(U(p))$, as before $M(U(p))$ varies over time. The algorithm DTO evicts a vertex from the mid point of $M(U(p))$.

Proposition 5.4. *After $g+1$ evictions from $S^2(p)$, $|M(U(p))| \leq 2|p|$*

Proof. We first claim that after any fault in $S^2(p)$, p includes at least one of the extreme points of $M(U(p))$. This follows because $C(U(p)) \subseteq M(U(p)) \subseteq q$, $C(U(p)) \subseteq p \subseteq q$, so $p \cap M(U(p)) \neq \emptyset$. As p is a subpath of q and $M(U(p))$ is a subpath of q, either p is a subpath of $M(U(p))$ or an extreme point of $M(U(p))$ is in p. p cannot be a subpath of $M(U(p))$ because p contains a marked vertex whereas $M(U(p))$ consists only of unmarked vertices. Thus, it must be that an extreme point of $M(U(p))$ is in p.

There must be at least one fault in $S^2(p)$ prior to the $(g+1)$ eviction from $S^2(p)$. One the next eviction from $S^2(p)$ following the first fault from $S^2(p)$, we know that the midpoint of $M(U(p))$ is in $S^2(p) \subseteq p$ whereas one endpoint of $M(U(p))$ is in p, thus $|M(U(p))| \leq 2|p|$. ☐

The $g+1$ evictions from $S^2(p)$ described in the lemma above can cause at most $g+1$ faults in $S^2(p)$. This means that we have associated at most $g+1$ faults with p prior to the configuration where $|M(U(p))| \leq 2|p|$.

We now count the number of evictions from this point onwards, this is a bound on the number of faults in $S^2(p)$.

After every g evictions, the size of $M(U(p))$ decreases by a factor of roughly $1/2$. So long as $|M(U(p))| \geq 10g$, this factor is at most $6/10$. Thus, after $O(g(\log|p| - \log(10g)))$ evictions we have $|M(U(p))| \leq 10g$. On the remaining vertices we can fault at most $10g$ times, giving us a total number of faults on this stage of $O(g \cdot \max\{\log|p| - \log g, 1\})$.

As described above, taking a summation over all paths $p \in \mathcal{P}(\mathcal{D}^R)$, and adding the faults on degree 3 vertices and new vertices concludes the proof of the lemma. ☐

The Lower Bound

The idea is to construct a vine decomposition $\mathcal{V} = (B, \mathcal{Q})$ on $k+1$ vertices (see Definition 2.1) whose value matches the value of *some* recursive decomposition $\mathcal{D}^R$, up to a constant factor:

$$\nu(\mathcal{V}) \geq \Omega(1) \cdot \tilde{v}(\mathcal{D}^R). \tag{1}$$

If this is true, then from Lemma 5.2, Lemma 2.2, Proposition 2.3, and Lemma 2.4 it follows that the competitive ratio that the adversary can force upon any online algorithm is no worse than the competitive ratio of DTO.

Consider a decomposition $\mathcal{D} = (S, \mathcal{P})$ (this decomposition is either a decomposition of a proper path or of a complex). As a first step towards obtaining our goal of the previous paragraph, we seek a vine-decomposition $\mathcal{V} = (B, \mathcal{P}')$ such that

1. The set of vines in $\mathcal{V}$ is a subset of the set of paths in $\mathcal{D}$.
2. The backbone of the vine decomposition $\mathcal{V}$ includes the separating set S of the decomposition and includes the paths of the decomposition that were not chosen as vines.
3. The value of $\mathcal{V}$ is no less than a constant fraction of the value of $\mathcal{D}$.

Definition 5.10. A *proper vine decomposition* $\mathcal{V}$ of a *proper path* p is a vine decomposition of the graph induced by G^s on the vertices of p, such that the vines are sub-paths of p, and the endpoints of p are in the backbone of $\mathcal{V}$.

Definition 5.11. A *vine selection* of a decomposition $\mathcal{D} = (S, \mathcal{P})$ of a proper path q is a set $A(\mathcal{D}) = \mathcal{P}' \subseteq \mathcal{P}$ (called vines) such that

- The induced graph on $S \cup \bigcup_{p \in \mathcal{P} \setminus A(\mathcal{D})} V[p]$ is connected.
- $\sum_{p \in A(\mathcal{D}))} \log |p| \geq c \sum_{p \in \mathcal{P}} \log |p|$, where $c = 1/32$.
- The endpoints of q are in $S \cup \bigcup_{p \in \mathcal{P} \setminus A(\mathcal{D})} V[p]$.

It follows that $(\bigcup_{p \in (\mathcal{P} \setminus A(\mathcal{D}))} V[p] \bigcup S,\ A(\mathcal{D}))$ is a proper vine decomposition of q.

Definition 5.12. A *vine selection* of a decomposition $\mathcal{D} = (S, \mathcal{P})$ of a complex $\mathcal{C} = (B, \mathcal{Q} = \{q_1, \ldots, q_t\})$ is a set $A(\mathcal{D}) = \mathcal{P}' \subseteq \mathcal{P}$ such that

- The induced graph on $\bigcup_{p \in \mathcal{P} \setminus A(\mathcal{D})} V[p] \bigcup B \bigcup S$ is connected.
- $\sum_{p \in A(\mathcal{D})} \log |p| \geq c \sum_{p \in \mathcal{P}} \log |p|$, where $c = 1/32$.

It follows that $(\bigcup_{p \in (\mathcal{P} \setminus A(\mathcal{D}))} V[p] \bigcup B \bigcup S,\ A(\mathcal{D}))$ is a vine decomposition.

To construct a vine decomposition as required in Equation (1), we make use of a special type of decomposition, called an *irreducible decomposition*.

Definition 5.13. Given a path $p = \langle v_1, \ldots, v_m \rangle$, denote the *span* of an edge $e = v_i v_j$ between two vertices on the path by $|e|_p = |j - i|$. An *irreducible path* p is a proper path in which for every edge e whose endpoints are on p, $|e|_p < |p|^{1/4}$. An *irreducible decomposition* is a proper decomposition $\mathcal{D} = (S, \mathcal{P})$ such that all paths $p \in \mathcal{P}$ are irreducible.

The following Lemma shows that it is possible to construct an irreducible decomposition along with a related vine selection.

Given a simple path $p = \langle v_1, v_2, \ldots, v_k \rangle$, a maximal degree two subpath of p is a subpath $q = \langle v_i, v_{i+1}, \ldots, v_j \rangle$, $1 \leq i \leq j \leq k$, such that the $\deg_{G^s}(v_\ell) = 2$ for all $i \leq \ell \leq j$ while if $i > 1$ then $\deg_{G^s}(v_{i-1}) > 2$, and if $j < k$ then $\deg_{G^s}(v_{j+1}) > 2$.

Lemma 5.5. *1. Given a proper path q and assuming that every maximal degree two subpath is of length ≥ 15, then q has an irreducible decomposition $\mathcal{D} = (S, \mathcal{P})$ and a related vine selection $A(\mathcal{D})$.*

2. *Given a complex $\mathcal{C} = (B, \mathcal{Q})$, and assuming that for all $q \in \mathcal{Q}$ every maximal degree two subpath of q is of length ≥ 15, then $\mathcal{C}$ has an irreducible decomposition $\mathcal{D} = (S, \mathcal{P})$ and a related vine selection $A(\mathcal{D})$.*

The proof of the lemma appears in Section 5.4.

We are now ready to construct the required recursive decomposition and vine decomposition, as required in Equation (1). We give a constructive algorithm that builds both simultaneously, the algorithm makes use of recursive decompositions.

1. use Lemma 5.5 to obtain an irreducible decomposition $\mathcal{D} = (S, \mathcal{P})$ and related vine selection $A(\mathcal{D})$;
2. recursively find for every sub-path $p \in \mathcal{P}$, a vine decomposition and recursive decomposition;
3. using the resulting recursively obtained vine decompositions and the vine selection $A(\mathcal{D})$, we construct the required vine decomposition (Lemma 5.12 in Section 5.4).

Lemma 5.6 summarizes the construction for proper paths, the construction for a complex is handled in Lemma 5.7.

Lemma 5.6. $\exists c'$, $c \geq c' > 0$, *such that any proper path q, with maximal degree 2 paths of lengths at least* 15, *has a recursive decomposition $\mathcal{D}^R$ along with a proper vine decomposition $\mathcal{V}$ of q such that* $\nu(\mathcal{V}) \geq c' \cdot [\tilde{v}(\mathcal{D}^R) - \log |q|]$.

The proof appears in Section 5.4. Lemma 5.6 is used to construct the vine-decomposition.

Lemma 5.7. *Given a complex $\mathcal{C} = (B, \mathcal{Q} = \{q_1, \ldots, q_t\})$ in G^s, and let ℓ be the number of vertices $x \in \bigcup_{q \in \mathcal{Q}} V[q]$ such that* $\deg_{G^s}(x) \geq 3$. *There exists a recursive decomposition $\mathcal{D}^R$ and vine-decomposition $\mathcal{V}$ of it such that* $\max\{\ell + t, \nu(\mathcal{V})\} = \Omega(\ell + t + \tilde{v}(\mathcal{D}^R))$.

Proof. First we change G^s by adding new degree-2 vertices in such a way that every maximal degree-2 path has a length of at least 15 (we need at most $14(l+t)$ new vertices). Denote the new graph G', the complex $\mathcal{C}$ in G^s naturally induces a complex $\mathcal{C}' = (B, \mathcal{Q}' = \{q'_1, \ldots, q'_t\})$ in G'. From Lemma 5.5 we have an irreducible decomposition $\mathcal{D}' = (S, \mathcal{P})$ of the complex, along with a vine selection $A(\mathcal{D}')$. The vine selection $A(\mathcal{D}')$ induces a vine decomposition, $\mathcal{V}'_1$, on G' such that $\nu(\mathcal{V}'_1) \geq c \sum_{p \in \mathcal{P}} \log |p|$.

