

Almost-Linear-Time Maximum-Cardinality Matching in General Graphs

OpenAI

September 24, 2026

Abstract

We prove that maximum-cardinality matching in a simple undirected graph with n vertices and m edges can be found by one uniform randomized algorithm in $(n + m)^{1+o(1)}$ time. The bound holds on every computation path in a logarithmic-word model, and the algorithm returns an explicit maximum matching with probability at least $2/3$. The algorithm reduces the problem, with witnesses preserved, to perfect matching in graphs of maximum degree three. There it maintains a matching through factor-critical odd blocks. A Hall-type surplus of these blocks is certified by fractional allocations, at several scales, of vertices outside the blocks to adjacent blocks, and temporary shortfalls of these allocations are repaired locally. The algorithm makes progress on a logarithmic objective by steps along approximate minimum-ratio circulations. These circulations come from a fully dynamic cycle data structure, which is constructed here using a static routing sparsifier that preserves resource loads. A dovetailed schedule combines all parameter choices into one program. A reduction gives the same time and probability guarantees for deciding whether a simple host graph has a spanning subgraph with prescribed vertex degrees $f(v)$, where $0 \leq f(v) \leq \deg(v)$, and for finding one when it exists.

Mathematics Subject Classification (2020). Primary 05C85; Secondary 05C70, 68W20, 68Q25.

Keywords. maximum-cardinality matching; general graphs; almost-linear time; f -factors; factor-critical graphs; minimum-ratio cycles; dynamic graph algorithms; randomized algorithms.

Contents

1	Introduction	4
1.1	Statement and computational model	4
1.2	Context and prior work	5
1.3	Overview of the proof	6
1.4	Organization	8
2	Conventions and static tools	9
2.1	Representations, fixed-parameter bounds and failure conventions	9
2.2	Exact integral flow	10
2.3	A polynomial-time fallback	11

3	Matching states and structural operations	11
3.1	The phase state and its ledgers	12
3.2	The top objective and the step primitive	14
3.3	The main actions and the ledger edits	15
3.4	Lower bounds on the top-ledger coordinates	16
3.5	Reset schedule and the exchange problem	18
3.6	Tight sets and factor-critical contraction	20
3.7	Randomized search on an edge between two bins	21
3.8	Matching materialization	24
4	Local projection and the perfect-matching bound	25
4.1	The local projection lemma	25
4.2	Local exploration and bounds on the exchange maximum	26
4.3	The threshold test	27
4.4	Recovering the exact flow and the full residual cut	32
4.5	Debt, structural work, and progress	33
4.6	Implementation and total phase cost	35
4.7	Absence of guard failures and the perfect-matching theorem	38
5	Ascent steps along approximate minimum-ratio circulations	41
5.1	Setting and the step primitive	41
5.2	A dynamic minimum-ratio cycle data structure with forest paths	43
5.3	Degree reduction and integer prices	44
5.4	The acceptance test and the step	45
5.5	Oriented forest operations	46
5.6	Preserving exact values under forest changes	47
5.7	Drift detection and the number of refreshes	48
5.8	Precision, running time, and the proof of Lemma 5.1	50
5.9	Amortized analysis of the augmented link-cut forest	51
6	A resource-preserving static routing sparsifier	52
6.1	Stages, pieces, and normalized sampling	54
6.2	Median cuts, integral matchings, and an expanding overlay	56
6.3	Recursive routing on the overlay	58
6.4	Filtering real embeddings and verifying a stage	62
6.5	Failure probability and finite computation	63
7	Approximation ratio of the cycle data structure	65
7.1	Parameters, certified scores, and the bottom level	65
7.2	Batched updates and the decomposition target	67
7.3	Verified covers and a finite family of forests	68
7.4	Route distributions and the decomposition of a circulation	70
7.5	Local correction cycles	73
7.6	The child graph	75
7.7	The one-level reduction and the recursion	76

8	Maintaining and exposing the cycle data structure	78
8.1	Maintaining endpoint classes	79
8.2	Maintaining the logical paths	80
8.3	Small changes to the compressed child graph	84
8.4	Prebuilt path copies and the lifting of child forest paths	85
8.5	Explicit recourse and persistent edge lifetimes	88
8.6	Total work, finite arithmetic, and reliability	90
9	Explicit output and one uniform program	93
9.1	Witness-preserving reductions	93
9.2	Threshold search and its error probability	95
9.3	Running time of a single branch	96
9.4	An instruction-level schedule	97
9.5	Proof of Theorem 1.1	99
10	Prescribed-degree factors	100
	References	103

1 Introduction

A matching in a graph is a set of edges with pairwise disjoint endpoints. The maximum-cardinality matching problem asks for such a set with the largest possible number of edges. We prove an almost-linear bound in the adjacency-list input size for general graphs, with explicit output and a single uniform algorithm. The running-time bound holds on every computation path, including every unsuccessful random computation, and it counts all finite-precision work and all input and output.

1.1 Statement and computational model

The input is a simple undirected unweighted graph $G = ([n], E)$, where $n \geq 1$ and $m = |E|$. It is given by an array of adjacency-list boundaries and two neighbor occurrences per edge. Labels and indices use $O(\log(n + 2))$ bits. The lists may have any valid order. Isolated vertices count toward n , and the graph need not be connected. Write $p = n + m$, which is within a constant factor of the number of input word records, and write $\nu(G)$ for the maximum matching size.

We use a classical randomized word-RAM whose word length is

$$w = \lceil c_0 \log_2(n + 2) \rceil,$$

for a constant $c_0 \geq 2$ chosen with the algorithm. Unit-cost instructions are word reads and writes, comparison, addition, subtraction, multiplication modulo 2^w , integer quotient and remainder with nonzero divisor, Boolean operations, and shifts by fewer than w positions. Each independent fair random bit costs one instruction. Arithmetic on longer integers is carried out by multiword routines, whose instructions are counted. Allocation, initialization, data movement, preprocessing, and output writing are included. The algorithm is uniform: it uses no oracle and no advice that depends on the input size. Since $n \leq p \leq n^2$ for a simple graph, the word length is also $\Theta(\log(p + 2))$.

Theorem 1.1 (Maximum matching in almost-linear input-size time). *There exist a single uniform randomized word-RAM algorithm, constants $c_0 \geq 2$ and $C \geq 1$, and a nonincreasing function $\eta : \mathbb{N}_{\geq 1} \rightarrow \mathbb{R}_{\geq 0}$ with $\eta(q) \rightarrow 0$ as $q \rightarrow \infty$ such that the following holds. For every input graph $G = ([n], E)$ with $m = |E|$ and $p = n + m$, the algorithm halts on every computation path in at most*

$$Cp^{1+\eta(p)}$$

instructions. With probability at least $2/3$ it outputs an explicit list of edges of a matching of size $\nu(G)$.

Thus the time bound holds for every outcome of the random bits; only the correctness of the output is probabilistic. The function η is part of the theorem: one program attains the exponent $1 + \eta(p)$ for all inputs.

For every fixed integer $\kappa \geq 1$, Theorem 1.1 gives $C_\kappa n^{1+\eta(n)}$ instructions on graphs with $m \leq \kappa n$, for some constant C_κ . The same algorithm and the same function η work for all κ , which need not be supplied as an input. At the dense extreme $m = \Theta(n^2)$, the bound is $n^{2+o(1)}$. Both specializations are derived at the end of the proof of Theorem 1.1 in Section 9.

The theorem applies to every input graph. Section 9 reduces each cardinality query to a perfect-matching instance of maximum degree three by reductions that preserve explicit witnesses (Lemmas 9.2 and 9.3), and it constructs the single finite program and the function η of Theorem 1.1.

A reduction also gives an almost-linear bound for prescribed vertex degrees. Given an integer $f(v)$ with $0 \leq f(v) \leq \deg_G(v)$ for every vertex v , an f -factor is a spanning subgraph $([n], F)$ with

$\deg_F(v) = f(v)$ for every v . Corollary 10.1 gives one uniform randomized algorithm that decides whether an f -factor exists and returns its edge list when it does, with success probability at least $2/3$ and $(n + m)^{1+o(1)}$ instructions on every computation path. The degree prescription is an array of one-word integers. Section 10 gives the witness-preserving reduction, which increases the number of edge incidences by at most a logarithmic factor.

1.2 Context and prior work

The structure of matchings has long connected combinatorial existence criteria with efficient construction. Hall’s theorem characterizes when a family of sets has distinct representatives [16], and Tutte’s perfect-matching theorem identifies the obstruction caused by odd components [34]. Berge’s augmenting-path criterion turns maximum cardinality into a question about alternating paths [2]. Edmonds made this approach algorithmic in general graphs through blossom contraction and expansion [7]. His matching-polytope theorem gives a complementary explanation of the odd-set constraints [8]: vertex-degree inequalities alone allow, for example, value $1/2$ on each edge of a triangle. These odd-set constraints reappear below as the obstacle for flow-based methods.

Tutte’s short proof of the factor theorem reduces prescribed-degree f -factors to perfect matchings [35]. The reduction in Section 10 follows this incidence-port approach, replacing the local complete connections to auxiliary vertices by the rearrangeable networks used in Section 9. This controls the size of the perfect-matching instance even when a host vertex has high degree. There are also direct algorithms: Gabow finds a maximum-cardinality edge set subject to $\deg_F(v) \leq f(v)$ in $O(n^{2/3}m)$ time on simple graphs [11, Theorem 4.1]. After linear-time input processing, including removal of zero-demand vertices, this gives an $O(n + n^{2/3}m)$ factor test: an f -factor exists exactly when the returned set has $2|F| = \sum_v f(v)$.

Combinatorial algorithms for maximum matching grew out of batching shortest augmenting paths. This led to the bipartite algorithms of Hopcroft and Karp [18] and, independently, Karzanov [19], and to the Micali–Vazirani algorithm for general graphs [26, 39]. The latter has running time $O(m\sqrt{n})$ on a RAM; reading the explicit input and removing isolates gives the convenient universal bound $O(n + m\sqrt{n})$. This is $O_\kappa(n^{3/2})$ when $m \leq \kappa n$. Gabow and Tarjan developed a different route through weighted scaling [12], and Gabow later gave a direct weighted-search formulation at the same cardinality bound [10]. On very dense graphs, Goldberg and Karzanov improve the combinatorial bound to $O(n + m\sqrt{n} \log(n^2/m)/\log n)$ by combining skew-symmetric flow with graph compression [15, Theorem 10.3]. The weighted problem has its own sequence of improvements: the scaling algorithm of Duan, Pettie, and Su finds an exact maximum-weight matching in $O(m\sqrt{n} \log(nW))$ time for integer weights of magnitude at most W [6], where the logarithm is taken to be at least a constant.

Algebraic methods offer another important approach. Random evaluation of the Tutte matrix connects matching to linear algebra [24]. Mucha and Sankowski, and subsequently Harvey, obtained randomized algorithms that construct maximum matchings at the matrix-multiplication time scale, conventionally written $O(n^\omega)$ field operations [17, 28]. These algorithms already recover explicit witnesses and can work over finite fields of polynomial size. Their matrix-multiplication bound is most competitive on dense graphs.

For bipartite matching, the unit-capacity flow formulation permits continuous optimization to accelerate exact search. Mądry’s electrical-flow interior-point method [25] and the moderately dense algorithm of van den Brand et al. [36] were intermediate advances. The almost-linear exact integral-flow algorithms of Chen et al. [3] and van den Brand et al. [37] then yield almost-linear exact bipartite matching. The use of approximate minimum-ratio circulations as optimization directions, that is, of circulations whose gain in the objective is large relative to their weighted length, has

its modern antecedent in this almost-linear flow framework [3, 37]. For general graphs a flow formulation alone does not suffice, because of the odd-set constraints recalled above; a flow-based method must also control odd vertex sets. Supplying this control while keeping the work almost linear is the obstacle addressed here (Section 1.3). The deterministic flow theorem is used here through the static exact-flow statement of Lemma 2.1.

The proof combines several established ideas with tools adapted to the odd-set control. Its constructive factor-critical blocks belong to the contraction-and-expansion tradition, while Hall’s theorem underlies the reassignment of their exposed vertices. The allocations at several scales that certify the Hall surplus, and the bounds on their temporary shortfalls, are developed in Sections 3 and 4. The routing sparsifier of Section 6 uses the cut-matching structure of Khandekar, Rao, and Vazirani [20], together with concentration and conductance estimates [30, 33]. Its recursive virtual graphs follow the hierarchical random-walk embeddings of Ghaffari, Kuhn, and Su [14]. Here the sampled support and the substituted routes must also preserve supplied resource loads.

For dynamic graphs, Chen et al. prove a deterministic *fully dynamic* approximate minimum-ratio-cycle theorem with flat forest representations [4]. Our routing hierarchy is related to the origin-aware construction of Rozhoň et al. [29]; the finite forest family and associated cycle extraction adapt the construction in [4, Section 4.5]. The preallocated forest copies used to lift recursive paths follow Kyng, Meierhans, and Probst Gutenberg [21]. The recent distance-oracle approach of Kyng et al. [22] provides another development of this method. Sections 7 and 8 construct in full the cycle data structure used here. Besides the quality of its answers, the step primitive of Section 5 (described in Section 1.3) needs four further properties from it, stated in Lemma 5.2: guarantees against adaptively chosen updates, explicitly reported changes of representation, edge copies that persist over stated lifetimes, and bounds on the length of the returned paths counted with repetition. Sections 5 and 8 show how the values accumulated on retained edge copies survive these changes of representation.

1.3 Overview of the proof

Reduction to a degree-three subproblem. The central subproblem is to find a perfect matching in a loopless graph of maximum degree three, called *subcubic*. Section 9 reduces each query “does G have a matching of size k ?” to this subproblem while retaining the original edge identifiers, so that every successful answer can be decoded into a matching of the input graph; a binary search over k then finds a maximum matching. The perfect-matching procedure receives a power of two S bounding the number of vertices of the reduced graph, and its time bounds are stated in terms of S .

The matching state. The procedure represents its current matching through a partition of the vertices. Its odd blocks, called *bins*, each carry a constructive witness that they are factor-critical: for every vertex of the bin, the other vertices of the bin can be matched internally. Some vertices outside the bins, called *rows*, are assigned along edges to distinct bins. Each assigned bin matches its exposed vertex to its row, and a bin without a row contributes one unmatched vertex. The remaining vertices are stored as matched pairs, apart from unmatched vertices set aside during repairs. Three structural operations change the state (Section 3): a *gain* matches two bins without rows along an edge joining them and increases the matching size by one; a contraction merges a set of bins and the rows enclosed by it into one bin; an expansion turns a matched pair adjacent to a bin into a row and a new single-vertex bin. The procedure works in *phases*; a phase ends after a prescribed number of gains, roughly proportional to the *deficit*, the number of further edges needed for a perfect matching at the start of the phase.

The obstacle. The difficulty is to repair this representation after structural changes without repeatedly scanning all its bins. The procedure relies on a strict Hall-type surplus: every nonempty

set X of bins is larger than the set of rows all of whose neighboring bins lie in X . This surplus underlies an alternative: either the row assignment can be rearranged so that two adjacent bins have no row, which permits a gain, or a tight set containing them can be contracted. It must be maintained through every structural operation.

The central mechanism: ledgers with debt. The Hall surplus is certified by fractional allocations of rows to bins. A *ledger* allocates the same amount from each row, the *row allocation*, among its neighboring bins; the *residual* of a bin is the difference between that amount and the bin's allocated load. There is one ledger at each of several levels, from the top level to the bottom level; the number of levels below the top is the main parameter a . Each requires every residual to be at least a multiple of the bin's weight, which is half its size rounded up; the multiplier decreases by one from each level to the next toward the bottom, and a nonnegative *debt* records by how much this floor may temporarily fail. The bottom ledger has no debt whenever the main loop selects its next action, and it then certifies the strict Hall surplus. The top ledger carries a logarithmic objective, which admits an improving circulation direction whenever the graph has a perfect matching and no structural operation applies. The intermediate ledgers make repairs local: each is restored on its own schedule.

A repair, called a *projection*, restores the floor of one ledger by an exact integral flow problem on row-bin incidences. Either the flow restores the floor, or it identifies a set of bins whose removal makes the floor hold. Because adjacent floors differ by one unit of weight, a removal decreases the debt of every ledger above it that is kept through the removal by at least the removed weight. The flow problem is solved locally: exploration starts at bins whose debt is substantial relative to their size and extends to a further bin only when enough flow arrives there, so the debt also bounds the part of the graph that is examined.

Pricing steps and the cycle data structure. When no structural operation applies, the procedure optimizes the top objective. Both this optimization and the local flow tests use the same *step primitive* (Lemma 5.1). A *pricing call* searches for an approximate minimum-ratio circulation, that is, a circulation direction with sufficient objective increase relative to its weighted length; when it accepts such a direction, the call takes a small exact dyadic step, called a *pricing step*. A pricing call on the top objective may instead return the outcome LOW. If the graph has a perfect matching and no structural operation applies, an improving direction exists, and then the call takes a pricing step, reports a detected failure (an exhausted randomized search, Section 2), or suspends; so LOW implies that the graph has no perfect matching. Directions are represented by edges and by paths in a changing forest, so a long path need not be expanded at every step.

The directions come from a fully dynamic cycle data structure on subcubic graphs (Lemma 5.2), constructed in full in Sections 6–8. A static routing sparsifier provides short routes whose total use of specified resources is bounded. The data structure is recursive: during an interval of updates, forests built at the start of the interval decompose every current circulation into local correction cycles and a circulation in one smaller graph, and prebuilt copies of paths lift the forest of the smaller graph back to the current graph while keeping the values maintained by the step primitive.

Why the total time is almost linear. Theorem 4.8, the main fixed-parameter result, combines four estimates. First, each completed phase that starts with a large deficit reduces it by a constant factor, so there are logarithmically many phases. Second, within a phase the row allocation and the number of gains are balanced so that the debt created by gains, and with it the total removed weight and the total size of the structural operations, is linear in the number of vertices. Third, levels closer to the top are restored less often, and at each restoration the debt one level up is bounded in proportion to the interval between restorations at that level; hence each level contributes almost-linear total work. Fourth, each pricing step increases the logarithmic objective by at least $S^{-O(1/a)}$, up to a constant factor, while the objective varies over a range of almost-linear size apart

from the losses at structural operations; this bounds the number of pricing steps. All the overhead factors are of the form $S^{O(1/a)+o_a(1)}$, where the coefficient hidden in $O(1/a)$ is absolute, so the subcubic test takes time $S^{1+O(1/a)+o_a(1)}$.

One uniform program. For each sufficiently large fixed a , these ingredients give a subcubic perfect-matching test whose time bound holds on every computation path. A failed deterministic check suspends the run for the current parameters without an answer, whereas an exhausted randomized search gives a possibly erroneous negative answer (Section 2); suspension cannot occur once the input is large enough for the fixed parameters. Section 9 runs the tests for an increasing sequence of parameter choices, together with a polynomial fallback, by weighted dovetailing as in universal search. The error probabilities sum to less than $1/3$, and the schedule is as fast as each parameter choice up to a factor depending only on that choice and lower-order overhead. This gives the single function η of Theorem 1.1.

Significance. The combinatorial bounds recalled above are of order $m\sqrt{n}$ up to logarithmic factors, and the algebraic bounds are of order n^ω . Theorem 1.1 gives an almost-linear bound in the explicit input size throughout the range of densities, including $n^{2+o(1)}$ at the dense extreme. It extends to general graphs the almost-linear time that flow methods give for bipartite matching, by combining approximate minimum-ratio circulations with the control of odd sets through bins and ledgers. Through the reduction of Section 10, the same bound holds for finding f -factors (Corollary 10.1).

1.4 Organization

The sections follow the dependencies of the proof from the matching procedure down to its data structures, and then assemble the main theorem.

1. *Conventions and static tools* (Section 2). Representation and cost conventions, the parameters S, a, σ , and the distinction between suspension and detected failure; exact static integral flow (Lemma 2.1) and the polynomial perfect-matching fallback (Lemma 2.4).
2. *Matching states and structural operations* (Section 3). Bins, rows and ledgers (Definition 3.1), the Hall certificate (Lemma 3.2), the structural operations, and the exact projection (Lemma 3.6). The step primitive of Lemma 5.1 is used here through its stated properties.
3. *Local projection and the perfect-matching bound* (Section 4). Local implementation of the projection (Lemma 4.1), the accounting of debt and structural work (Lemma 4.5), progress (Lemma 4.6), and the main fixed-parameter result, Theorem 4.8: for each sufficiently large fixed a , a subcubic perfect-matching test with time $S^{1+O(1/a)+o_a(1)}$ on every path. Theorem 4.8 is proved assuming Lemma 5.1.
4. *Ascent steps along approximate minimum-ratio circulations* (Section 5). Lemma 5.1 is proved from the cycle data structure of Lemma 5.2.
5. *A resource-preserving static routing sparsifier* (Section 6). Static routing with resource bounds, Lemma 6.1.
6. *The cycle data structure* (Sections 7 and 8). Section 7 proves the quality of the answers through the one-level reduction (Lemma 7.8) and its iteration through the recursion (Corollary 7.10); Section 8 maintains the smaller graphs and the public forest, through which paths of the smaller graphs are lifted, and bounds the work (Lemma 8.9), completing the proof of Lemma 5.2.
7. *Explicit output and one uniform program* (Section 9). Theorem 4.8 is applied through the following chain: the witness-preserving reductions (Lemmas 9.1, 9.2 and 9.3) and the binary search over the cardinality threshold; the absence of suspension and the running time of each fixed

branch on every computation path (Lemma 9.4); the schedule, realized by one finite program with polynomial halting time (Lemma 9.5); and the construction of η in the proof of Theorem 1.1 (Section 9.5).

8. *Prescribed-degree factors* (Section 10). The reduction of Lemma 10.2 and Corollary 10.1.

2 Conventions and static tools

2.1 Representations, fixed-parameter bounds and failure conventions

Auxiliary graphs may have parallel edges or arcs, always distinguished by identifiers. Fix a reference orientation on every variable edge. Traversing an edge backward negates its coefficient. A signed circulation has zero divergence at every vertex; its length norm for positive lengths ℓ is

$$\|\ell f\|_1 = \sum_e \ell_e |f_e|.$$

This norm differs from the *raw length* of a supplied traversal list: the latter counts every traversal with its multiplicity, before cancellation. In the cycle data structure of Lemma 5.2, a forest edge maps either to an input edge, with its endpoint correspondence specified, or to a null edge whose two endpoints have the same image. A null traversal has zero projected length, although a reduction may assign it an additional positive bookkeeping length.

The cost of data structures is measured as follows. All representation sizes include vertices or slots, individual edges, terminal incidences, and explicitly stored lists. Ordinary adjacency lists, heaps, balanced dictionaries, and path-copy persistent maps incur logarithmic overhead per record operation. After an implicit path update, every coordinate that an argument reads is obtained by a counted operation. Arithmetic and key-length costs are included separately when they exceed one word.

The matching procedure uses a size bound $S = 2^s$, an integer parameter $a \geq 1000$, and a reliability parameter σ . In the final schedule $\sigma \leq a + 1$. A bound of the form

$$S^{O(1/a)+o_a(1)}$$

has an absolute coefficient in $O(1/a)$; for each fixed a, σ , the second exponent tends to zero as $S \rightarrow \infty$. Constants and powers of logarithms may depend on these fixed parameters. All smaller local routines measure their permitted logarithmic factors against the indicated size bound. Statements “for sufficiently large S ” are made with a and σ fixed: they hold for all $S \geq S_0$, with a threshold S_0 that may depend on a and σ . The parameters are not fixed uniformly while taking $a \rightarrow \infty$.

Two kinds of failure are distinguished. An implementation may check a deterministic size or precision prerequisite, or a cap on a number of operations; such a deterministic check is called a *guard*. If a guard fails, the implementation *suspends* the entire parameter branch, that is, the run of the procedure for the current choice of parameters (see (9.3)), without an answer. In contrast, exhaustion of a capped randomized routine is a *detected failure*; the calling perfect-matching test returns a negative answer immediately. The fixed-parameter analysis proves that deterministic suspensions cannot occur on any path once the input is sufficiently large for the fixed parameters of the branch. A *guarded prefix* is a prefix of a computation path that ends before the first deterministic interruption of the path, that is, before the first failed guard, which suspends the branch. These conventions are used in Theorem 4.8 and the schedule in Section 9.

2.2 Exact integral flow

Lemma 2.1 (Static integral flow). *For an explicitly represented directed network of total size at most P , with integral capacities, demands and costs bounded in magnitude by a fixed power of P , exact integral minimum-cost flow can be computed uniformly and deterministically in $P^{1+o(1)}$ time. Its finite-precision operations can be simulated with polynomially logarithmic overhead on logarithmic-bit words. In particular, integral source–sink maximum flow with nonnegative upper capacities and zero lower capacities returns an explicit integral arc vector and a residual-reachability minimum-cut certificate within that bound.*

Proof. The deterministic minimum-cost-flow theorem of van den Brand et al. [37, Theorem 1.1], with its finite fixed-point model and deterministic integral rounding [37, Section 2 and Lemma 4.1], provides an algorithm for exact integral minimum-cost flow on such networks. The stated logarithmic dependence on integral magnitudes and polynomial logarithmic word lengths is absorbed in $P^{o(1)}$. A fixed-point word is stored as an array of logarithmic-bit words. Longer virtual addresses can be keys in a dictionary of touched cells; no unused address space is initialized.