From Lemma 5.6, we have for each path $p \in \mathcal{P}$, a recursive decomposition $\mathcal{D}^R(p)$ and a proper vine decomposition $\mathcal{V}'_2(p)$ such that $\nu(\mathcal{V}'_2(p)) \geq c'[\tilde{v}(\mathcal{D}^R(p)) - \log |p|]$.

We construct a new vine decomposition for G', $\mathcal{V}'_2$. The set of vines in $\mathcal{V}'_2$ is the union of the vines in all the $\mathcal{V}'_2(p)$, $p \in \mathcal{P}$. The backbone of $\mathcal{V}'_2$ is the union of the sets B, B'_p, $p \in \mathcal{P}$, and S, where B is the backbone of the complex $\mathcal{C}' = (B, \mathcal{Q}')$, B'_p is the backbone of the proper vine decomposition of $V'_2(p)$, $p \in \mathcal{P}$, and S is the separating set of $\mathcal{D}'$.

To show that $\mathcal{V}'_2$ is indeed a vine decomposition we need to show that the backbone is connected and that all vines are connected to the backbone at their endpoints.

First we show that the B is connected to all vertices of S. This follows because any two successive separating points $x_i, x_{i+1} \in S$, are either neighbors or connected to opposite endpoints of some path $p \in \mathcal{P}$, the endpoints of p are connected via the backbone of $V'_2(p)$, thus x_i and x_{i+1} are connected via the backbone of $\mathcal{V}'_2$. It follows inductively that all the separating points are connected to B.

Within every path $p \in \mathcal{P}$ we have a valid vine decomposition, whose backbone is connected to B via vertices in S, thus $\mathcal{V}_2'$ is itself a vine decomposition. It also follows that $\nu(\mathcal{V}_2') = \sum_{p\in\mathcal{P}} \nu(\mathcal{V}_2'(p))$.

Let $\mathcal{V}' = \mathcal{V}_1'$ if $\nu(\mathcal{V}_1') \geq \nu(\mathcal{V}_2')$, and let $\mathcal{V}' = \mathcal{V}_2'$ otherwise. Let $\mathcal{D}'^R$ be a recursive decomposition of the complex $\mathcal{C}' = (B, \mathcal{Q}')$ defined in the natural way as $\mathcal{D}'^R = (S, \{\mathcal{D}^R(p) :\ p \in \mathcal{P}\})$.

$$\begin{aligned}\nu(\mathcal{V}') &= \max\{\nu(\mathcal{V}_1'), \nu(\mathcal{V}_2')\} \geq \tfrac{1}{2}\nu(\mathcal{V}_1') + \tfrac{1}{2}\nu(\mathcal{V}_2') \\ &\geq \tfrac{1}{2}c \sum_{p\in\mathcal{P}} \log|p| + \tfrac{1}{2}c' \sum_{p\in\mathcal{P}} \left[\tilde{v}(\mathcal{D}^R(p)) - \log|p|\right] \\ &\geq \tfrac{1}{2}c' \sum_{p\in\mathcal{P}} \tilde{v}(\mathcal{D}^R(p)) = \tfrac{c'}{2} \cdot \tilde{v}(\mathcal{D}'^R)\end{aligned}$$

Define $\mathcal{D}^R$ and $\mathcal{V}$ on G^s, by removing from $\mathcal{D}'^R$ and $\mathcal{V}'$ the artificial vertices. $\mathcal{D}^R$ is still a recursive decomposition (although not necessarily irreducible), $\mathcal{V}$ is still a vine decomposition, and $\tilde{v}(\mathcal{D}'^R) \geq \tilde{v}(\mathcal{D}^R)$. There are at most $\ell + t$ vines in $\mathcal{V}'$ and the value of every each is reduced by at most $\log 16 = 4$, so $\nu(\mathcal{V}) \geq \nu(\mathcal{V}') - 4(\ell + t)$. Therefore

$$5(\ell + t) + \nu(\mathcal{V}) \geq (\ell + t) + \nu(\mathcal{V}') \geq \ell + t + \tfrac{c'}{2}\tilde{v}(\mathcal{D}'^R) = \Omega(\ell + t + \tilde{v}(\mathcal{D}^R)). \qquad \square$$

The vine decomposition $\mathcal{V}$ in Lemma 5.7 is on a graph of $k + g$ vertices, and thus we are not quite done yet. In the following lemma we either reduce the number of vertices in $\mathcal{V}$ to $k + 1$, or find a "big cycle" in it. The proof appears in Section 5.4.

Lemma 5.8. *Under the conditions of Lemma 5.7 one of the following happens:*

- *There exists a vine decomposition* $\mathcal{V}'$ *with at least* g *vertices less than* $\mathcal{V}$*, such that* $\max\{\ell + t, \nu(\mathcal{V}')\} = \Omega(\ell + t + \tilde{v}(\mathcal{D}^R))$;
- *There exists a vine* V *such that* $\max\{\ell + t, \log|V| - \log g\} = \Omega(\ell + t + \tilde{v}_g(\mathcal{D}^R))$.

We conclude:

Lemma 4.2. $f_3 = O(g \cdot r(G_{III}, k))$.

Proof. At the end of sub-phase II, Π induces a vine decomposition. This vine decomposition defines the complex $(B, \{q_1, \ldots, q_t\})$ on the simplification graph G^s.

Let $\mathcal{D}^R$ and $\mathcal{V}$ be the recursive decomposition and the vine decomposition obtained by Lemma 5.7. From Lemma 5.2 we know that $f_3 = O(g(\ell + t + \tilde{v}_g(\mathcal{D}^R)))$. Hence, we are left to see that $\ell + t + \tilde{v}_g(\mathcal{D}^R) = O(r(G_{III}, k))$.

Claim: $\ell = O(r(G_{III}, k))$. This is so since by Proposition 2.3 there exists a tree on $k + 1$ vertices with $\Omega(\ell)$ leaves, so by Lemma 2.2, $\ell = O(c_k(G_{III}))$.

Claim: $t = O(r(G_{III}, k))$. This is since when disconnecting the paths $q_1, \ldots, q_t$ in their middle, and by removing $g - 1$ vertices from these paths, we get a backbone on $k + 1$ vertices with at least $t/2$ leaves, so $t = O(c_k(G_{III}))$.

From Lemma 5.8 we have one of the two:

- A vine decomposition $\mathcal{V}'$ in G^s with at most $|V(G^s)| - (g-1)$ vertices such that $\ell + t + \tilde{v}_g(\mathcal{D}^R) = O(\ell + t + \nu(\mathcal{V}'))$. $\mathcal{V}'$ naturally induces a vine decomposition in G_{III} (with the same name) with the same value and with at most $|V(G_{III})| - (g - 1) \leq k + 1$ vertices. From Lemma 2.4, $\nu(\mathcal{V}') = O(r(G_{III}, k))$. Thus, $\ell + t + \tilde{v}_g(\mathcal{D}^R) = O(r(G_{III}, k))$.
- A vine V in G^s, which is also a vine in G_{III}, that satisfies $\ell + t + \tilde{v}_g(\mathcal{D}^R) = O(l + t + (\log|V| - \log g))$. From Lemma 2.5, $\log|V| - \log g = O(r(G_{III}, k))$, thus $\ell + t + \tilde{v}_g(\mathcal{D}^R) = O(r(G_{III}, k)))$.

$\square$

5.4 Proofs of the Combinatorial Lemmata

In this section we supply the omitted proofs from the previous section.

Proof of Lemma 5.5

Constructing the irreducible decomposition is done by way of *refinement.*

Definition 5.14. A *refinement* of a decomposition $\mathcal{D} = (S, \mathcal{P})$ is a decomposition $\mathcal{D}' = (S', \mathcal{P}')$ of the same object such that $S \subsetneq S'$.

The next two lemmas show how to construct an irreducible decomposition along with a related vine selection.

Lemma 5.9. *For every proper path p there exists a decomposition $\mathcal{D} = (S, \mathcal{P})$, and a vine selection $A(\mathcal{D})$, such that S includes the endpoints of p.*[1]

Proof. Let $p = \langle v_1, \ldots, v_{|p|} \rangle$. S is constructed in two stages. First, we take all degree 2 vertices in p such that no edge "covers" them. *I.e.*, vertices v_i such that there is no edge $e = v_j v_l \in E$ with $j < i < l$. At this point, Either $S = \emptyset$, or this is a proper decomposition. We consider every resulting proper sub path individually, finding a decomposition and vine selection. Combining the decompositions and vine selections for the sub paths gives the required result.

From now on, assume $p' = \langle v_1, \ldots, v_m \rangle$ is a proper path, that all its vertices are either of degree greater than 2 or are covered by some edge whose endpoints are in $V[p']$.

We choose a set of edges $E' = \{e_1, \ldots, e_{m'}\}$ by induction as follows: let $l_1 = 1$, $e_1 = v_{l_1} v_{r_1}$ for the maximum available r_1. Assume inductively that $e_1 = v_{l_1} v_{r_1}, \ldots, e_{i-1} = v_{l_{i-1}} v_{r_{i-1}}$, where $r_{i-1} < m$, were already chosen. Consider the edge $e = v_l v_r$ such that $l \leq r_{i-1}$ and r is maximized under this condition. Note that $r > r_{i-1}$. If $r_{i-1} - l \leq r - r_{i-1}$, we set $e_i = e$. Otherwise, consider the edge $e' = w_{l'} w_{r'}$ such that $l' \leq r$ and r' is maximized under this condition. Note that $r' > r$. We set $e_i = e'$.