Maximum flow is obtained by adding a bounded sink-to-source return arc and minimizing its negative flow. Its capacity can be the sum of the original capacities. Feasibility and absence of a residual source–sink path certify maximality. Residual reachability supplies the cut. $\square$

Every call of Lemma 2.1 below is a maximum-flow computation with nonnegative upper capacities and zero lower capacities. Integral lower bounds are used in feasibility arguments, through the classical cut criteria for circulation feasibility [9, 13]. Lemma 2.2 gives the precision margin used in the threshold tests of Section 4.

Lemma 2.2 (Integral cuts from a small total violation). *Let every lower and upper arc bound be integral, with lower bound no larger than upper bound. If an exactly conserved real arc vector has total bound violation less than one, the bounds admit an exactly feasible integral circulation.*

Proof. For a vertex set X , feasibility requires the sum of lower bounds on arcs leaving X to be at most the sum of upper bounds on arcs entering X . For integral bounds this circulation cut criterion is also sufficient: subtract the lower bounds, introduce a super-source and super-sink for the resulting vertex imbalances, and apply the integral maximum-flow/minimum-cut theorem; saturating the imbalance arcs then gives an integral circulation within the bounds. It therefore suffices to verify the criterion. Conservation of the real vector implies that any failure of the inequality is at most its total bound violation. The two bound sums are integers, so a positive discrepancy would be at least one, a contradiction. Hence the criterion holds, and the bounds admit an exactly feasible integral circulation. $\square$

Lemma 2.3 handles capacities of arbitrary bit length with one fixed implementation of Lemma 2.1. In the fixed-parameter applications, the bit length b and all key lengths contribute only factors that are powers of the logarithm of the size bound; the fixed-parameter bounds above include such factors.

Lemma 2.3 (Capacity scaling for maximum flow). *Maximum flow with nonnegative integral upper capacities of at most b bits can be computed by b calls to one fixed implementation of Lemma 2.1 for capacities bounded by a fixed power of the network size, each call on a residual graph of linear size; the capacity arithmetic and identifier translations between the calls are counted in addition.*

Proof. The capacities are processed one binary digit at a time, starting from the most significant one, and after each round the current flow is a maximum flow for the capacities formed by the digits

read so far. Begin with zero capacities. In each binary digit round write the new capacity vector as $c' = 2c + \xi$, where every ξ_e is zero or one. If f was a maximum flow of value v for c , then $2f$ is feasible for c' . Let A be the number of original arcs. An old minimum cut has new capacity at most $2v + A$, so the largest residual augmentation has integral value $\Delta \leq A$.

An optimal integral residual augmentation decomposes into source–sink paths and cycles; after the cycles are discarded, its remaining total path value is Δ , so every forward and reverse residual arc carries at most $\Delta \leq A$. Truncating every residual capacity at A therefore preserves the optimum augmentation, which is computed with the fixed polynomial-capacity routine. Adding forward usage and subtracting reverse usage updates $2f$ to a maximum flow for c' . This proves the induction over digits.

Local vertices and arcs can be renumbered, using dictionaries to translate their caller identifiers on input and output. Integer arrays, comparisons, and translations are counted at their bit or multiword cost. $\square$

When paths from an integral flow are needed, they are obtained by pairing incoming and outgoing flow units and tracing source paths, dropping cycles. The work of this extraction is counted as the total represented throughput, which may exceed the number of graph arcs. The routing sparsifier of Section 6 supplies the throughput bounds at each such use.

2.3 A polynomial-time fallback

The final schedule needs one fixed polynomial fallback, which guarantees halting and polynomially bounded machine addresses at every finite input size. Edmonds' deterministic algorithm supplies the following interface.

Lemma 2.4 (Polynomial perfect-matching fallback). *There is one uniform deterministic algorithm on explicit loopless graphs of maximum degree three that halts within a fixed polynomial number of finite-word instructions. It returns a checked perfect matching when one exists and a negative answer otherwise.*

Proof. The algorithm applies Edmonds' deterministic polynomial-time maximum-cardinality matching algorithm [7, Section 2, pp. 451–452]. For parallel edge copies, it first deduplicates unordered endpoint pairs while retaining one original edge identifier for each pair. This preserves perfect matchability and explicit witness decoding. It returns the computed matching if it covers every vertex, and a negative answer otherwise; every returned identifier and vertex incidence is checked before acceptance.

The classical algorithm, the deduplication, and this check run within a fixed polynomial number of steps on every input. Their combinatorial records and polynomially bounded integer indices have polynomial bit cost, so multiword and touched-address implementations retain a fixed polynomial bound in the stated model. Exact maximum cardinality makes the returned answer correct in both cases. $\square$

Section 9 applies this fallback after the witness-preserving reductions of Lemmas 9.2 and 9.3.

3 Matching states and structural operations

We describe a perfect-matching procedure on a loopless graph of maximum degree three. This section and Section 4 together prove Theorem 4.8, the fixed-parameter perfect-matching test for such graphs. This section develops the combinatorial state of the procedure, its ledger transformations, and its

structural searches, and proves the invariants and per-operation guarantees (Lemmas 3.2–3.10) on which the phase analysis relies. The local implementation of projection and the phase analysis follow in Section 4. Both this section and Section 4 use the step primitive of Lemma 5.1, proved in Section 5.

The graph has $N \leq S$ vertices, where $S = 2^s$ is the size bound of Section 2. Disconnected graphs are allowed; parallel edge copies retain separate identifiers, and degree counts incidences. A successful answer is an explicit perfect matching, checked before acceptance. A negative answer may arise from a capped randomized search; the probability of an erroneous negative answer is bounded in Theorem 4.8. The running-time bounds hold for every input graph; the progress bound of Lemma 4.6 is the one place where the existence of a perfect matching is assumed.

3.1 The phase state and its ledgers

The empty graph is answered directly by the empty matching, and odd N gives a negative answer. Otherwise the procedure starts with the empty matching and works in phases. A phase starts with an actual matching M , whose positive deficit is $K = N/2 - |M|$. At deficit zero the procedure is finished. Within a phase we represent the changing matching by odd blocks and single vertices assigned to those blocks.

We call the vertices and edges of the input graph *physical*, to distinguish them from the bins and rows formed from them. A *bin* is a physical odd vertex set with a constructive factor-critical witness: after prescribing any one vertex as its exposure, the others can be matched internally. Its weight is $w_d = (|d| + 1)/2$. The bins form a disjoint family $\mathcal{D}$. The *rows* $\mathcal{A}$ are physical singleton vertices outside them. An explicit injection

$$\mu : \mathcal{A} \longrightarrow \mathcal{D}$$

assigns each row along a specified physical edge to its bin. The bipartite incidence graph $\mathcal{B}$ contains every physical row–bin edge, with different incidences retained separately. Each row has degree at most three.

An assigned bin exposes the physical endpoint of its assigned edge and is matched there to its row. A free, or unassigned, bin exposes one unmatched vertex. The remaining unblocked vertices form a set U of explicitly matched pairs. A phase may also set aside individual unmatched vertices as *blockers*. This state specifies an actual matching, although a changed internal exposure need not be propagated through a bin witness until the bin is *materialized*, that is, until its internal matching for the prescribed exposure is written out explicitly (Lemma 3.10). Initially the bins are the unmatched singleton vertices, U contains all matched pairs, and there are no rows or blockers.

Constructive odd blocks, with an exposed vertex chosen during expansion, follow the blossom contraction-and-expansion tradition of Edmonds [7, Sections 4.10–4.14 and 7.1–7.2]. Here each block carries the witness needed to expand the represented matching; Section 4 proves the required progress bound.

For $X \subseteq \mathcal{D}$, define its captive rows and deficiency by

$$A(X) = \{r \in \mathcal{A} : N_{\mathcal{B}}(r) \subseteq X\}, \quad h(X) = |X| - |A(X)|. \quad (3.1)$$

Each row has its assigned bin as a neighbor. The injection therefore gives $h(X) \geq 0$, even at intermediate states. For a function on bins, such as the weight w , write $w(X) = \sum_{d \in X} w_d$, and similarly for the other bin functions.

Fix integers $a \geq 1000$ and $0 \leq \sigma \leq a + 1$, and set $C = a + 1$; “for sufficiently large S ” has the meaning fixed in Section 2. The phase seeks

$$g_* = \max \left\{ 1, \left\lfloor \frac{K}{100C} \right\rfloor \right\}, \quad L = \left\lceil \frac{10000CN}{K} \right\rceil \quad (3.2)$$

gains, each increasing the matching size by one. We call L the *row allocation*: in every ledger of Definition 3.1, each row allocates exactly L . The choices balance the gain target against the work per gain: $Lg_* = O_a(N)$. When $K < 100C$, this holds since then $g_* = 1$ and $L \leq 10000CN + 1$; the general case is derived in the stage “Ticks” of the proof of Lemma 4.5. Both constants serve the proof of Lemma 4.6: the factor $100C$ keeps the gains and blockers of a phase few enough that at least $K/2$ bins can be freed simultaneously (see (4.16)), and the factor $10000C$ makes L large enough for the directional estimate there. At a phase boundary we materialize the matching and initialize a new state.

For the reset periods, pricing steps (defined in Section 5.1), and exact arithmetic, put

$$B = 2^{\lceil s/a \rceil}, \quad q = B^{-100}, \quad \delta = S^{-1000}, \quad \varepsilon = S^{-3000}. \quad (3.3)$$

Here B is the base of the reset periods (3.11) and enters the time bound of Lemma 5.1; q bounds the weighted length of a pricing step and the relative error of cached values (Section 3.2); δ sets the threshold 3δ at which cached top coordinates are repaired (Section 3.4); and the top ledger uses the grid $\varepsilon\mathbb{Z}$, while the other ledgers use integers. Because δ is a large negative power of S , the repairs add negligible debt: by Lemma 3.5(iv), they add less than S^{-800} on every guarded prefix of a phase; the stage “Objective change and total debt” of its proof counts at most S^{30} repairs of 4δ each.

The ledgers retain enough unallocated capacity at each bin to rearrange the row assignment; debt records a temporary shortfall from the specified residual floor. They have three roles. At settled states the debt-free bottom ledger supplies the strict Hall surplus of Lemma 3.2; the top ledger supplies a smooth progress objective (Section 3.2); and the intermediate integral ledgers make repairs local and amortizable. Adjacent floors differ by one. When a projection fails, this gap gives every coarser ledger preserved through the removal a net debt decrease of at least the removed weight (Lemma 3.6).

A *settled state* is a state at which the main loop of Section 3.3 selects its next action. Before such a selection, the scheduled projection of every nonterminal structural operation has been performed (Section 3.5), and the dirty coordinates (defined in Section 5.1) have been serviced (Section 3.4).

Definition 3.1 (Ledger invariant). There are ledgers $j = 0, \dots, a$, with $c_j = C - j$. Ledger 0 is the *top* ledger and ledger a the *bottom* ledger; a ledger with smaller index is *coarser*, and the *parent* of ledger $j \geq 1$ is ledger $j - 1$. Ledger j has nonnegative incidence allocations x_e^j , with

$$\sum_{e \text{ at row } r} x_e^j = L \quad (r \in \mathcal{A}).$$

Its column residual and nonnegative debt are

$$y_d^j = L - \text{load}_d^j, \quad \text{load}_d^j = \sum_{e \text{ at bin } d} x_e^j, \quad p_d^j \geq 0,$$

and they satisfy

$$y_d^j + p_d^j \geq c_j w_d. \quad (3.4)$$

At the top level,

$$x_e^0 > 0, \quad s_d = y_d^0 + p_d^0 - C w_d > 0 \quad (3.5)$$

are ε -grid coordinates. The other ledgers are integral. At each settled state, the bottom debt p^a is zero.

Ledger j may be viewed as a matrix with one row of sum L for each element of $\mathcal{A}$, which explains the name “row”, and one column for each bin. Initially allocations are empty and debts vanish. Singleton residuals are L , and $L > C$, so the top slacks are positive.

Lemma 3.2 (Hall surplus at settled states). *At each settled state and every nonempty $X \subseteq \mathcal{D}$,*

$$Lh(X) \geq y^a(X) \geq w(X) > 0. \quad (3.6)$$

Thus $h(X) \geq 1$, every bin has weight at most L , and a tight set $h(X) = 1$ has total weight at most L .

Proof. Every captive row allocates all L units inside X , so $\text{load}^a(X) \geq L|A(X)|$ and $y^a(X) \leq Lh(X)$. Zero bottom debt and $c_a = 1$ give $y^a(X) \geq w(X)$. Weights are positive integers, proving the assertions. $\square$

3.2 The top objective and the step primitive

The top ledger chooses its next allocation by increasing

$$\Phi = \sum_{e \in E(\mathcal{B})} \log x_e^0 + \sum_{d \in \mathcal{D}} \log s_d. \quad (3.7)$$

Orient allocation arcs from rows into bins, and add a slack arc s_d from one common auxiliary vertex into each bin. For fixed p^0, w , circulation increments preserve exactly the row sums and the column equations

$$\sum_{e \text{ at } d} x_e^0 + s_d = L + p_d^0 - Cw_d.$$

Conversely, respecting these equations gives a circulation: the common slack-source equation follows by summing them and the row equations.

We use the following specialization of Lemma 5.1 to the top ledger; pricing calls, pricing steps, descriptors, cached values, exact resets, notifications, and dirty coordinates are as defined in Section 5.1. Let f denote the vector of the positive allocation and slack coordinates of the top ledger; here the gradient and derivative-sum lengths are $g_e = \lambda_e = 1/f_e$. A pricing step makes an exact ε -grid circulation increment h , the increment of (5.4), with $\sum_e |h_e|/f_e \leq q/100$, preserves positivity, and increases Φ by $\Omega(q^2)$. The alternative outcome LOW of a pricing call is impossible whenever a circulation direction $d \neq 0$, the direction of (5.3), satisfies

$$\sum_e \frac{d_e}{f_e} \geq \frac{1}{2} \sum_e \frac{|d_e|}{f_e},$$

unless the primitive reports a detected failure or suspends. The sums here range over both kinds of arc in the top graph.

Each top coordinate thus has an exact value f_e , which the primitive stores implicitly, and a cached value $\hat{f}_e$, which it supplies to the client. After its notifications have been serviced, the cached values obey $|f_e - \hat{f}_e| \leq (q/100)\hat{f}_e$. Explicit point reads return exact values, and an explicit setting resets its coordinate exactly, so that the cached value equals the exact one. The primitive reports refreshed coordinates in a list, so the client can repair small arguments and update expansion eligibility without scanning all coordinates after a step. By Lemma 5.1(iii), apart from client settings, the number of these notifications is at most the number of pricing steps times $S^{O(1/a)+o_a(1)}$. Section 5 proves Lemma 5.1, including its time and reliability bounds; Section 3.4 verifies the positivity and representation conditions required of this client.

The procedure has the following guards in the sense of Section 2. It checks the prerequisites

$$S \geq 2^{100}, \quad 100000C \leq S^{1/20}, \quad B \leq S^{1/20}, \quad q \geq S^{-1}. \quad (3.8)$$

It caps main-loop actions, including structural operations and pricing calls, at S^{20} per phase, and separately caps the pricing calls in each local threshold test of Section 4 at S^{20} . Each primitive instance has its S^{100} external-operation guard of Lemma 5.1(iv), together with incrementally checked numerical and representation bounds. Failed prerequisites or deterministic caps suspend the branch. Exhaustion of a prescribed randomized search instead returns a negative perfect-matching answer immediately. For each fixed sufficiently large choice of a, σ , the deterministic checks interrupt no path once S is sufficiently large; this is proved by the analysis of guarded prefixes of computation paths in Section 4 (Lemma 4.7).

3.3 The main actions and the ledger edits

The actions below distinguish progress in matching size from changes to its representation. A gain increases the size by one; contraction and expansion preserve it. A projection may also remove bins, moving their free exposures to the blocker list without changing matching size. The processing of one event, including its projection, is listed in Section 3.5.

At a settled state, take the first applicable action in this order.

1. If a physical edge joins distinct bins i, j , act on this edge as follows. If the injection can be rearranged to leave both free, match their prescribed exposed endpoints along this edge and move their vertices into U . This is a gain. The g_* -th gain ends the phase immediately, without further ledger repair. Otherwise contract the unique minimal tight $X \supseteq \{i, j\}$, with $h(X) = 1$, together with its captive rows to one bin. Lemma 3.8 proves this alternative and supplies the witnesses; Lemma 3.9 gives its randomized implementation.
2. If bin d has an edge to $v \in U$ and cached top slack $\hat{s}_d \geq 4C$, expand along it. Make v a row and its current U -mate a singleton bin, with their old matching edge as the assigned edge of v .
3. If neither action applies, make a pricing call to the top primitive instance of Lemma 5.1. LOW or a detected failure returns a negative answer; suspension suspends the parameter branch.

After each nonterminal structural operation, perform the scheduled projection of Section 3.5 before returning to a settled state.

In a contraction, $|A(X)| = |X| - 1$, so the new block has physical size $2w(X) - 1$ and weight $w(X)$. Delete captive rows and preserve every other assignment. At most one exterior assignment entered X , because captive rows already assigned $|X| - 1$ of its bins. Retain allocations on surviving incidences and aggregate p, y , and top s , by summing over X . This is exact: removing captive rows deletes $L(|X| - 1)$ load, while $|X|$ residual columns become one. New incidences, including edges from a surviving row to a former captive row, now inside the new bin, start at zero in integral ledgers; the top ledger uses the positivity repairs of Section 3.4.

On expansion, ledger $j \geq 1$ allocates the new row's whole L to its assigned child and gives that child debt c_j . Its other new incidences start at zero. At the top, place $C + 1$ units on the new row's incidence into the parent along which the expansion was triggered, and $L - C - 1$ on the assigned child incidence, with no initial child debt. The child has top slack one. The expansion is triggered by a cached slack $\hat{s}_d \geq 4C$, so Lemma 3.5(v) gives parent slack s_d at least $3C$, which remains positive after the $C + 1$ units are placed. Other new top incidences receive the small positive initial values of the positivity repairs.

Gains and removals by a projection delete bins by the following transformation.

Definition 3.3 (Deletion transformation). Given a set R of bins, the *deletion transformation* removes all rows assigned to R . For a removal by a projection, it materializes each assigned bin with its row as matched pairs in U , and for each free bin it places the exposed vertex on the blocker list and the other vertices into matched pairs in U . At a gain the two newly freed bins instead use their joining edge, so no blocker is introduced. A *preserved* ledger is one that is kept through the deletion and updated in place rather than rebuilt by the scheduled projection; Section 3.5 specifies these ledgers (at a removal by the projection of level j , the levels $0, \dots, j-1$). In each preserved ledger, it redirects each allocation of a surviving row into R to the row's current assigned edge, adding an equal debt at the recipient bin. The surviving rows and the bins outside R form the exterior of R . Each application of the deletion transformation is an *erasure* of the bins of R .

Lemma 3.4 (Deletion and net debt). *Let R be a bin set to which the deletion transformation (Definition 3.3) is applied, and consider a preserved ledger of floor cw , that is, a ledger satisfying (3.4) with c in place of c_j . The redirection preserves the ledger invariant (Definition 3.1, with floor cw) on the exterior of R . If O is the redirected mass, I is the allocation of the removed rows to the exterior, and f is the number of free bins removed, then*

$$\begin{aligned} O &= I + Lf - y(R), \\ I &\leq L(h(R) - f), \\ \Delta\left(\sum_d p_d\right) &= O - p(R) \leq Lh(R) - cw(R). \end{aligned} \tag{3.9}$$

The last bound is at most $2L$ at a gain. A gain decreases the represented unmatched count by two; contractions, expansions, and removals by a projection leave it unchanged.

Proof. The debt identities follow by counting the allocation of the removed rows; the unmatched count is checked operation by operation.

Redirection. A surviving row's assigned bin is outside R . The added allocation and debt cancel in $y + p - cw$; removing rows only increases exterior slack.

Debt identities. Exactly $|R| - f$ rows are removed. Their allocation into R is $L(|R| - f) - I$, giving

$$y(R) = L|R| - (L(|R| - f) - I + O) = Lf + I - O.$$

Every captive row is removed, as its assigned bin lies in R . Only the other $h(R) - f$ removed rows contribute to I . Deleting debt $p(R)$ and adding O gives the net debt change, and (3.4) gives the inequality. At a gain, R has two bins and $h(R) \leq 2$.

Unmatched count. There is one unmatched vertex per free bin or blocker. Expansion adds one bin and one assigned row. Contraction replaces $|X|$ bins and $|X| - 1$ assigned captive rows by one bin, preserving any exterior assignment. A removal by a projection transfers free bins' unmatched vertices to blockers. A gain matches two free exposures. This proves the unmatched-count assertions. $\square$

3.4 Lower bounds on the top-ledger coordinates

Structural edits change the graph and its column equations, while pricing preserves those equations. The following local positivity repairs make both kinds of operation compatible with the step primitive.

The *positivity repairs* of the top ledger are the following operations; a *large donor* in a row is an entry of that row with value at least $L/3$ when it is chosen or, in a newly expanded row, one of its two entries specified at the expansion (Section 3.3), which (b) uses as donors.

- (a) A new positive allocation receives 4δ from a large donor in its row, and 4δ debt is added at the destination bin.
- (b) The two specified large entries of a newly expanded row may be installed first and used as donors for its remaining incidences.
- (c) The transfer of (a) is used whenever an existing allocation has cached value at most 3δ .
- (d) If a cached slack is at most 3δ , that slack and its debt are increased by 4δ .

The *service* after each step applies these repairs; it examines only the coordinates touched by structural edits and repairs and the dirty coordinates reported by the primitive. Touched values are read exactly and explicit settings are reset exactly; touched slacks are checked after an exact reset too, including values at the small-value threshold. Notifications of the primitive arising during this service are queued and served.

Structural edits reach the top ledger as edit batches. An explicit edit batch reads affected coordinates exactly, retains unchanged allocations on retargeted incidences, and computes positive final settings before insertions or descriptor changes. The next pricing call is made after the full row and column equations of the batch hold again. The identity $y_d^0 = s_d - p_d^0 + Cw_d$ supplies a top residual without summing the values of implicitly moving incidences. Old descriptors and edge mappings remain available while the primitive processes removals (Section 5.7). The service is completed before the next main action is selected. For expansions, the procedure also maintains bin-to- U contact lists and a cached-slack eligibility index; a U -contact of a bin is an edge from the bin to a vertex of U .

Lemma 3.5 (Lower bounds, debt, and slack thresholds for the top ledger). *Using the positivity repairs (a)–(d) and the service of Section 3.4, the following hold.*

- (i) *The top ledger can be maintained on the ε -grid with logarithmic arguments above δ before pricing and above $\delta/10$ at refreshes.*
- (ii) *Apart from structural work, the repairs use $O(1)$ explicit point operations per dirty coordinate.*
- (iii) *Between structural operations a small-allocation repair decreases Φ by at most $O(\delta)$, and slack repairs never decrease it.*
- (iv) *Positivity repairs add less than S^{-800} debt on every guarded prefix of a phase (Section 2).*
- (v) *The cached and exact slacks satisfy*

$$\begin{aligned} \hat{s}_d \geq 4C &\implies s_d \geq 3C, \\ \text{no eligible expansion} &\implies s_d \leq 5C \quad \text{for every bin with a } U\text{-contact.} \end{aligned} \tag{3.10}$$

Proof. Each repair moves 4δ into a small coordinate within one row, or raises a small slack, and adds the same amount of debt, so the row sums and the column equations of Section 3.2 are preserved. The margins then follow from the cached-value bound, and the debt bound from a count of the repairs.

Effect of one repair. Reading the row's at most three entries finds a large donor, since their sum is L . The destination bin's increased load is canceled by the added debt; the donor bin's slack increases by 4δ . If both bins coincide, their load is unchanged and their slack still increases by that amount.

No repair chains. Here a *motion* is a change of top-ledger values by a pricing step. Between motions or structural changes, each row has at most three entries to repair. A donor initially at least $L/3$ remains above the repair threshold 3δ after these transfers of 4δ . Both changed allocations are reset exactly at nonsmall values. Transfers never decrease a slack, and a slack repair changes no other top coordinate. Notifications arising during this service are queued and served, but price and

forest maintenance itself causes no motion (Lemma 5.1(iii)). Without a new motion or structural change, an already repaired coordinate cannot require another repair. Thus every repair is caused directly by a motion or a structural change.

Positivity margins. For a single positive logarithm the cached-value bound gives

$$|f - \hat{f}| \leq (q/100)\hat{f}.$$

After service, an unrepaired coordinate has cached value above 3δ , while a repaired one has been increased by 4δ and reset exactly. Before a step, a cached value above 3δ thus gives $f \geq (1 - q/100)\hat{f} > \delta$; this bound and the step-size bound leave the stated margins before a step and at refreshes after it. Contraction aggregates positive slacks; deletion increases surviving slacks; expansion has the positive margins specified above. New descriptors are inserted only after their positive final values are computed. All settings are exact ε -grid operations.