We continue until v_m is reached, and denote by $r_{m'} = m$. It is easily seen that $\forall i$ $(1 \leq i \leq m')$, $r_{i-1} < l_{i+1}$ (taking $r_0 = 1$, $l_{m'+1} = m$), and that both $(l_i)_i$ and $(r_i)_i$ are strictly increasing sequences.

We construct $\mathcal{D}$ by adding to S the endpoints of the edges in E'. Let $p'_1, \ldots, p'_{s'}$ denote the resulting decomposed sub-paths, in their order in p'. The vine selection $A(\mathcal{D})$ is constructed by taking roughly half of the sub-paths as vines, as described below (see also Fig. 2). For $2 \leq i \leq m'$:

1. If $r_{i-1} < l_i$, we declare the sub-path $\langle v_{r_{i-1}+1}, \ldots, w_{l_i - 1} \rangle$ as sub-backbone.
2. Otherwise, we declare the shorter of the two sub-paths $\langle v_{1+\max\{l_{i-1}, r_{i-2}\}}, \ldots, v_{l_i - 1} \rangle$ and $\langle v_{l_i + 1}, \ldots, v_{r_{i-1} - 1} \rangle$ as sub-backbone.

$A(\mathcal{D})$ consists of all the other sub-paths, not declared as sub-backbones. It is easily seen that the sub graph induced on $S \cup \bigcup_{p'_i \notin A(\mathcal{D})} p'_i$ is connected and includes the endpoints of p'_i.

In case (1), the sub-path $v_{\max\{l_{i-1}, r_{i-2}\}+1}, \ldots, v_{r_{i-1}-1}$ would be vine selected, and would be bigger than the sub-backbone $v_{r_{i-1}+1}, \ldots, v_{l_i - 1}$. In case (2) The sub-path not declared as sub-backbone is vine-selected and is bigger. Hence every sub-path in the backbone, has an adjacent sub-path in the vine-selection which is bigger. This means that we have constructed a mapping

[1] $\mathcal{D}$ is not necessarily proper. Properness will be dealt in Lemma 5.10.

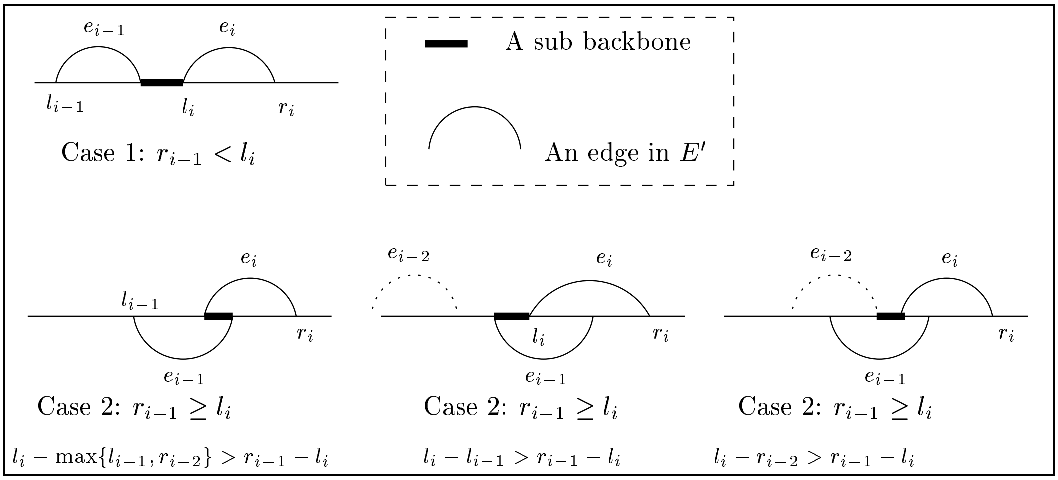

Figure 2: The different cases in the proof of Lemma 5.9. In case 1 we have a sub backbone between r_{i-1} and l_i, which is smaller than the vine between $\max\{r_{i-2}, l_{i-1}\}$ and r_{i-1}. In case 2 the sub backbone is chosen again to be smaller from an adjacent vine.

$f : \{p'_1, p'_2, \ldots, p'_{s'}\} \to A(\mathcal{D})$, such that $|f^{-1}(\{p'_i\})| \leq 3$ for all $p'_i \in A(\mathcal{D})$, and $|p'_i| \leq |f(p'_i)|$ for all $i\{1, \ldots, s'\}$. Therefore

$$\sum_{p'_i \in A(\mathcal{D})} \log(|p'_i| + 1) \geq \frac{1}{3} \sum_{i=1}^{s'} \log(|p'_i| + 1). \qquad \square$$

Lemma 5.10. *Given an initial decomposition $\mathcal{D}^I = (S^I, \mathcal{P}^I)$ and a vine selection $A(\mathcal{D}^I)$, and assuming that every maximal degree-2 sub-path of a path $p \in \mathcal{P}$ has length of at least* 15. *Then there exists a refinement irreducible decomposition $\mathcal{D} = (S, \mathcal{P})$ and a vine selection $A(\mathcal{D})$.*

Note: The lemma holds, both for a decomposition of the complex, and for a decomposition of a proper path. We denote by B the backbone of the complex and we assume $B = \emptyset$ for a decomposition of a proper path.

Proof. The proof is by induction. Given a decomposition $\mathcal{D} = (S, \mathcal{P})$ and a vine selection $A(\mathcal{D})$ such that $\sum_{p \in A(\mathcal{D})} \log |p| \geq c \sum_{p \in \mathcal{P}} \log |p|$, if $\mathcal{D}$ is an irreducible decomposition then we are done. Otherwise, there exists an edge $e = uv$ that contradicts $\mathcal{D}$ being proper or irreducible. We call e a violating edge.

In this case we build a refinement $\mathcal{D}' = (S', \mathcal{P}')$ from $\mathcal{D}$ as follows.

- $S' = S \cup (\{u, v\} \setminus B)$.
- If $u, v \in p$, for some $p \in \mathcal{P}$, we decompose p to three sub-paths: p_1, p_2 and p_3. We define $\mathcal{P}' = \mathcal{P} \cup \{p_1, p_2, p_3\} \setminus \{p\}$.
- If $u \in p$, $v \in q$, such that $p, q \in \mathcal{P}$ and $p \neq q$, we decompose p and q to two sub-paths: p_1, p_2, q_1, q_2. We define $\mathcal{P}' = \mathcal{P} \cup \{p_1, p_2, q_1, q_2\} \setminus \{p, q\}$.

$\mathcal{D}'$ is obviously a legal decomposition. Moreover, the set of non-violating edges strictly increases (with e becoming non-violating). Therefore, this process is finite, and at the end we get an irreducible decomposition.

We are left to find a vine selection $A(\mathcal{D}')$ to $\mathcal{D}'$ such that

$$\sum_{p' \in A(\mathcal{D}')} \log(|p'|+1) \geq c \sum_{p' \in \mathcal{P}'} \log(|p'|+1). \tag{2}$$

Denote

$$\begin{aligned} \tilde{\Delta} &= \sum_{p' \in \mathcal{P}'} \log(|p'|+1) - \sum_{p \in \mathcal{P}} \log |p|, \\ \Delta &= \sum_{p' \in A(\mathcal{D}')} \log(|p'|+1) - \sum_{p \in A(\mathcal{D})} \log |p|. \end{aligned}$$

Note that $\tilde{\Delta} \geq 0$. We will see how to choose $A(\mathcal{D}')$ such that $\Delta \geq c\tilde{\Delta}$ (proving equation (2)), and the sub graph spanned by $B \cup S' \cup \bigcup_{p' \notin A(\mathcal{D}')} p'$ remains connected.

We denote by $B(\mathcal{D})$ the graph on edges of the initial decomposition plus the violating edges added so far, that both their end points are in $B \cup S \cup \bigcup_{p \in \mathcal{P} \setminus A(\mathcal{D})} p$. $B(\mathcal{D}')$ is defined analogously for edges with endpoints in $B \cup S' \cup \bigcup_{p' \in \mathcal{P}' \setminus A(\mathcal{D}')} p'$. *I.e.*, $B(\mathcal{D})$ is the backbone of the vine decomposition in the old decomposition, and $B(\mathcal{D}')$ is the backbone of the vine decomposition after the refinement.

We use the notation $n_i = |p'_i| + 1$. From the assumptions of the lemma, $n_i \geq 16$, and we make use of fact that for $n \geq 16$, $\log n - 2 \geq \frac{1}{2} \log n$.

We do a case analysis according to the places of u and v. The different cases are illustrated in Fig. 3.

1. Both u and v are not on paths in $\mathcal{D}$ (they are in $S \cup B$). In this case e is not a violating edge, which is impossible.

2. One of u or v is on a path in $\mathcal{D}$ and the other is not. In this case $|\mathcal{P}'| = |\mathcal{P}| + 1$. Assume $u \in p$, for some $p \in \mathcal{P}$, and $v \in S \cup B$, so $p = p'_1 \cup \{u\} \cup p'_2$, where $p'_1, p'_2 \in \mathcal{P}'$. $\tilde{\Delta} = \log \frac{n_1 n_2}{n_1 + n_2}$.

   (a) If $p \in A(\mathcal{D})$, then we construct $A(\mathcal{D}') = A(\mathcal{D}) \cup \{p'_1, p'_2\} \setminus \{p\}$. $B(\mathcal{D}')$ is connected, and $\Delta = \log \frac{n_1 n_2}{n_1 + n_2}$, so $\Delta \geq \tilde{\Delta}$.