Objective change and total debt. Only the large donor's logarithm can decrease during a transfer, by $O(\delta)$; recipient allocation and donor slack increase. A slack repair increases its logarithm. There are $O(S)$ coordinates at any state, with constantly many repairs per coordinate or changed physical incidence between motions or edit batches. By the S^{20} phase-action cap, a guarded prefix has $O(S^{20})$ motions and edit batches, including the last completed edit batch, and since repairs cause no further repairs, each is followed by $O(S)$ repairs. This gives $O(S^{21}) \leq S^{30}$ such repairs on a guarded prefix. Consequently their 4δ debt additions total at most $4\delta S^{30} = 4S^{-970}$, which is less than S^{-800} .

Expansion eligibility. The relative cached-value inequality, with $q < 1$, gives $s_d \geq 3C$ when $\hat{s}_d \geq 4C$; if no eligible contact exists, its cached slack is below $4C$ and its slack s_d below $5C$. $\square$

For the top primitive instance, Lemma 5.1 is applied with nominal lifetime size SB , which takes the place of the nominal bound R of that lemma. Lifetime size counts vertex and edge births and endpoint insertions, including retargeting; value settings, positivity repairs, and repricing on retained topology do not create new incidences. Lemma 5.1(iv) bounds the running time of this instance by

$$(SB + B + \#\{\text{external pricing, point, or edge operations}\})S^{O(1/a)+o_a(1)}$$

instructions under $SB + B \leq S^{1+2/a}$, with polynomial labels renumbered if needed. Section 4.6 verifies this nominal size, and Lemma 4.7 the numerical bounds, on guarded prefixes for sufficiently large S .

3.5 Reset schedule and the exchange problem

A phase has a tick counter initially zero. A contraction or expansion contributes one tick; a gain needing ledger repair contributes L . A removal by a projection contributes no ticks. For $j = 0, \dots, a$, put

$$b_j = B^{a-j}. \tag{3.11}$$

The *period quotient* of level j is the integer part of the tick count divided by b_j . The processing of one event is as follows.

1. The structural operation of Section 3.3 is performed with its ledger edits; the top ledger receives them as an edit batch with the positivity repairs of Section 3.4.
2. If the phase continues (Section 3.3), then after these preliminary structural edits the tick counter advances, and the smallest $j \in \{1, \dots, a\}$ whose period quotient increased during the event is selected. Such a level exists since $b_a = 1$.

3. The exchange projection below constructs a fresh zero-debt j -ledger of floor $c_j w$. The projection may remove one simultaneous bin set by the deletion transformation; the coarser levels $0, \dots, j-1$ are preserved through this removal by Lemma 3.4, and every level after j receives a copy with zero debt.
4. The service of Section 3.4 is completed, and the state is settled.

Whenever the phase continues, the scheduled projection thus restores the debt-free bottom ledger before the next settled state.

For $j > 1$, the projection starts from the integral parent ledger $j-1$ as it stands after the preliminary edits, with allocation x , residuals y , and debt total $P = \sum_d p_d$. For $j = 1$, it uses in place of the top ledger the allocation x with L on each current assigned edge and zero elsewhere, with its residuals y . Define the shortfall $b(d)$ and the excess $u(d)$ of each bin d relative to the floor $c_j w_d$ by

$$b(d) = (c_j w_d - y_d)_+, \quad u(d) = (y_d - c_j w_d)_+, \quad \beta = \sum_d b(d). \quad (3.12)$$

The *exchange network* has source-to-bin capacities $b(d)$, bin-to-sink capacities $u(d)$, and, for every incidence $e = (r, d)$,

$$d \longrightarrow r \quad [x_e], \quad r \longrightarrow d \quad [L+1], \quad (3.13)$$

where brackets indicate integral upper capacities and all lower capacities are zero.

Lemma 3.6 (Exchange projection). *Let $j \in \{1, \dots, a\}$. Let x, y be the allocation and residuals of the integral parent ledger $j-1$, as it stands after the preliminary edits, if $j > 1$, and, if $j = 1$, the allocation with L on each current assigned edge and zero elsewhere and its residuals. Consider the exchange network with capacities (3.12) and (3.13). An integral maximum flow in the exchange network produces either a zero-debt ledger of floor $c_j w$ on the whole state, or a nonempty bin set R whose removal leaves such a ledger on the remaining state. In the latter case,*

$$Lh(R) < c_j w(R). \quad (3.14)$$

Every preserved coarser ledger loses at least $w(R)$ in net debt, and the number f of new blockers satisfies

$$f \leq h(R) < \frac{Cw(R)}{L}. \quad (3.15)$$

Proof. The flow modifies the parent allocation. A flow of full value fills every shortfall from the floor; otherwise the bins reachable in the residual network form a set whose captive rows allocate all their mass inside it, and this forces (3.14).

The modified allocation. For an integral maximum flow F , set

$$x'_e = x_e - F_{d,r}(e) + F_{r,d}(e).$$

Every row has total incoming exchange capacity L , so its outgoing exchange flow is at most L . Thus $x' \geq 0$, its row sums are L , and no row-to-bin arc of capacity $L+1$ is saturated. If $F_{\text{src},d}$ and $F_{d,\text{snk}}$ denote source and sink arc flows, then

$$y'_d = y_d + F_{\text{src},d} - F_{d,\text{snk}}.$$

At value β , every bin with $b(d) > 0$ receives $b(d)$, while a bin with $u(d) > 0$ loses at most $u(d)$, proving $y' \geq c_j w$.

The residual-reachable set. Otherwise let R be the set of all residual-reachable bins. For an incidence $e = (r, d)$, residual capacity from d to r , combining the original arc and the reverse of its companion, is

$$x_e - F_{d,r}(e) + F_{r,d}(e) = x'_e.$$

A positive allocation into a reachable bin makes its row reachable. All that row's neighbors are reachable because its row-to-bin arcs are unsaturated. The rows allocating into R are therefore exactly its captives, which allocate their full L there. Hence $y'(R) = Lh(R)$. Every sink arc from a reachable bin is saturated, whereas the source arc of some reachable bin remains unsaturated. Summing the residual changes over R gives (3.14).

The remaining state. Every source arc into a bin outside R is saturated, as otherwise it would make that bin reachable; so $y' \geq c_j w$ there. A surviving row has no allocation into R , since such a row would be captive and removed. Removing the assigned rows of R only increases exterior residuals. Restricted x' is the requested ledger.

Coarser ledgers and blockers. For a coarser level $\ell < j$, one has $c_\ell \geq c_j + 1$. Lemma 3.4 gives net debt change at most

$$Lh(R) - c_\ell w(R) < -w(R).$$

Thus the net debt of that ledger decreases by more than $w(R)$. Every captive is among the removed assigned rows, giving $f \leq h(R)$; now (3.14) and $c_j \leq C$ give the blocker bound. $\square$

At $j = 1$, construct this network globally and use Lemma 2.1, with Lemma 2.3 if necessary. Its size is $O(S)$ and its capacities are polynomial in S , yielding $S^{1+o(1)}$ time including the arc-flow vector and residual cut. For $j > 1$, the parent inequality $y + p \geq (c_j + 1)w$ permits the local implementation of Lemma 4.1. It returns these allocation modifications and, when needed, the entire residual-reachable removal set in

$$(P + B)S^{O(1/a) + o_a(1)}$$

time. Each primitive instance it calls has conditional probability of detected failure at most $2^{-\sigma} S^{-60}$ by Lemma 5.1(v).

3.6 Tight sets and factor-critical contraction

Whether the injection can be rearranged to free two given bins is a Hall condition on the rows.

Lemma 3.7 (Hall criterion for freeing bins). *An injection of the rows into the bins along incidences of $\mathcal{B}$ that avoids a set $F \subseteq \mathcal{D}$ of forbidden bins exists precisely when*

$$|A(Z)| \leq |Z \setminus F| \quad (Z \subseteq \mathcal{D}).$$

For $F = \{i, j\}$, the criterion becomes

$$h(Z) \geq |Z \cap \{i, j\}|. \tag{3.16}$$

Proof. Necessity is immediate. Sufficiency follows from Hall's condition [16, Theorem 1] applied to any row subset: if Z is its full neighbor set, those rows belong to $A(Z)$, and $Z \setminus F$ is large enough. $\square$

Lemma 3.8 (The tight-set alternative). *Suppose (3.6) holds and an edge joins distinct bins i, j . Either the injection can be rearranged to leave both free, or a unique inclusion-minimal $X \supseteq \{i, j\}$ satisfies $h(X) = 1$. In the latter case $w(X) \leq L$, and the physical block $(\bigcup_{d \in X} d) \cup A(X)$ is constructively factor-critical using the joining edge, its row-bin incidences, and its child-bin witnesses.*

Proof. The failure of the Hall criterion for freeing i and j is a tight set; minimal tight sets are unique by submodularity of h ; and the factor-critical witnesses are injections given by two further Hall conditions.

The alternative. By Lemma 3.7 with $F = \{i, j\}$, the injection can be rearranged to leave both bins free precisely when (3.16) holds for all $Z \subseteq \mathcal{D}$. By the bottom surplus, failure means exactly a tight set containing both bins.

Uniqueness of the minimal tight set. Each neighborhood-inclusion indicator in $|A(Z)|$ is super-modular; thus h is submodular. Two tight sets containing i, j have nonempty intersection and union, both of deficiency at least one. Submodularity forces their intersection to be tight. Intersecting all such sets gives the unique minimal X . Its weight bound follows from Lemma 3.2.

Exposing a vertex of a child bin. To expose a physical vertex v in child bin $d \in X$, match all captive rows into $X \setminus \{d\}$. The Hall inequalities are $h(Z) \geq 1$ when $d \in Z$ and $h(Z) \geq 0$ otherwise, for $Z \subseteq X$. Expose v inside d and each assigned edge's physical contact inside every other child.

Exposing a captive row. To expose a captive row r , match i, j along the joining edge, and match the other captives into $X \setminus \{i, j\}$. The needed Hall inequalities are

$$h(Z) + \mathbf{1}_{\{r \in A(Z)\}} \geq |Z \cap \{i, j\}| \quad (Z \subseteq X). \quad (3.17)$$

A proper subset containing both i, j has $h(Z) \geq 2$ by minimality and integrality. Other nonempty proper subsets use $h(Z) \geq 1$, and for $Z = \emptyset$ both sides vanish. For $Z = X$, the deficiency is one and r is captive. Thus (3.17) holds. Each child exposes its prescribed physical contact, so the child witnesses lift these injections and the joining edge to the desired internal matchings. $\square$

3.7 Randomized search on an edge between two bins

We implement the alternative of Lemma 3.8 for an edge joining distinct bins i, j at a settled state, without examining the physical interiors of visited bins. The search first tests for a small tight set by capacity-one witness trials, refines a witness to the minimal tight set, and otherwise tries to free both bins jointly by random walks. Lemma 3.9 states its guarantees.

The search uses the walk length and repetition count

$$\ell = 16L(1 + s), \quad T_{\text{rep}} = 1000(\sigma + s + 1). \quad (3.18)$$

Every walk below is capped at length ℓ , and every randomized trial is repeated in T_{rep} fresh copies. The walk length makes the probability that a walk is not absorbed within the cap at most $N \exp(-16(1 + s))$, by (3.19) in the proof of Lemma 3.9; the repetition count makes the failure of all copies of a trial exponentially unlikely in $\sigma + s$.

The first stage searches for a capacity-one witness. It uses a directed reverse graph with split vertices $d_{\text{in}} \rightarrow d_{\text{out}}$ of capacity one for each bin. The source has capacity-two arcs to $i_{\text{in}}, j_{\text{in}}$. A free bin's output has a capacity-two arc to the sink. An assigned bin's output has capacity-two arcs to the inputs of all neighbors of its assigned row, retaining individual incidences. Original outdegrees are bounded. The trials use doubling vertex budgets $v_0 = 1, 2, 4, \dots$, through the first budget at least $8L$, with T_{rep} fresh trials at each budget. A trial explores the original graph from the source until reaching the sink or visiting v_0 vertices. It sends one unit along a predecessor path, ending at the sink if reached, and otherwise at a uniformly sampled visited vertex. Then it explores residual reachability with the same budget. The trial accepts the residual source side only if this residual search exhausts it without reaching the sink; the result of a truncated search, or of one that reaches the sink, is rejected. A tight set has $|X| \leq w(X) \leq L$ by Lemma 3.2, so the budgets pass $4|X|$, the size from which a trial succeeds with probability at least one half (stage "Success of a tight-set trial" in the proof of Lemma 3.9).

The second stage refines an accepted witness X' . It lists the captives of X' through its assigned bins, checking each row's bounded neighbor list. On these rows and bins, it uses source-to-row capacities one, row-to-bin capacities κ , bin-to-sink capacities one, and additional source arcs of capacity κ to i, j , where $\kappa = 2|X'| + 3$ exceeds the finite cut values of interest. It starts with the captive assigned edges, a flow of value $|A(X')|$, performs one augmentation, and returns the bins of the smallest source-side minimum cut, found by residual reachability. After refinement, if the witness was found at a budget exceeding $16|X| + 16$, the perfect-matching test is aborted negatively; we call this a *late-witness abort*. Since the budgets double, the first budget at least $4|X|$ is at most $\max\{1, 8|X|\} \leq 16|X| + 16$, so on a tight pair a late-witness abort occurs only after every trial at that budget has failed.

The third stage applies if no tight witness is found: it tries to free the bins jointly. At an assigned bin, a reverse walk follows an incidence of its assigned row with probability x_e^a/L ; free bins absorb. First free i by a successful walk to a free bin: erase loops while keeping the assignment fixed, and then shift each encountered row to the next bin of the resulting path. If i was already free, no change is needed. With the new injection, explore from j until another free bin is reached, or until reaching $8L$ distinct assigned bins. In the first case use its predecessor path. In the second, sample a reached bin uniformly, retain its prefix path from j , and try a bottom-ledger walk from there; concatenate the retained prefix with that walk, erase loops, and shift assignments. The cap $8L$ makes the average probability of hitting i from a sampled bin at most $1/8$ (see (3.20)). Loop erasure uses a stack indexed by bin. The search makes T_{rep} attempts at first freeing and T_{rep} sampled second-freeing attempts. Exhaustion or failure after the capped repetitions gives a negative answer.

Uniform choices are made as follows. A range-uniform draw generates enough independent fair bits to cover the range and rejects out-of-range strings; its string attempts are capped at T_{rep} , returning negative on exhaustion. An integral allocation draw uses a uniform element of $\{1, \dots, L\}$ and the row's at most three cumulative intervals.

Lemma 3.9 (Soundness, error probability, and work of the pair search). *Let a, σ and the state be as in Section 3.1. At a settled state, let an edge join distinct bins i, j , and run on it the search of Section 3.7 with the parameters (3.18). A returned contraction is the minimal tight set with a valid factor-critical witness; a returned gain has a valid joint-freeing witness. Conditional on the current (valid) state, the probability of a premature negative answer, including a late-witness abort, is at most $2^{-\sigma} S^{-60}$. A search that returns a contraction, without a late-witness abort, costs $O(|X| \text{polylog } S)$; a gain attempt or final abort costs $O(L \text{polylog } S)$; in both bounds the constants may depend on σ through T_{rep} . These work bounds hold on every path.*

Proof. The search is sound because only certified witnesses are accepted. Each randomized trial succeeds with constant probability, so T_{rep} repetitions make a premature negative answer unlikely, and a union bound over all draws gives the stated probability. The work bounds follow from the budgets and caps.

Soundness of a capacity-one witness. An accepted source side is fully exhausted and sink-free. Its original outgoing arcs are saturated and its incoming flow is zero, so its original outgoing capacity equals net path flow, at most one. That capacity is positive since the original sink was reachable. It is therefore exactly one. No capacity-two arc crosses the source side: the unique crossing is a split arc, both specified inputs are inside, and every included output has all of its assigned row's neighbors inside. An included output has its own input inside too. It is reached either through its split arc, or through the residual reverse of an original outgoing arc. In the second case that outgoing arc carried the predecessor-path flow, so its split arc also carried flow and has a residual reverse to its input. Thus the included inputs form a bin set X' in which all bins except the one

with the crossing split arc have distinct captive assigned rows. Consequently $h(X') \leq 1$, and the bottom surplus makes X' a tight witness containing i, j .

Refinement. Given a bin source side $Z \supseteq \{i, j\}$, the optimal row source side is $A(Z)$, with cut capacity

$$|A(X')| + h(Z).$$

The minimum is $|A(X')| + 1$. Starting from the captive assigned edges, one augmentation therefore reaches the maximum, and residual reachability gives its smallest source-side minimum cut. Its bins are the unique minimal tight X . This takes linear work in $|X'|$, up to dictionaries.

Soundness of joint freeing. Each retained outgoing transition follows the row assigned at its origin, since the assignment stays fixed throughout path construction. Hence a path to a free bin certifies the reassignment, and an already-free target needs no rearrangement. A successful joint freeing therefore always certifies a gain. Together with the two preceding stages, this proves soundness of every returned contraction and gain.

Success of a tight-set trial. For the minimal tight X , its $|X| - 1$ captive rows are assigned to all but one bin t of X . The set comprising the source, all inputs of X , and all outputs of $X \setminus \{t\}$ has at most $2|X|$ vertices and outgoing capacity one: only t 's split arc leaves it. The original sink is reachable. Otherwise all reachable bins are assigned, and their assigned rows have every neighbor among those bins, contradicting positive deficiency for that nonempty set. When $v_0 \geq 4|X|$, either the endpoint is the sink or exactly v_0 vertices were visited. At most $2|X|$ of the latter lie in the small cut, so the endpoint is outside with probability at least one half. A simple predecessor path to the exterior uses its sole outgoing arc once and cannot return: it would then have to use that same outgoing arc again to reach its endpoint. It therefore saturates the outgoing capacity and has no positive incoming flow across the cut. The cut has no outgoing residual arc. Since $v_0 > 2|X|$, residual search then exhausts without the sink. Thus a tight-set trial succeeds with probability at least one half once its budget is at least $4|X|$. On a tight pair, a late-witness abort requires failure of every trial at the first budget at least $4|X|$, so missing the early witness is exponentially unlikely in T_{rep} .

Freeing walks. For the transition submatrix P_m on assigned bins, each destination column d has sum at most

$$\text{load}_d^a / L \leq 1 - w_d / L.$$

This remains true after an injection rearrangement: each assigned row still supplies exactly one transition row. Since $w_d \geq 1$, the probability of surviving ℓ steps from any start is at most

$$N \exp(-\ell / L). \quad (3.19)$$

Indeed the matrix column norm is at most $1 - 1/L$. Each entry of P_m^ℓ is at most its column sum, and summing a starting row over at most N columns proves the bound. After i is freed, joint freeability guarantees a path from j to another free bin, avoiding i , in the reverse graph in which each assigned bin leads to all neighbors of its assigned row. Otherwise the exhausted set T of reached assigned bins has all its assigned rows captive in $T \cup \{i\}$. This set contains i, j and has deficiency at most one, contrary to (3.16) (Lemma 3.7). Let H_d be the probability of eventually hitting i from an assigned bin, and p_i the one-step column into i . For the assignment fixed during this attempt,

$$H = P_m H + p_i, \quad w^\top H \leq L \mathbf{1}^\top (I - P_m) H = L \mathbf{1}^\top p_i \leq L. \quad (3.20)$$

The first inequality is the destination-column bound with $H \geq 0$; the last uses $\text{load}_i^a / L \leq 1$. As weights are at least one, the average hit probability over the $8L$ reached bins is at most $1/8$. Together with (3.19), this gives a constant chance of reaching another free bin within the walk cap while

avoiding i ; the concatenation then frees j while leaving i free. For a jointly freeable pair, the first walk has constant success probability by (3.19). The second has probability at most $1/8$ of hitting i , by (3.20), and its nonabsorption probability at the length cap is bounded by (3.19). With $\ell = 16L(1 + s)$ and $N \leq 2^s$, their sum is at most $1/8 + N \exp(-16(1 + s)) \leq 1/8 + e^{-16}$, strictly below one. Unsuccessful trials do not change the injection, so the same conditional estimates apply repeatedly.

Random bits and the union bound. Each elementary attempt of a range-uniform draw succeeds with probability at least one half, so a capped draw fails with probability at most $2^{-T_{\text{rep}}} = 2^{-1000(\sigma+1)}S^{-1000}$. Under (3.8), fewer than S^{10} such draws occur in one pair search, including every budget and trial. A union bound over constant-probability trial failures and capped fair-bit rejection failures gives $2^{-\sigma}S^{-60}$. Equivalently, the capped process coincides with unlimited exact rejection until its first cap failure.

Work bounds. On every path, a search that returns a contraction costs $O(|X| \text{polylog } S)$, with constants depending on σ through T_{rep} ; a final abort may cost $O(L \text{polylog } S)$. Doubling budgets sum to a constant multiple of the last, so the post-refinement budget test gives $O(|X| \text{polylog } S)$ for a returned contraction even on an unfavorable random path. The one pushed path creates only linearly many residual reversals, so each bounded exploration takes time linear in its budget, up to dictionaries. Other attempts use $O(L)$ vertex caps, $O(L \log S)$ walk caps, and the repetition counts T_{rep} . The search generates outgoing reverse arcs from the bin and assignee data and from the assigned row's physical neighbor list, of degree at most three, and it keeps path-created residual reversals and local visited records in dictionaries. Hence no query scans all physical vertices of a visited bin. Random bits, initialization, loop erasure, record operations, and explicit path or witness output are included in the work. $\square$

3.8 Matching materialization

A bin is *dissolved* when it is materialized and its vertices leave the bin family, as at an erasure (Definition 3.3) or at a phase boundary.

Lemma 3.10 (Materializing a bin). *A current bin can be materialized with any prescribed physical exposure in time linear in its physical size up to logarithmic factors. All current bins can be dissolved at an erasure or phase boundary in time linear, up to logarithmic factors, in their total physical size.*

Proof. Each contraction stores its abstract incidence graph. A prescribed exposure determines at most two augmentations at each node of the resulting contraction tree, by Lemma 3.8 and (3.17), and the recursion visits each child once.

Contraction records. At a contraction store the child-bin records, captive rows and their physical incidences, joining edge, and captive assignment at contraction time. This gives a contraction tree whose leaves include both child-bin vertices and captive-row vertices. The stored captive assignment covers all but one child.

Exposure in a child. If the desired exposure v is in child d , forbid d and remove its captive assignment if present. At most one row becomes unmatched. Lemma 3.8 guarantees an injection of all captives avoiding d , so at most one ordinary row augmentation suffices. If d was the unused child, none is needed. Expose v inside d , and the physical assigned-edge contact in every other child. A preserved exterior assignment merely prescribes v to be its contact endpoint.

Exposure at a captive row. If the desired exposure is a captive row r of the contraction, delete r and its assigned edge, reserve i, j for the stored joining edge, and remove any remaining assignments into those children. At most two remaining rows become unmatched. Inequality (3.17) guarantees a

matching of the remaining rows into $X \setminus \{i, j\}$, so at most two augmentations suffice. The joining edge prescribes exposures in i, j ; every other child gets its row-edge contact.

Recursion and total work. In either case each child receives exactly one prescribed exposure and is recursively materialized once. The searches scan only this node's stored abstract incidence graph, of size linear in its arity because rows have degree at most three. At the start of dissolving a root, one traversal assigns subtree leaf intervals. The interval containing a prescribed physical vertex identifies its direct child, with logarithmic search overhead if needed. No contraction is unary, and each record has size linear in its arity. The sum of record sizes and leaves is therefore linear in the dissolved root's physical size. Summing the local searches, endpoint lookups, and single child visits proves the bound. Disjoint roots at an erasure or phase boundary can be handled in the same total physical-size time. $\square$

In the accounting of Section 4.6, any later return of these vertices to bins is counted as a new period of bin membership, begun by initialization or by an expansion (see also Lemma 4.5). Thus each dissolved contraction record is counted exactly once, for its materialization, even when later descents pass through an unchanged nested witness.

4 Local projection and the perfect-matching bound

This section completes the phase algorithm of Section 3. It implements the projection onto an integral ledger $j > 1$ locally, proves the progress and cost bounds of a phase, and assembles the fixed-parameter perfect-matching test. We use the state notation of Section 3, its prerequisite checks (3.8) and the caps stated after them, and the step primitive (Lemma 5.1), which is proved in Section 5. Asymptotic notation follows the convention of Section 2.