   (b) If $p \notin A(\mathcal{D})$, then there is a simple path in $B(\mathcal{D})$ between u and v. Without loss of generality, assume the path pass through p'_2. Construct $A(\mathcal{D}') = A(\mathcal{D}) \cup \{p'_2\} \setminus \{p\}$. $B(\mathcal{D}')$ is connected, and $\Delta = \log n_2$, so $\Delta \geq \log n_2 + \log \frac{n_1}{n_1 + n_2} = \tilde{\Delta}$.

3. u, v are on different paths, $u \in p$ and $v \in q$, for some $p, q \in \mathcal{P}$ and $p \neq q$. In this case $|\mathcal{P}'| = |\mathcal{P}| + 2$. Assume $p = p'_1 \cup \{u\} \cup p'_2$ and $q = p'_3 \cup \{v\} \cup p'_4$. $\tilde{\Delta} = \log \frac{n_1 n_2 n_3 n_4}{(n_1+n_2)(n_3+n_4)}$.

   (a) If $p, q \in A(\mathcal{D})$. Without loss of generality n_1 is the minimum of $\{n_1, n_2, n_3, n_4\}$. Construct $A(\mathcal{D}') = A(\mathcal{D}) \cup \{p'_2, p'_3, p'_4\} \setminus \{p, q\}$. Obviously $B(\mathcal{D}')$ is connected, and $\Delta = \log \frac{n_2 n_3 n_4}{(n_1+n_2)(n_3+n_4)}$. Without loss of generality, assume $n_3 \leq n_4$, so $0 \geq \log \frac{n_2 n_4}{(n_1+n_2)(n_3+n_4)} \geq -2$. Thus

$$\begin{aligned} \Delta = \log n_3 + \log \frac{n_2 n_4}{(n_1+n_2)(n_3+n_4)} &\geq \log n_3 - 2 \geq \tfrac{1}{2} \log n_3 \geq \\ \tfrac{1}{4}(\log n_3 + \log n_1) &\geq \tfrac{1}{4}\left[\log n_3 + \log n_1 + \log \frac{n_2 n_4}{(n_1+n_2)(n_3+n_4)}\right] = \tfrac{1}{4}\tilde{\Delta}. \end{aligned}$$

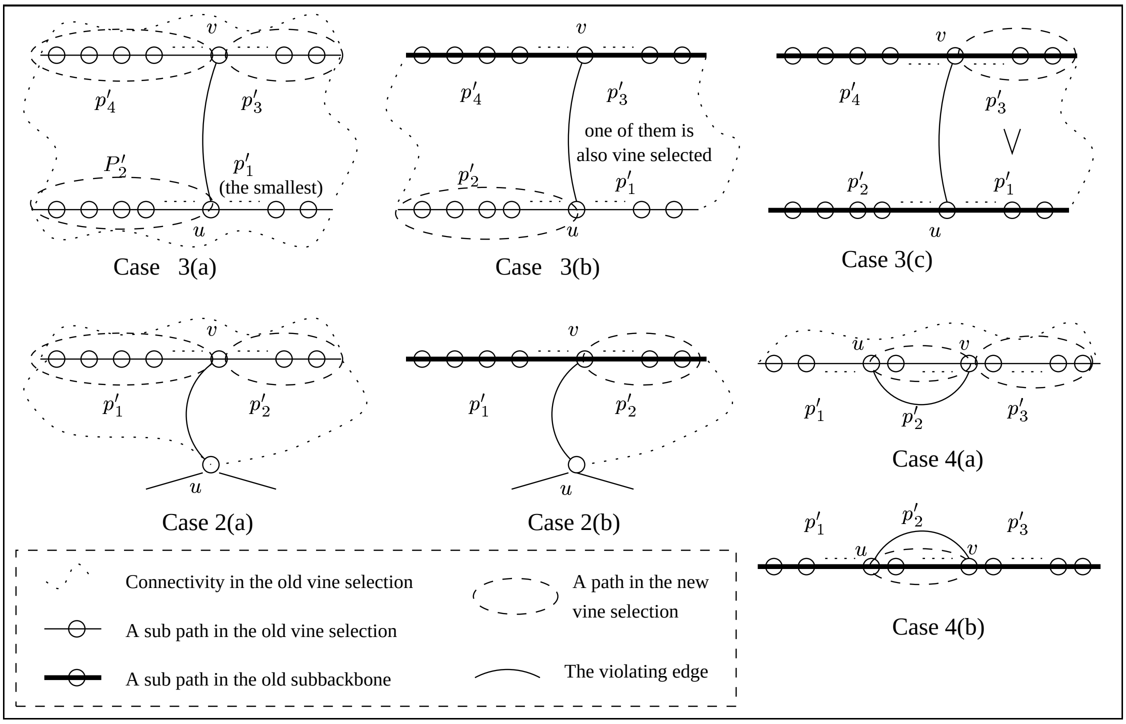

Figure 3: The different cases in Lemma 5.10. In case 2, the edge $e = uv$ is between a path and the backbone. In case 3, e is between two different paths. In case 4, e is inside the same path. The different sub-cases corresponds to the different possibilities of the paths to be either in the vine-selection, or as sub-backbones.

(b) If one of $\{p, q\}$ is in $A(\mathcal{D})$ and the other is not. Without loss of generality assume $p \in A(\mathcal{D})$, and $q \notin A(\mathcal{D})$. Without loss of generality, $n_1 \leq n_2$. p'_1 has two endpoints, one of which is u. The other endpoint of p'_1 is connected via a simple path in $B(\mathcal{D})$ to v. Without loss of generality, assume this path includes p'_3. If $n_1 \geq n_3$ then we construct $A(\mathcal{D}') = A(\mathcal{D}) \cup \{p'_1, p'_2\} \setminus \{p\}$. If, on the other hand, $n_1 < n_3$ then we construct $A(\mathcal{D}') = A(\mathcal{D}) \cup \{p'_3, p'_2\} \setminus \{p\}$. In either case, $B(\mathcal{D}')$ is connected, and

$$\Delta = \log \frac{\max\{n_1, n_3\} \cdot n_2}{n_1 + n_2} = \log \max\{n_1, n_3\} + \log \frac{n_2}{n_1 + n_2} \geq$$
$$\tfrac{1}{2} \max\{\log n_1, \log n_3\} \geq \tfrac{1}{4}(\log n_1 + \log n_3) \geq \tfrac{1}{4}(\log n_1 + \log \min\{n_3, n_4\}) \geq$$
$$\geq \tfrac{1}{4}\left[\log n_1 + \log \min\{n_3, n_4\} + \log \frac{n_2 \max\{n_3, n_4\}}{(n_1 + n_2)(n_3 + n_4)}\right] = \tfrac{1}{4}\tilde{\Delta}$$

(c) If $p, q \notin A(\mathcal{D})$. Then there is a simple path in $B(\mathcal{D})$ from u to v. Without loss of generality, assume it goes through p'_1, and p'_3. Without loss of generality, assume $n_1 \leq n_3$. Construct $A(\mathcal{D}') := A(\mathcal{D}) \cup \{p'_3\}$. $B(\mathcal{D}')$ is connected, and

$$\Delta = \log n_3 \geq \frac{1}{2}\left[\log \min\{n_1, n_2\} + \log \min\{n_3, n_4\}\right] \geq \tfrac{1}{2}\tilde{\Delta}$$

4. u, v are on the same path. In this case $|\mathcal{P}'| = |\mathcal{P}| + 2$. Without loss of generality, assume $p = p'_1 \cup \{u\} \cup p'_2 \cup \{v\} \cup p'_3$, for some $p \in \mathcal{P}$ and $p'_1, p'_2, p'_3 \in \mathcal{P}'$. Here $e = uv$ does not contradict

$\mathcal{D}$ being proper, so it must contradict $\mathcal{D}$ being irreducible. Therefore $n_2 \geq (n_1+n_2+n_3)^{1/4}$, so $\log n_2 \geq \frac{1}{4}\log\max\{n_1,n_2,n_3\}$. Without loss of generality, assume that $n_1 \leq n_3$. $\tilde{\Delta} = \log\frac{n_1 n_2 n_3}{n_1+n_2+n_3}$.

(a) If $p \in A(\mathcal{D})$, then construct $A(\mathcal{D}') = A(\mathcal{D}) \cup \{p'_2, p'_3\} \setminus \{p\}$.

$$\begin{aligned}\Delta = \log\frac{n_2 n_3}{n_1+n_2+n_3} = \log\min\{n_2,n_3\} + \log\frac{\max\{n_2,n_3\}}{n_1+n_2+n_3} \geq \\ \geq \log\min\{n_2,n_3\} - \log 3 \geq \tfrac{1}{4}\log\min\{n_2,n_3\} \geq \\ \geq \tfrac{1}{32}(\log n_1 + \log\min\{n_2,n_3\}) \geq \tfrac{1}{32}\tilde{\Delta}\end{aligned}$$

(b) If $p \notin A(\mathcal{D})$, then construct $A(\mathcal{D}') = A(\mathcal{D}) \cup \{p'_2\}$.

$$\Delta = \log n_2 \geq \tfrac{1}{8} 2\log\max\{n_1,n_2,n_3\} \geq \tfrac{1}{8}\tilde{\Delta} \qquad \square$$

We Combine lemmata 5.9 and 5.10:

Lemma 5.5. *1. Given a proper path q and assuming that every maximal degree two subpath is of length ≥ 15, then q has an irreducible decomposition $\mathcal{D} = (S,\mathcal{P})$ and a related vine selection $A(\mathcal{D})$.*

2. Given a complex $\mathcal{C} = (B,\mathcal{Q})$, and assuming that for all $q \in \mathcal{Q}$ every maximal degree two subpath of q is of length ≥ 15, then $\mathcal{C}$ has an irreducible decomposition $\mathcal{D} = (S,\mathcal{P})$ and a related vine selection $A(\mathcal{D})$.