The results of the section combine into Theorem 4.8 as follows. Lemma 4.1 performs each scheduled projection with $j > 1$ in time $(P + B)S^{O(1/a) + o_a(1)}$, where P is the debt of the parent ledger. Lemma 4.5 bounds the removed weight, the blockers, the expansions, the contractions and the ticks of a phase. Together with the parent-debt bound (4.21), these results give the phase-time bound (4.23) of Section 4.6. The proof of Theorem 4.8 bounds the number of phases by $O_a(\log S)$, so the total time is $S^{1 + O(1/a) + o_a(1)}$. Lemma 4.7 shows that, for sufficiently large a and S , no deterministic guard fails, so that this time bound holds on every path and no suspension occurs. When the graph has a perfect matching, Lemma 4.6 excludes a LOW answer before the phase target is reached, and the step bound (4.20) excludes an infinite sequence of pricing steps. A union bound over the instances of the step primitive and the pair searches of Lemma 3.9 gives the error bound (4.25). The proof of Theorem 4.8 in Section 4.7 carries out this assembly.

The threshold tests use the potential-reduction viewpoint of approximate minimum-ratio circulations developed by Chen et al. and its dynamic threshold formulation by van den Brand, Liu, and Sidford [3, 38]. The logarithmic objective, local exploration and projection guarantees needed here are proved directly.

4.1 The local projection lemma

The exchange network of Lemma 3.6 has one vertex for every bin and row. Its source and sink will be denoted by s_*, t_* , to distinguish them from the size exponent in $S = 2^s$. Recall from (3.12) that, for the integral parent ledger $j - 1$ and the requested floor cw with $c = c_j$,

$$b(d) = (cw_d - y_d)_+, \quad u(d) = (y_d - cw_d)_+, \quad \beta = \sum_d b(d), \quad P = \sum_d p_d. \quad (4.1)$$

The floor (3.4) of the parent is one unit higher:

$$y_d + p_d \geq (c + 1)w_d. \quad (4.2)$$

In particular, $\beta \leq P$. Lemma 3.6 requires a maximum exchange flow and, when the maximum is below β , its full residual source-reachable bin set. The following lemma computes this information without scanning every bin.

Lemma 4.1 (Local exact projection). *Let a , σ and C be as in Section 3.1, B as in (3.3), and c and P as in (4.1). Suppose an integral parent ledger satisfies (4.2) in a partition state of a subcubic graph on at most S physical vertices. Its allocations, column residuals, weights, and positive debts support point access and sparse indexed updates. Assume $P = O_a(S)$ for the fixed-parameter asymptotic bound. Then the exact exchange projection can be implemented in*

$$(P + B)S^{O(1/a)+o_a(1)} \quad (4.3)$$

total time on every path, at all sufficiently large sizes for fixed sufficiently large parameters.

The implementation either returns the exact exchange-flow changes and, if needed, the full residual reachable set R of Lemma 3.6; reports a detected stochastic failure; or suspends on a deterministic prerequisite check. It uses $O(\log S)$ instances of the step primitive. For each reached instance with valid prerequisites, the conditional probability of a detected failure is at most $2^{-\sigma}S^{-60}$. All updates to a fresh child ledger and all necessary physical scans are confined to $O(P)$ represented incidences and bin weight, apart from the padding and subpolynomial factors in (4.3). The stated deterministic prerequisites do not cause suspension in the sufficiently large fixed-parameter range.

The construction tests whether the full exchange maximum is at least an integer threshold. An optimistic local network supplies an upper bound on that maximum, and the same explored region with the capacities of the exchange network supplies a lower bound (Section 4.2). Section 4.3 constructs and analyzes this test, and Section 4.4 shows how two of its regions recover the full residual cut and completes the proof of Lemma 4.1.

4.2 Local exploration and bounds on the exchange maximum

Assume $\beta > 0$ for the threshold construction. The exploration starts with every *seed* bin, that is, every bin satisfying $p_d \geq w_d/2$. Their total weight is at most $2P$. The index of positive integral debts has at most P entries, so listing these seeds and computing their deficits takes $O(P)$ point operations. Every deficit is a seed: if $y_d < cw_d$, then (4.2) gives $p_d > w_d$.

Expanding a bin exposes all its bin-to-row exchange arcs, its source and sink capacities in the exchange network, and all row-to-bin arcs and endpoints of the rows thereby discovered. Arc identifiers are retained, so an already represented arc is not inserted a second time. The seeds are expanded. An unexpanded bin reached as an endpoint is a *boundary*. It has no represented bin-to-row arcs yet and no positive source capacity, since it is not a seed. The *region* of an exploration consists of its expanded bins, its boundaries, the rows discovered so far, and all represented arcs; it grows whenever a further bin is expanded. The *optimistic network* of the region has the represented arcs with their exchange capacities, gives every boundary a sink arc of capacity $\beta + 1$, and has zero lower bounds. The sink capacity $u(d)$ of a boundary in the exchange network is available by a point lookup, and (4.2) implies

$$u(d) = y_d - cw_d \geq w_d - p_d > w_d/2. \quad (4.4)$$

In particular $u(d) \geq 1$, by integrality. The *restricted network* of the region has the same represented arcs, but uses the exchange sink capacity $u(d)$ at every boundary. It embeds into the full exchange network.

Lemma 4.2 (An upper bound on the full exchange maximum). *Let β be as in (4.1), and assume $\beta > 0$. For every region obtained by the exploration of this subsection, which expands all seeds and possibly further reached bins, the maximum flow value of its optimistic network is at least the maximum flow value of the full exchange network of Lemma 3.6.*

Proof. Take an integral path decomposition of a maximum flow of the full exchange network and discard its cycles. Each source path starts at a seed. Truncating each path at its first boundary, if any, and sending it from there directly to the sink gives a flow in the optimistic network of the same value. Indeed, the retained interior arcs are represented with their exchange capacities, and the total flow is at most β , less than every optimistic boundary sink capacity. $\square$

4.3 The threshold test

4.3.1 The scaled network and its objective

The optimistic and restricted networks have zero lower bounds. For optimization we use a third network on the represented arcs, the *scaled network* of the test: its capacities are scaled, its logarithms have padded arguments, and some sink arcs acquire permanent positive lower bounds. These lower bounds limit the total explored weight. Lemma 2.2 converts its exactly conserved real points into integral feasibility statements, with the total violation including the return arc.

The test with integer threshold θ , where $1 \leq \theta \leq \beta$, begins a fresh exploration and multiplies all ordinary capacities by 16. It introduces a return arc $t_* \rightarrow s_*$ with variable z , and sets

$$T = 16\theta, \quad W = 20000(P + 1). \quad (4.5)$$

The objective is

$$\Phi(F, z) = \sum_{e \text{ ordinary}} \left[\log(F_e - \underline{c}_e + \zeta_e^-) + \log(\bar{c}_e + \zeta_e^+ - F_e) \right] - W \log(T - z). \quad (4.6)$$

Here $\underline{c}_e, \bar{c}_e$ are the current unpadded bounds, initially with lower bound zero, and $\zeta_e^-, \zeta_e^+ > 0$ are dyadic pads. At the current point, write $g = \nabla \Phi$, and let λ_e be the sum of the absolute derivatives of the individual weighted logarithmic terms on coordinate e , as in Lemma 5.1. In particular, this is not in general $|g_e|$: the two ordinary-arc derivatives can cancel, whereas on the return arc $g_z = \lambda_z = W/(T - z)$. All flows start at zero. Added arcs also start at zero, and every subsequent motion is an exact circulation supplied by Lemma 5.1; no client operation changes an existing flow value. Figure 1 shows the scaled network and the transition from an optimistic boundary to a permanently locked sink.

The four networks of the construction and their roles are the following.

Network	Arcs and bounds	Role
Full exchange network (Section 3.5)	all bins and rows; capacities $b(d), u(d), x_e, L + 1$; zero lower bounds	its maximum flow value v is to be determined
Optimistic network of a region	represented arcs with exchange capacities; sink capacity $\beta + 1$ at every boundary; zero lower bounds	maximum at least v (Lemma 4.2); shows $v < \theta$ after Low
Restricted network of a region	represented arcs with exchange capacities, including $u(d)$ at every boundary; zero lower bounds	embeds into the full network, so its maximum is at most v ; shows $v \geq \theta$ after success
Scaled network of a threshold test	represented arcs with capacities multiplied by 16; return arc $t_* \rightarrow s_*$ with flow z ; padded bounds; locks (4.10)	optimized by the step primitive with objective (4.6); converted to the unscaled networks by Lemma 2.2 and division by 16 (Lemma 4.3)

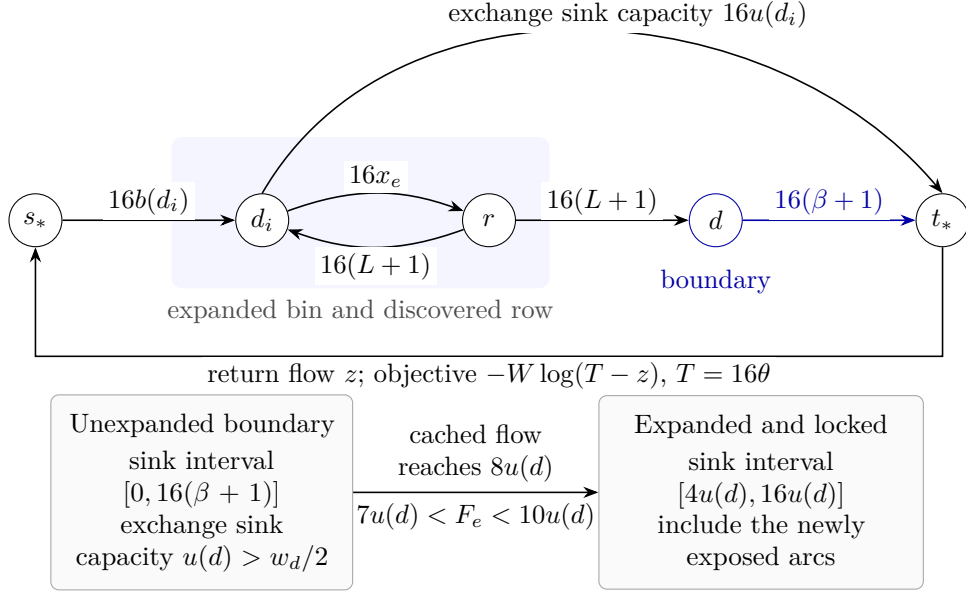


Figure 1: The scaled network of a threshold test and the boundary transition in Lemma 4.1. Labels on ordinary arcs are scaled upper capacities; omitted unpadded lower bounds are zero. The schematic shows one expanded incidence and one boundary incidence. Initially all seeds (bins with $p_d \geq w_d/2$), and therefore all deficit bins, are expanded. A boundary has no outgoing bin-to-row arcs and no positive source capacity until expansion. Padded bounds are used by the step primitive. Expansion keeps the current flow and installs a permanent lower lock. If the last step succeeds before a new trigger is served, the flow F_e on the boundary's sink arc e still satisfies $F_e < 10u(d) < 16u(d)$, as in (4.11), so the exchange sink capacity is respected up to padding.

4.3.2 Padding of the logarithmic arguments and the range of the return flow

Each pad starts at 4δ . If a cached argument is at most 3δ , its pad increases by 4δ and that coordinate is reset exactly. Explicit changes of bounds also reset it exactly. Increasing a pad only increases the objective. The cached-value and step-size guarantees (5.5) and (5.4), exactly as for the positivity margins of Lemma 3.5(i), keep all ordinary arguments above δ before a new step and above $\delta/10$ at a refresh after motion. A repaired argument does not need another increase without further motion or a change of its bounds. The repairs can therefore be made on dirty and newly inserted coordinates, without a scan of the represented graph at each step.

The caps stated after (3.8), namely S^{20} main-loop actions per phase, S^{20} pricing calls per threshold test, and the S^{100} external-operation cap, give, on every guarded prefix (Section 2), a total sum of pads

$$\alpha := \sum_{e \text{ ordinary}} (\zeta_e^- + \zeta_e^+) < S^{-800}. \quad (4.7)$$

One may bound the number of potential repairs using all $O(S)$ ordinary arcs of the full exchange graph. Alternatively, the external-operation cap gives at most a constant multiple of S^{100} pad settings, each of size $O(\delta)$, and their total is smaller than the right side of (4.7) for large S . There is no intervening infinite repair chain: a repaired argument is reset exactly, and the current point does not move during the service. All ordinary unpadded-bound violations sum to at most α . Conservation at the source gives

$$-\alpha \leq z \leq 16\beta + \alpha. \quad (4.8)$$

At each loop test the algorithm reads z exactly. It stops successfully when $z \geq T - 1/4$. Otherwise the next step cannot cross T : the return contribution to its length is $W|\Delta z|/(T - z)$, and the step's total length is at most $q/100$. Thus its relative return-argument change is at most $q/(100W)$. In particular even the final successful step leaves

$$T - z \geq 1/8. \quad (4.9)$$

4.3.3 Expansion of boundaries and the size of the explored region

Before another pricing call, the test expands every boundary whose cached sink flow is at least $8u(d)$ and replaces its unpadded optimistic sink interval by

$$[4u(d), 16u(d)]. \quad (4.10)$$

This lower lock is retained for the rest of the threshold test, and the newly discovered arcs are added with flow zero. Pads can be reinitialized when the sink bounds change.