Proof. We first find an initial decomposition (not necessarily proper) and related vine selection as required by Lemma 5.10. For a proper path, Lemma 5.9 gives the needed initial decomposition and vine selection. For a complex $(B,\mathcal{Q})$, we take an initial decomposition $(\emptyset,\mathcal{Q})$ and the related vine selection is $\mathcal{Q}$. Using Lemma 5.10 we obtain an irreducible decomposition and related vine selection. $\square$

Proof of Lemma 5.6

First we need the following definitions:

Definition 5.15. Denote $M = |q|$. Define *"good" vine decomposition* $\mathcal{V}$ of a proper path q related to a recursive irreducible decomposition $\mathcal{D}^R(q)$, to be a vine decomposition of q such that

- For every vine $V \in \mathcal{V}$, $|V| < M^{1/4}$.
- $\nu(\mathcal{V}) \geq c_M \cdot \left[\tilde{v}(\mathcal{D}^R(q)) - \log M\right]$.

Where c_m is defined recursively

$$c_m = \begin{cases} c/4 & m < 8^4; \\ c_{\lceil m^{1/4}\rceil} \cdot \left(1 - \frac{5}{m^{3/16}}\right) & \text{otherwise.}\end{cases}$$

Proposition 5.11. *There exists $c' > 0$ such that for all m, $c_m \geq c'$.*

Proof. Define $c'_j = \frac{c}{4}\prod_{i=2}^{j}(1 - \frac{5}{8^{(4^i)\cdot 3/16}})$, and $c' = \lim_{j\to\infty} c'_j$. It is easy to see by induction that $(c_m)_{m>0}$ and $(c'_j)_{j>0}$ are monotonously non-increasing and $c_{8^{4^j}} = c'_j$. Also $c' > 0$, because $1 > \frac{5}{8^{(4^i)\cdot 3/16}} \geq 0$ and $\sum_{i=1}^{\infty}\frac{5}{8^{(4^i)\cdot 3/16}} < \infty$. $\square$

Thus, constructing a recursive irreducible decomposition along with "good" vine decomposition would suffice to prove Lemma 5.6. The following Lemma is the inductive argument that allows the construction.

Lemma 5.12. *Let $\mathcal{D}^R(q) = (S, \{\mathcal{D}^R(p_1), \ldots, \mathcal{D}^R(p_s)\})$ be a recursive irreducible decomposition of a proper path q. Assume that:*

- *There is a vine selection $A(\mathcal{D})$ of $\mathcal{D}$.*
- *For every sub-path p_i, there is a "good" vine decomposition $\mathcal{V}(p_i)$ related to $\mathcal{D}^R(p_i)$.*

Then we can construct $\mathcal{V}$, a "good" vine decomposition related to $\mathcal{D}^R(q)$.

Proof. Denote $m_i = |p_i| + 1$ and $M = |q|$. From the irreducibility property of $\mathcal{D}^R(q)$, $m_i \leq M^{\frac{1}{4}}$. For every $p \in A(\mathcal{D})$ we do the following:

Let $m = |p|$. From the assumptions, the vines in $\mathcal{V}(p)$ have lengths at most $m^{1/4}$. We partition p to $t = \lceil m/(5m^{1/4}) \rceil = \lceil m^{3/4}/5 \rceil$ consecutive sub-paths $S_1, \ldots, S_t$, each of size between $(5-2)m^{1/4}$ and $(5+2)m^{1/4}$, such that no sub-path partitions a vine in $\mathcal{V}(p)$. Let X_i be the sum of the values of the vines in S_i. Thus $\sum_i X_i = \nu(\mathcal{V}(p))$. Let $S = S_{i_1}$ be the sub-path that satisfies $X = X_{i_1} = \min_i\{X_i\}$, so $X \leq \frac{5}{m^{3/4}}\nu(\mathcal{V}(p))$. S is partitioned into three consecutive sub-paths $S^{\text{lft}}, S^{\text{mid}}, S^{\text{rgt}}$ such that $|S^{\text{mid}}| = m^{1/4}$, and $|S^{\text{lft}}| \geq m^{1/4}$, $|S^{\text{rgt}}| \geq m^{1/4}$.

Let p^{lft}, p^{rgt} denotes the sub-paths p that are to the left and to the right (respectively) of S^{mid} in p. Let $B(p)$ denotes the backbone of $\mathcal{V}(p)$. We construct $\mathcal{V}^{\text{lft}}$, a vine decomposition of p^{lft} by defining B^{lft}, the backbone of $\mathcal{V}^{\text{lft}}$, to be $B^{\text{lft}} = (B(p) \cup S^{\text{lft}}) \cap p^{\text{lft}}$. We claim that $\mathcal{V}^{\text{lft}}$ is a legal vine decomposition, and that the endpoints of p^{lft} are part of B^{lft}. It is easy to see that the endpoints of the vines in $\mathcal{V}^{\text{lft}}$ are adjacent to B^{lft} and the endpoints of p^{lft} are part of B^{lft}.

We are left to prove that the sub-graph spanned by B^{lft} is connected. First, note $\exists v_l \in S^{\text{lft}} \cap B(p)$, otherwise S^{lft} is part of a vine in $\mathcal{V}(p)$ and its length is at least $m^{1/4}$, which contradicts the assumptions. The sub-graph induced on S^{lft} is obviously connected. For every vertex $u \in B^{\text{lft}} \setminus S^{\text{lft}}$, there is a *simple path* p in $B(p)$ from u to v_l. Consider the maximal prefix of p which is entirely inside p^{lft}. The end of the maximal prefix must be inside S^{lft}, otherwise there is an edge e in p such that $|e|_p \geq |S^{\text{lft}}| = m^{1/4}$, in contradiction to p being irreducible. Thus B^{lft} is connected.

In conclusion, $\mathcal{V}^{\text{lft}}$ is a legal vine decomposition of p^{lft}. Similarly, we construct a vine decomposition $\mathcal{V}^{\text{rgt}}$ of p^{rgt} such that the endpoints of p^{rgt} are part of the backbone of $\mathcal{V}^{\text{rgt}}$.

We construct $\mathcal{V}$, a vine decomposition of q such that every vertex in a sub-path $p \notin A(\mathcal{D})$ will have the same role (part of the backbone or part of a vine) as in $\mathcal{V}(p)$. For $p \in A(\mathcal{D})$ we declare $S^{\text{mid}}(p)$ to be a vine, and the other vertices, to have the same role as in $\mathcal{V}^{\text{lft}}(p)$ and $\mathcal{V}^{\text{rgt}}(p)$. $\mathcal{V}$ is a vine decomposition of q as it is seen from the previous discussion and because the endpoints of the sub paths are part of the backbone (see Def. 5.10). Also, every vine of $\mathcal{V}$ has length less than $M^{1/4}$.

Because $c_{m'}$ is non-increasing as function of m', $c_M = \min_{m' \leq M^{1/4}} c_{m'}(1 - \frac{5}{m'^{3/4}})$. Denotes $m = |p|$. From the construction, for every $p \in A(\mathcal{D})$,

$$\nu(\mathcal{V}^{\text{lft}}(p)) + \nu(\mathcal{V}^{\text{rgt}}(p)) \geq \left(1 - \frac{5}{m^{3/4}}\right)\nu(\mathcal{V}(p)) \geq$$
$$\geq \left(1 - \frac{5}{m^{3/4}}\right) c_m \left(\tilde{v}(\mathcal{D}^R(p)) - \log m\right) \geq c_M \left(\tilde{v}(\mathcal{D}^R(p)) - \log m\right).$$

We conclude

$$\begin{aligned}
\nu(\mathcal{V}) \geq & \sum_{p \in A(\mathcal{D})} \tfrac{1}{4} \log |p| + \sum_{p \in A(\mathcal{D})} c_M \left[\tilde{v}(\mathcal{D}^R(p)) - \log |p|\right] + \\
& + \sum_{p \notin A(\mathcal{D})} \left[\tilde{v}(\mathcal{D}^R(p)) - \log |p|\right] \\
& \geq c_M \cdot \left[\sum_{i=1}^{s} \log(|p_i| + 1) + \sum_{i=1}^{s} \left[\tilde{v}(\mathcal{D}^R(p_i)) - \log(|p_i| + 1)\right]\right] \\
& = c_M \sum_{i=1}^{s} \tilde{v}(\mathcal{D}^R(p_i)) = c_M \left[\tilde{v}(\mathcal{D}^R(q)) - \log M\right]
\end{aligned}$$

$\square$

Lemma 5.6. *There exists a constant $c' > 0$ such that any proper path q with maximal degree 2 paths of lengths at least 15, has a recursive irreducible decomposition $\mathcal{D}^R$ along with a vine decomposition $\mathcal{V}$ of q such that $\nu(\mathcal{V}) \geq c' \cdot [\tilde{v}(\mathcal{D}^R) - \log |q|]$.*

Proof. We show the existence of recursive irreducible decomposition and "good" vine decomposition by induction on the structure of the recursive decomposition.

The base of the induction is degree 2 path. Degree-2 path has a unique recursive irreducible decomposition, namely all the vertices are in the separating set of the decomposition. The "good" vine decomposition is also trivial: all the vertices are in the backbone.

Otherwise, we use Lemma 5.5 to get an irreducible decomposition $\mathcal{D} = (S, \mathcal{P})$ and a vine selection $A(\mathcal{D})$. Inductively, Each $p \in \mathcal{P}$ has recursive irreducible decomposition $\mathcal{D}^R(p)$ along with "good" vine decomposition $\mathcal{V}(p)$. Hence, all the assumptions of Lemma 5.12 are satisfied, and we get recursive irreducible decomposition along with "good" vine decomposition for the given proper path. $\square$

Proof of Lemma 5.8

Lemma 5.8. *Under the conditions of Lemma 5.7 one of the following happens:*

- *Either there exists a vine decomposition $\mathcal{V}'$ with at least g vertices less than $\mathcal{V}$, such that* $\max\{\ell + t, \nu(\mathcal{V}')\} = \Omega(\ell + t + \tilde{v}(\mathcal{D}^R))$;
- *or there exists a vine V such that* $\max\{\ell + t, \log |V| - \log g\} = \Omega(\ell + t + \tilde{v}_g(\mathcal{D}^R))$.

Proof. Let $(B, \{q_1, \ldots, q_t\})$ denotes the complex of G^s at the end of sub-phase II. From Lemma 5.7 we have a vine decomposition $\mathcal{V}$ on at most $k + g$ vertices such that $\max\{\ell + t, \nu(\mathcal{V})\} = \Omega(\ell + t + \tilde{v}_g(\mathcal{D}^R))$.

First, if $\sum_i |q_i| \leq 8g$ or $\max_i |q_i| \leq g$, then $\tilde{v}_g(\mathcal{D}^R) \leq |\mathcal{P}_g(\mathcal{D}^R)|(\log 8g - \log g) = O(\ell + t)$, and the lemma trivially holds.

Otherwise, for the rest of the proof we assume that $\sum_i |q_i| > 8g$ and $\max_i |q_i| > g$. We also assume that $\nu(\mathcal{V}) \geq l + t$, otherwise the lemma holds trivially. Let q_1 denotes the biggest path in the complex, hence $|q_1| > g$. We denote by $B(\mathcal{V})$ the backbone of $\mathcal{V}$.

We further assume that $\max_{V \in \mathcal{V}} \log |V| \leq \nu(\mathcal{V})/24$. We will deal with the other case at the end of the proof.

We divide the vertices of $q_1 \cup \cdots \cup q_t$ between four sets X_1, X_2, X_3 and X_4 as follows: Let $v_1, v_2, \ldots, v_m$ be the vertices of $q_1 \cup q_2 \cup \cdots \cup q_t$ ordered as the order in $q_1, q_2, \ldots, q_t$. Let $t_0 = 1$

and $t_4 = m$. For $i \in \{1,2,3\}$, let t_i be the smallest index such that $t_i \geq t_{i-1} + g + 1$ and v_{t_i} is not on a vine. Now we assign $X_i = \{v_{t_{i-1}}, \ldots, v_{t_i}\}$. Note that the the vertices $v_{t_1}, v_{t_2}, v_{t_3}$ might appear in two consecutive sets.

All four sets have size at least $g+1$, unless there exists a vine V such that $|V| \geq g+1$. In this case we construct $\mathcal{V}'$ from $\mathcal{V}$ by throwing V. $\nu(\mathcal{V}') = \Omega(\nu(\mathcal{V}))$ because $\log|V| \leq \nu(\mathcal{V})/24$, and we are done.

We denote by $\nu(X_i)$ the sum of the values of the vines in X_i, $\sum_i \nu(X_i) = \nu(\mathcal{V})$. Assume that $\nu(X_{i_4}) = \max_{i \leq 4} \nu(X_i)$, so $\nu(X_{i_4}) \geq \nu(\mathcal{V})/4$.

For a vertex $u \in X_{i_4} \cap B(\mathcal{V})$, define C_u to be the vertices in $\mathcal{V}$ but *not* in X_{i_4}, that are connected to u via a simple path whose inner vertices are in $B(\mathcal{V}) \cap X_{i_4}$. Note that $C_u \neq \emptyset$. We say that u is locally connected to A if $C_u \subseteq A$. Let Y_{i_j} $(j \leq 3)$ be the set of vines in X_{i_4} that one of their endpoints is locally connected to X_{i_j}. Then $\nu(Y_{i_1}) + \nu(Y_{i_2}) + \nu(Y_{i_3}) \leq 2\nu(X_{i_4})$, since each vine is counted at most twice. Assume that Y_{i_1} has the minimal value of the three, so $\nu(Y_{i_1}) \leq \frac{2}{3}\nu(X_{i_4})$.

We throw from $\mathcal{V}$ the vertices as follows: (i) all the vertices in X_{i_1}, (ii) the vertices in the backbone of X_{i_4} that are locally connected to X_{i_1} and (iii) the vertices of the vines in Y_{i_1}. We add *all* the vertices in X_{i_2} and X_{i_3} to the backbone. The resulting is a "vine decomposition" $\tilde{\mathcal{V}}$ that the endpoints of its vines are all part of its backbone, and $\nu(\tilde{\mathcal{V}}) \geq \frac{1}{3}\nu(X_{i_4}) \geq \frac{1}{12}\nu(\mathcal{V})$.

The "backbone" of $\tilde{\mathcal{V}}$ consists of at most 3 connected components: (i) One that includes B (The backbone of the initial complex), (ii) one that includes X_{i_2}, and (iii) one that includes X_{i_3}. Actually there might be at most 2 connected components: Assume X_{i_2} is disconnected from B in the "backbone" of $\tilde{\mathcal{V}}$. In this case X_{i_2} is contained in one path q_{i_0} of the initial complex. In one side of X_{i_2} (in q_{i_0}), there are vertices from X_{i_1}, and in the other side, vertices from X_{i_4}. Now, either X_{i_3} contains an end of one of the original paths of the complex (in this case it is connected to B) or it adjacent to X_{i_2} in q_{i_0} (in this case it is connected to X_{i_2}). In both cases we have only two connected components.

If the backbone consists of only one connected component, we are done. Otherwise, denote the connected components $C \supseteq X_{i_2}$ and $D \supseteq B$. On q_{i_0}, $X_{i_4} \cap q_{i_0}$ separates X_{i_2} from $D \cap (q_{i_0} \cup \{q_{i_0}$'s neighbors in $B\})$.

Assume that C is on the left side of $X_{i_4} \cap q_{i_0}$ in q_{i_0}. Let $u \in C \cap q_{i_0}$ be the right most vertex of C in q_{i_0}. u is either in X_{i_4} or adjacent to X_{i_4}. Let $v \in D$ be the left most vertex of D (in q_{i_0} and its endpoints) that is to the right of u. The vertices in q_{i_0} between u and v are either a vine V of $\tilde{\mathcal{V}}$, or vertices that were thrown from $\tilde{\mathcal{V}}$. Either case we add them to the backbone of $\tilde{\mathcal{V}}$ and the resulting is the desired vine decomposition $\mathcal{V}'$, and

$$\nu(\mathcal{V}') \geq \nu(\tilde{\mathcal{V}}) - \log|V| \geq \tfrac{1}{2}\nu(\tilde{\mathcal{V}}) = \Omega(\nu(\mathcal{V})).$$

We are left with the case of a single vine V such that $\log|V| \geq \nu(\mathcal{V})/24$. In this case $|q_1| \geq |V|$, so $\nu(q_1) \geq \log|V| \geq \nu(\mathcal{V})/24 \geq d(l + t + \tilde{v}(\mathcal{D}^R))$, for some global constant $d > 0$.
If $|\mathcal{P}_g(\mathcal{D}^R)| \geq \frac{1}{d}$ then

$$\log|q_1| - \log g \geq d\big[\ell + t + \tilde{v}(\mathcal{D}^R) - \big|\mathcal{P}_g(\mathcal{D}^R)\big| \cdot \log g\big] = \Omega(\ell + t + \tilde{v}_g(\mathcal{D}^R))$$

If, on the other hand, $|\mathcal{P}_g(\mathcal{D}^R)| < \frac{1}{d}$, then

$$\tilde{v}_g(\mathcal{D}^R) = \sum_{p \in \mathcal{P}_g(\mathcal{D}^R)} (\log|p| - \log g) \leq \tfrac{1}{d}(\log|q_1| - \log g)$$

Either case, $\max\{l + t, \log|q_1| - \log g\} = \Omega(l + t + \tilde{v}_g(\mathcal{D}^R))$ $\square$

6 Refined Locality of Reference

Definition 6.1. An *extended access graph* on a set of pages $\mathcal{P}$ is a (finite) labeled undirected graph $G = (V, E, \ell)$, with a labeling function $\ell : V \to \mathcal{P}$ that labels vertices with pages.

A sequence of requests $\sigma = (\ell(v_i))_{i \geq 1}$, is a finite sequence of labels attached to vertices from G, such that either $v_{i+1} = v_i$, or $v_i v_{i+1} \in E$. A paging algorithm in this model should maintain the invariant, that following the ith request, $\ell(v_i)$ should be in some page slot. $r_{\mathrm{ON}}(G, k)$, $r(G, k)$, $r_{\mathsf{obl}}(G, k)$ are defined similarly for extended access graphs.

For a given extended access graph G, we define a parameter $\Delta(G)$ that indicates "how quickly" the locality of reference may change.

Definition 6.2.

$$\Delta(G) = \min\{|P| : P = \langle u = v_1, v_2, \ldots, v_l = v\rangle \text{ is a path in } G \text{ and } u \neq v \text{ and } \ell(u) = \ell(v)\}.$$

As convention, when G is an (non extended) access graph, we fix $\Delta(G) = \infty$.

We remark that since in practice, a paging algorithm would get only the pages that were requested and not the vertices, we assume that the starting vertex is known to the paging algorithm, and furthermore, $\Delta(G) > 2$. In this way, the paging algorithm can reconstructs sequence of vertices requested from the sequence of pages.

6.1 Truly Online Algorithms

Theorem 1 and Theorem 2 extend to the extended access graph model.

Theorem 3. *On extended access graph G with $\Delta(G) > (1 + \varepsilon)k$ and $\varepsilon > 0$,*

$$r_{\mathrm{DTO}}(G, k) \leq \max\{O(r(G, k)), 2/\varepsilon\}$$
$$r_{\mathrm{RTO}}(G, k) \leq \max\{O(r_{\mathsf{obl}}(G, k)), 2/\varepsilon\}.$$

In particular, for a fixed $\alpha > 1$, DTO *and* RTO *are strongly competitive for any G and k where $\Delta(G) > \alpha k$.*

Proof. Let g denotes the number of new pages in the phase. We consider two cases: If $g \geq \varepsilon k$ then DTO and RTO fault at most $k \leq \frac{2}{\varepsilon} \cdot \frac{\varepsilon k}{2} \leq \frac{2}{\varepsilon} \cdot \frac{g}{2}$ times in the phase.

If, on the other hand, $g < \varepsilon k$, then during the last two phases there were no requests to two different vertices with the same page, so the graph G_{III} is an actual sub-graph of G and therefore DTO faults at most $O(g \cdot r(G, k))$ times and RTO faults at most $O(g \cdot r_{\mathsf{obl}}(G, k))$ times, in the phase. □

When $\Delta(G)$ is slightly greater than k, DTO and RTO do not work well, as is seen in the following example.

Example 6.1. Consider the extended access graph G

$$\boxed{1} - \boxed{2} - \cdots - \boxed{k} - \boxed{k+1} - \boxed{1}$$

Here $\Delta(G) = k + 1$. OPT faults once per phase. The competitive ratio in this setting (both deterministic and randomized) is clearly a constant. But the request sequence that requests every page going from left to right and back again "confuses" DTO and RTO: In subphase I they would evict page 1 and $k + 1$. Subphase II would not be executed, and in subphase III they would fault $\Omega(\log k)$ times [expected].

6.2 Strongly competitive algorithm for paths with $k+1$ pages

In this section we present a non-truly online but strongly competitive algorithm for graphs that are simple paths on $k+1$ pages. This algorithm would also serve us in showing impossibility results concerning truly online algorithms.

Given a finite simple path $G = \langle v_1, v_2, \ldots, v_m\rangle$ with labels $\ell : \{v_i\}_i \to \mathcal{P}$, where $\mathcal{P} = \{p_1, \ldots, p_{k+1}\}$, and all the pages are present, *i.e.*, ℓ is surjective. Fix a starting vertex v_i. We define a partial order $\prec_i$ on $\mathcal{P}$, $p \prec_i q$ if any path from v_i to a vertex labeled by q must include a vertex labeled by p. It is easy to verify that $\prec_i$ is indeed a partial order. Let M_i be the set of maximal elements in $\prec_i$. Note that:

1. M_i includes at most two pages that appears only in one side of v_i.
2. Any request sequence that starts at v_i and access all the pages in M_i, must access all the pages in $\mathcal{P}$.
3. When restricting to pages in M_i that appear in both sides of v_i we define linear orders on them: $p \prec_i^L q$ in which the first appearance of p to the left of v_i is before any appearance of q to the left of v_i; and Similarly, $\prec_i^R$, on the right side of v_i. Note $\prec_i^R$ is the opposite order of $\prec_i^L$.

Lemma 6.1. $r_{\mathsf{obl}}(G,k) = \Omega(\max_i \log(|M_i|+1))$.

Proof. Let $i_0 = \arg\max_i |M_i|$, $|M_{i_0}| > 0$. Using Yao's Principle (cf. [BEY98, Ch. 8]) we construct a probability distribution on the request sequences by the following iterative process: The requests sequence is composed of periods, each one is composed of sub-periods. Let $\{N_j\}_j$ the $\prec_{i_0}$–maximal pages that were left unmarked before the jth sub-period, at the beginning of a period $N_1 = M_{i_0}$. In the jth sub-period the adversary begins from v_{i_0}. It chooses a page $p \in N_j$ such that p is smaller than $|N_j|/2+1$ pages of N_j under both $\prec_{i_0}^L$ and $\prec_{i_0}^R$ (its existence follows from simple counting argument). If p appears on both sides of v_{i_0} in G, it chooses uniformly at random the left or the right side, otherwise it chooses the side that p appears. The adversary then requests in that direction the vertices until arriving the first vertex v labeled with p, and then return to vertex v_{i_0}. In this point a new sub-period begins. Note that $|N_{j+1}| \geq |N_j|/2-1$. The adversary continues this way until $N_j = \emptyset$. At this stage all the pages were requested, and the period ends. The adversary then begin a new period.

During a period, an optimal off-line algorithm faults at most twice, because the request sequence consists of at most two phases.

Next we show that any online algorithm has at least $(\log|M_{i_0}|)/2$ expected number of faults of during a period.

To see that we argue that there are $\Omega(\log|M_{i_0}|)$ sub-periods in a period and an online algorithm pays at least half in each sub-period. There are at least $\Omega(\log|M_{i_0}|)$ iterations before is period N_j becomes empty set, because in each iteration the size of N_j is roughly halved. In all sub-periods in which p appears in both sides of v_{i_0} (which are all sub-periods except maybe two), the expected number of faults of the online algorithm is at least $1/2$. This is because $\prec_{i_0}$-maximal pages that appear on both sides of v_{i_0} must appear, at least on one side of v_{i_0},"after" the hole. Thus, the hole is requested with probability of at least $1/2$. ☐

We shall see now a deterministic online algorithm called MAXFAR which has $O(\log|M_i|)$ faults in a phase that begins in the vertex v_i. Its description is quite simple:

MAXFAR. MAXFAR is a marking algorithm. Assume the current phase began at v_i. On the jth fault in the phase, let $N_j \subset M_i$ be the set of unmarked $\prec_{i_0}$–maximal elements. Choose a page $p \in N_j$ such that is in the middle of the $\prec_i^L$ order (which is also in the middle of the $\prec_i^R$ order), and evict it.

It is easy to see that $\{N_j\}_j$ is a decreasing sequence of sets, and $|N_{j+1}| \leq |N_j|/2 + 1$. Thus after at most $j = O(\log |M_{i_0}|)$ faults, $N_j = \emptyset$, and at this stage the phase is over. We conclude that $r_{\text{MAXFAR}}(G, k) = O(\max_i \log |M_i|)$. Putting together,

Theorem 4. *For any extended access graph G which is a simple path on $k+1$ pages,*

$$r_{\text{MAXFAR}}(G, k) = O(r_{\mathsf{obl}}(G, k)).$$

6.3 Impossibility Results

In this section we show that any *truly online* algorithm can not be strongly competitive on extended access graphs, when $\Delta(G)$ is slightly less than the real memory size. Formally, we prove that

Theorem 5. *1. For any function $f(k) = \omega(\log(k))$, for any deterministic truly on-line paging algorithm A and for any k there exists an extended access graph G such that $\Delta(G) \geq k - f(k)$ and $r_A(G, k) \geq f(k)$ but $r(G, k) = O(\log k)$.*

2. For any function $f(k) = k^{1-o(1)}$, for any randomized truly on-line paging algorithm A, and for any k, there exists an extended access graph G such that $\Delta(G) \geq k - f(k)$ and $r_A(G, k) = \Omega(\log k)$, but $r_{\mathsf{obl}}(G, k) = O(\log(k/f(k)) + \log\log k) = o(\log k)$.

Proof. Fix a function f, and $k > 0$.

Consider the following *part* of an extended access graph labeled with $k+1$ pages:

$$1, 2, \ldots, k, k+1, x_1, x_2, \ldots, x_{f(k)}.$$

The pages $\{x_i\}$ are the set $\{1, \ldots, f(k)\}$ in some permutation that will be determined by the adversary. The request sequence σ will also be determined by the adversary.

To prove part (1), fix a deterministic algorithm A. We split the page request sequence σ into phases (as defined for marking algorithms) and will show that (except for maybe the 1st phase) all phases cost of A at least $f(k) - 1$ whereas they cost the adversary only 1. We will show that there exists an online algorithm (not truly online) that can obtain a competitive ratio of $\log k$ on this graph. This will proves claim (1).

Assume that in the previous phase the requested pages were $\{1, \ldots, k\}$. The page request sequence is generated as follows: first, the adversary requests the sequence $k+1, k, \ldots, f(k) + 2, f(k) + 1, f(k) + 2, \ldots, k, k+1$.

As A is deterministic, the adversary knows what page p has been evicted. The adversary sets $j = 0$. Define $x_0 = k + 1$.

If $p > f(k)$ then the adversary requests the pages $x_j, x_{j-1}, \ldots, x_0 = k+1, k, \ldots, p+1, p, p+1, p+2, \ldots, k, x_0 = k+1, x_1, \ldots, x_j$. This is a legitimate traversal on the extended access graph.

If $p \leq f(k)$ then: (i) If p has not been requested in this phase then the adversary sets $j = j+1$ and $x_j = p$. (ii) There must be some $i \leq j$ such that $x_i = p$ and the adversary generates the requests $x_j, x_j - 1, \ldots, x_i, x_{i+1}, \ldots, x_j$.

When $j = f(k)$ then the next phase begins. k different pages have been requested in this phase. The cost to A has been at least $f(k) - 1$ faults.

The above argument is for one phase, but we can continue this process for an additional c phases, for any c, simply by adding an additional $c \cdot f(k)$ vertices on the right hand side (right of $x_{f(k)}$ above).

Note that (i) $\Delta(G) \geq k - f(k)$ for any such graph; (ii) G is a simple path on $k+1$ pages, so from Theorem 4, MAXFAR has a competitive ratio of $O(\log k)$.

Next, we turn to prove the impossibility result for the randomized case. The adversary construct a graph similarly to the previous case, but now it does not know where the hole (page p) is. Instead it resorts to a random process as follows: It maintains one maximal unrequested sub-path $i_1, \ldots, i_2$ of the vertices in containing $1, \ldots, f(k)$. In each sub-phase it first requests all the already requested pages in the phase. Let $i_3 = (i_1+i_2)/2$ be the middle vertex in unrequested segment. The adversary then chooses uniformly at random one of the two:

- Requesting the pages $i_3, \ldots, i_2$ by setting $x_{j+1} = i_2$, $x_{j+2} = i_2 - 1, \ldots, x_{j+(i_2-i_3)+1} = i_3$, and then setting $j = j + (i_2 - i_3) + 1$.
- Requesting the pages $i_1, \ldots, i_3$ by setting $x_{j+1} = i_3$, $x_{j+2} = i_3 - 1, \ldots, x_{j+(i_3-i_1)+1} = i_1$, and then setting $j = j + (i_3 - i_1) + 1$.

Note that there are $\log f(k)$ sub-phases, and on each, the truly online algorithm faults with probability at least 1/2. Thus, the expected number of faults for the truly online algorithm is at least $(\log f(k))/2 = \Omega((1 - o(1)) \log k) = \Omega(\log k)$.

Like the deterministic case, this process can be continued for an additional c phases, for any c, simply by adding an additional vertices on the right hand side.

Note also that by the construction we maintained $\Delta(G) \geq k - f(k)$, and that G is a simple path on $k+1$ pages. Thus MAXFAR is applicable in this scenario. To bound M_i, fix a vertex v_i that in its left all the pages are represented. Note that each each sub-phase (after v_i was reached) contributes at most one maximal element for M_{i_0} and thus $\lceil \log f(k) \rceil$ in a phase. Note also that after $\lceil (k+1)/f(k) \rceil$ phases, all the pages are also represented to the right of v_i, and therefore $|M_i| \leq \lceil k/f(k) \rceil \lceil \log f(k) \rceil$. Thus MAXFAR faults at most $O(\log(k/f(k)) + \log \log f(k)) = O(o(1) \log k + \log \log k) = o(\log k)$. $\square$

7 Implementation

Storing G_0 requires only $O(k \log n)$ bits. We also need to keep track of:

1. The vertices requested so far in the current phase.
2. The current unevicted vertices of G_0.
3. G_0 for the next phase.

One way to construct G_0 for the *next* phase is to store a pointer from every page requested to the previously requested page. This pointer will only be updated once in the phase (the first time the page is accessed during the phase). The resulting data structure is a tree with pointers from the leaves upwards, and the root is the first page requested in phase.[2] At the end of a phase, the pointers are scanned and an image of G_0 is built in memory. This image contains vertices, pointers,

[2]An alternative approach is to store for each page a pointer to the *next* page, and update it each time the page is accessed. We get a tree rooted with the last requested page in the phase. This tree seems to capture better the locality of reference, as it stores edges resulting from more recent requests. However, it also has more pointer update operations.

and the virtual addresses associated with the vertices. In total (hardware registers and memory) the memory requirement is $O(k \log n)$ bits.

As noted in the introduction this can be implemented efficiently in hardware. The $\log n$ bit virtual address is translated to a $\log k$ page slot address (by the virtual address translation mechanism). The spanning tree is built on the $\log k$ bit address space of page slots but the virtual addresses ($\log n$ bits) are stored in the vertices. Hence, the hardware requirement for implementing DTO and RTO is one $\log k$ bits pointer and one extra bit (marking bit) associated with every page slot. The extra processing associated with interpreting an address is one ($\log k$ bits) address comparison, one bit comparison, and (possibly) setting one $\log k$ bits pointer.

DTO and RTO, as presented here, used a spanning tree of G_P as their reference access graph. It is easy to see that instead, we can use any connected sub-graph of G_P. We can even use G_P itself and store it using only $O(k \log n)$ bits, instead of the $O(k^2 \log n)$ in the naive implementation. The key observation is that there is no need to actually store edges that both their endpoints have degree ≥ 3. Thus, when we add a new edge we increase the degree of its endpoints, and "forget" all the edges that both their endpoints have degree ≥ 3. In this way we only need to store at most $2k$ edges.

It would be of theoretical interest to reduce the memory requirements even further, to be independent of n. Our goal is to reduce the factor of $\log n$ in the storage requirement to $\log k$. Of course, the address translation table must still use $\Omega(k \log n)$ bits. But when we restrict ourselves to the paging eviction strategy, and assuming we are also told whether a page request is a hit or a fault, we can allow some small error, if its total effect on the number of page faults is insignificant.

Consider a universal set of hash functions $h : \mathbb{N} \to \{0, \ldots, m-1\}$, *e.g.,* $h(x) = ax + b \pmod{m}$, where m is prime and a and b are uniformly and independently sampled from $\mathbb{Z}_m$. The paging algorithm would work with the hash value of the page ($h(p)$) instead of the virtual address (p).

In this case the paging algorithm might errs and identify two different pages as the same. In the worst case, it would confuse our algorithms in the current and the next phase (because G_0 for the next phase is built incorrectly). Nonetheless, because the algorithms are marking and has no longer memory, the damage is restricted to these two phases, and so it would add at most $2k$ page faults.

The probability of such a bad thing to happen is bounded from above by the following simple argument: Look at at most $\binom{2k}{2}$ pairs of different pages requested during the last two phases. The probability of pair of different pages to collide is $1/m$, therefore we can bound the probability of an error during a phase by $\binom{2k}{2}\frac{1}{m} = O\left(\frac{k^2}{m}\right)$. Choosing $m = \Theta(k^3)$ insures that the expected number of added page faults would be only $O(k\frac{k^2}{m}) = O(1)$ per phase.

Hence, a data structure with $O(k \log m) = O(k \log k)$ bits gives a randomized algorithm with the same performance guarantees, up to a constant factor.

8 Concluding Remarks

In this paper we studied the access graph model for locality of reference. We have shown a somewhat surprising result: It is possible to be both truly online and still strongly competitive in the access graph model. The resulting algorithms seems practical, and can be proven to work well even when the locality of reference changes.

We remark that if one uses asymptotic competitive ratio instead of strcit competitive ratio there is a straightforward truly online algorithm as follows: Take any marking strongly competitive algorithm and execute it on "the observed access graph" so far *i.e.* the graph that has all the edges used so far in the request sequence. On phases in which the access graph did not changed these algorithms faults at most g times the competitive ratio of the observed graph. On phases in which the access graph do change they fault at most k times, and there are at most n^2 such phases. Therefore, using $C = n^2 k$ as the constant additive term, we get "strongly competitive algorithm". This solution, however, is not strictly competitive, has a large additive term, requires $\Omega(n^2)$ memory, and does not extend to the extended access graph model.

The following points are not completely resolved.

1. The proof of Lemma 4.2 is very lengthy. Is there a simpler and/or shorter proof?
2. In a preliminary version of this work [FM97] we claimed that both DTO and RTO can be patched so as to be strongly competitive for any G and k as long as $\Delta(G) > k$. The patch we devised turned out to be erroneous. Still, we conjecture that strongly competitive truly online algorithm are possible if $\Delta(G) > k$.
3. Finding an uniform and strongly competitive algorithm for the extended access graph model.
4. Is there strongly competitive algorithm for the *directed access graph model*? Or is it computationally hard problem?

Acknowledgments. We would like to thank Yair Bartal and Gideon Stupp for many helpful discussions.

References

[BDBK+94] Shai Ben-David, Allan Borodin, Richard Karp, Gábar Tardos, and Avi Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, January 1994.

[Bel66] L.A. Belady. A study of replacement algorithms for virtual storage computers. *IBM Systems Journal*, 5:78–101, 1966.

[BEY98] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, Cambridge, UK, 1998.

[BIRS95] Allan Borodin, Sandy Irani, Prabhakar Raghavan, and Baruch Schieber. Competitive paging with locality of reference. *J. Comp. Syst. Sci.*, 50(2):244–258, April 1995.

[CN99] Marek Chrobak and John Noga. LRU is better than FIFO. *Algorithmica*, 23:180–185, 1999.

[FK95] A. Fiat and A. Karlin. Randomized and multipointer paging with locality of reference. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pages 626–634, 1995.

[FKL+91] Amos Fiat, Richard Karp, M. Luby, L. A. McGeoch, Daniel D. Sleator, and N. E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12:685–699, 1991.

[FM97] Amos Fiat and Manor Mendel. Truly online paging with locality of reference (extended abstract). In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, pages 326–335, 1997.

[FR97] Amos Fiat and Ziv Rosen. Experimental studies of access graph based heuristics: Beating the lru standard? In *8th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 63–72, 1997.

[IKP96] S. Irani, A. Karlin, and S. Phillips. Strongly competitive algorithms for paging with locality of reference. *SIAM J. Comput.*, 25(3):477–497, June 1996.

[KPR92] Anna. R. Karlin, Steven J. Phillips, and Prabhakar Raghavan. Markov paging (extended abstract). In *Proceedings of the 33nd Annual Symposium on Foundations of Computer Science*, pages 208–217, 1992.

[LPR94] Carsten Lund, Steven Phillips, and Nick Reingold. IP over connection-oriented networks and distributional paging. In *35th Annual Symposium on Foundations of Computer Science*, pages 424–434, 20–22 November 1994.

[ST85] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communication of the ACM*, 28:202–208, 1985.