The transition is interior with a constant margin. At the start of a step, an untriggered boundary has flow less than $8.2u(d)$, by the cached-value error inequality (5.5) applied to its padded lower argument (whose unpadded bound is zero) and by $u(d) \geq 1$. During the step its absolute change is at most $(q/100)(F_e + \zeta_e^-)$, so its resulting flow is less than $10u(d)$. At a first trigger the same cache guarantee gives flow greater than $7u(d)$. Hence the new interval (4.10) is safely interior. Triggers are checked on dirty or newly supplied entries using their latest cached values. An exact reset without motion cannot invalidate the estimate. In particular, if success is detected immediately after a step, before a newly triggered boundary has been serviced, that boundary still has

$$F_e < 10u(d) < 16u(d). \quad (4.11)$$

Thus a successful exit requires no additional scan or expansion.

Every previously locked sink carries at least $4u(d)$, apart from its pad violation; each just-triggering sink carries more than $7u(d)$. All other sink flows are nonnegative apart from their pad violations, and their sum is z . Each nonseed expanded or triggering bin satisfies $2w_d < 4u(d)$ by (4.4). Since the pad violations sum to at most α and $z \leq 16\beta + \alpha$ by (4.8), twice the total weight of these bins is less than $16\beta + 2\alpha$, and $\beta \leq P$ bounds this weight by $8P + 1$. Including seeds, the expanded weight is at most

$$11(P + 1). \quad (4.12)$$

This also bounds physical scanning: an expanded bin has $2w_d - 1$ vertices of degree at most three. Per unit of expanded weight there are at most six bin-to-row incidences, at most eighteen row-to-bin incidences, and at most eighteen boundary occurrences. Including source and sink arcs, there are fewer than $1000(P + 1)$ ordinary arcs. Each is inserted once, the region only grows, and each sink is locked at most once. The same bound thus controls cumulative topological insertions in this test.

The numerical constants of a threshold test enter the argument through the following relations.

Constant	Role
factor 16	scales all ordinary capacities; $T = 16\theta$, and the integral certificate is divided by 16 at the end of the proof of Lemma 4.3
$W = 20000(P + 1)$	ten times the bound $2000(P + 1)$ on the number of ordinary logarithmic terms; gives the ratio $> 1/2$ in the proof of Lemma 4.3
pads 4δ , repair at 3δ	keep ordinary arguments above δ before a step and above $\delta/10$ after motion; total $\alpha < S^{-800}$ by (4.7)
stop at $z \geq T - 1/4$	the return violation at success is at most $1/4 + \alpha$, so the total violation is below one; the last step leaves (4.9)
trigger $8u(d)$	a first trigger occurs with flow between $7u(d)$ and $10u(d)$, inside the lock interval (4.10)
lock $[4u(d), 16u(d)]$	the upper end is the scaled exchange sink capacity; the lower end exceeds $2w_d$ by (4.4) and yields (4.12)
$8192(P + 1)$	an example of a sufficient nominal lifetime size of the primitive instance of one test: it bounds every new endpoint and arc of the test and satisfies hypothesis (5.2) (Sections 4.3.5 and 4.7)

4.3.4 The threshold certificate and the decision

Lemma 4.3 (Threshold certificate). *Consider a threshold test of Section 4.3 with threshold θ and T as in (4.5), and the optimistic and restricted networks of its region (Section 4.2). Suppose that the test has neither failed nor suspended. If the maximum flow value of the optimistic network is at least θ , equivalently at least T after all capacities are multiplied by 16, the test does not report LOW. If the test exits successfully, the maximum flow value of the restricted network is at least θ .*

Proof. Both conclusions follow by applying Lemma 2.2 to the current exactly conserved point, with two different sets of integral bounds.

Exclusion of LOW. Consider the current integral ordinary bounds, including the locks, together with the return interval $[0, 16\beta]$. The current point is exactly conserved. Its ordinary violation is at most α , and (4.8) adds at most α more for the return interval. The total violation, including the return arc, is at most $2\alpha < 1$. Lemma 2.2 supplies an exactly feasible integral circulation for these bounds.

Starting from its ordinary flow, one can attain the optimistic maximum while preserving all sink locks. If the present value is smaller, the ordinary residual network contains a source-to-sink path; erase cycles to make that path simple. The only additional lower bounds are on incoming sink arcs. Their reversals leave the sink and cannot occur on a simple source-to-sink path. Every augmentation

therefore preserves the locks, even if the starting flow contained cycles. This constructs, existentially, a feasible comparator (F', z') with $z' \geq T$.

The difference from the current point is a nonzero circulation. For each ordinary logarithmic term, a negative directional derivative has magnitude at most one, since the comparator is feasible even without the pads. There are fewer than $2000(P+1)$ such terms. The return term contributes

$$\frac{W(z' - z)}{T - z} \geq W$$

positively. If A_+ and A_- are the sums of positive and absolute negative individual contributions, then

$$A_+ \geq 20000(P+1), \quad A_- < 2000(P+1), \quad \frac{g^\top(F' - F, z' - z)}{\|\lambda(F' - F, z' - z)\|_1} = \frac{A_+ - A_-}{A_+ + A_-} > \frac{1}{2}.$$

The comparator enters only through the direction $(F' - F, z' - z)$, so Φ need not be defined at (F', z') . Lemma 5.1 rules out LOW.

Certificate at success. At a success exit, consider the bounds obtained by replacing the optimistic boundary capacities by the exchange capacities and imposing the return interval $[T, 16\beta]$. By (4.11), no unexpanded boundary violates its exchange upper capacity, even before trigger service. There are no unexpanded deficits. The ordinary violation is at most α ; the additional return violation is at most $1/4 + \alpha$. The total is less than one. Lemma 2.2 gives a feasible integral scaled flow of value at least $T = 16\theta$ in the restricted network. Dividing capacities and flow value by 16 proves that its unscaled maximum is at least θ . $\square$

The test continues until success, LOW, detected stochastic failure, or suspension. On LOW, a static integral maximum-flow computation verifies that the maximum of the optimistic network, in unscaled units, is below θ . On success, it verifies that the maximum of the restricted network, in unscaled units, is at least θ . A contradictory exact check causes suspension. By Lemma 4.3, if the test has neither failed nor suspended before the check, the check confirms the reported outcome. These calls use Lemmas 2.1 and 2.3 with size padded to $O(P+B)$. Capacity digit scaling in particular avoids a variable polynomial-capacity exponent when P is small relative to S .

Thus each verified test decides the threshold for the full graph. On LOW, Lemma 4.2 implies that the full maximum is below θ . On success, the restricted network embeds into the full graph and supplies value at least θ . The scaled network is discarded after the test; the exact checks use the optimistic or exchange capacities, respectively.

4.3.5 Number of steps and time of one test

The objective range at loop and service states is $O_a((P+1)\log S)$: there are $O(P+1)$ ordinary terms with arguments between $\delta/10$ and a checked upper bound polynomial in S (Lemma 4.7), and the weighted return term has the same total order by (4.9) and $W = O(P+1)$. Pad increases never lower the objective. An arc insertion or a lock change decreases the objective by at most $O(\log S)$, and there are only $O(P+1)$ such touches. The $\Omega(q^2)$ increase per successful pricing step consequently bounds their number by

$$O_a((P+1)q^{-2}\log S). \tag{4.13}$$

The primitive instance of a test is given nominal lifetime size $8192(P+1)$, for example; this size includes every new endpoint and arc. Point reads of the return arc, local pad changes, boundary

notifications, and the final static verification are included in the time of the test, and a pricing call makes no global scan. Since $q^{-2} = B^{200} \leq 2^{200} S^{200/a}$, the step count (4.13) is $(P + 1)S^{O(1/a) + o_a(1)}$; by the notification bound of Lemma 5.1(iii), so is the number of external operations of the test, which takes the place of the operation count of Lemma 5.1(iv). With the nominal size $8192(P + 1)$ in place of the nominal bound of that lemma, the time bound (5.6) therefore gives (4.3) for one threshold. The guard S^{20} on the pricing calls of a threshold test is eventually larger than (4.13) because $q^{-2} = O_a(S^{200/a})$ and $P = O_a(S)$.

4.4 Recovering the exact flow and the full residual cut

The threshold tests determine the maximum value, but the matching procedure also needs the exact allocation changes and, when a deficit remains, the entire residual source-reachable bin set. We now recover this information from the small regions explored by the tests.

Lemma 4.4 (Recovering the full residual cut). *Let β be as in (4.1), and let regions and their networks be as in Section 4.2. Suppose $\beta > 0$, and let v be the full unscaled exchange maximum. Suppose that, when $v > 0$, a region whose restricted network has maximum v is available, and that, when $v < \beta$, a region whose optimistic network has maximum v is available. The union of the available regions, using exchange capacities and original arc identifiers, suffices to compute an exact full exchange flow and, when $v < \beta$, the full residual source-reachable bin set.*

Proof. Suppose $v < \beta$, and let Q be a minimum source side of the supplied optimistic network. No boundary belongs to Q , because its sink arc alone has capacity $\beta + 1 > v$. Consequently every bin of Q is expanded and all its outgoing bin-to-row arcs of positive exchange capacity are represented. Every represented row has all its outgoing row-to-bin arcs, and every positive source arc is present because all deficits are seeded. Thus all arcs leaving Q in the full exchange network occur with the same capacity in the supplied optimistic network. The same set Q is a full-graph cut of capacity v .

The supplied restricted network has value v when $v > 0$. The algorithm solves an exact integral maximum flow in the union with exchange capacities, using the optimistic network alone when $v = 0 < \beta$, or the restricted network alone when $v = \beta > 0$. The union's value is exactly v , by the cut just proved or by the total source capacity β . Its extension by zero is a flow of the full network.

When $v < \beta$, equality of its value with the capacity of Q forces every outgoing cut arc to be saturated and every incoming cut arc to have zero flow. Neither an unsaturated forward arc nor a positive-flow reverse arc can leave Q . All positive-capacity original arcs needed within Q are represented, and every needed reverse arc is the reverse of an arc carrying positive flow in the union. Residual reachability computed in this local representation is therefore the full residual source reachability. Its bin part is precisely R . $\square$

Proof of Lemma 4.1. The algorithm finds v by binary search over threshold tests, recovers the flow and the residual set by Lemma 4.4, and the time bound follows from the per-test bound and the size (4.12) of the explored regions.

Seeds and the case $\beta = 0$. Listing the positive parent debts finds the seeds and computes β , in $O(P)$ point operations. If $\beta = 0$, the unchanged allocation already satisfies the requested floor, so giving the fresh child an empty debt map finishes the projection. The parent ledger remains unchanged.

Binary search and recovery. Otherwise binary search, using the verified threshold tests, determines the full unscaled maximum v . The algorithm retains a region whose restricted network has maximum v when $v > 0$. If $v < \beta$, it retains the optimistic network of a test with threshold $v + 1$ that reported LOW. Its maximum is exactly v : it is below $v + 1$ by the test, at least v by

Lemma 4.2, and integral. At most a constant number of additional threshold tests retrieves these regions. Lemma 4.4 then returns the exact flow and, when needed, its full residual bin set.

Size and time. The union has size $O(P + 1)$. Its flow support supplies all allocation changes. The set R has weight $O(P)$, either by (4.12) or by Lemma 3.6: the parent, as a preserved coarser ledger, loses at least $w(R)$ in net debt, so $w(R) \leq P$. Listing its assigned rows and their affected allocations requires only that physical weight and bounded row degree. There are $O(\log S)$ threshold tests, including the tests that retrieve these regions, because $\beta \leq P = O_a(S)$. The per-test bound (4.3), the final static flow on the union, and these sparse ledger changes give the asserted total time. For each reached primitive instance, the conditional probability of a detected failure is at most the bound of Lemma 5.1; a detected failure or suspension is returned to the caller immediately. The numerical and nominal-size prerequisite checks, including those used above, are justified uniformly on guarded prefixes in Lemma 4.7. $\square$

4.5 Debt, structural work, and progress

Recall from Section 2 that a guarded prefix of a computation path ends before the first failed guard of the path. The estimates of this subsection apply on guarded prefixes, for every input graph, whether or not it has a perfect matching; on such a prefix every returned structural action or projection satisfies its deterministic invariants. A *removal* is the deletion of the bin set R of Lemma 3.6 by a projection; the sum in (4.14) ranges over these removals.

4.5.1 Debt and structural counts in a phase

Lemma 4.5 (Bounds on removals and structural operations in a phase). *Let C be as in Section 3.1. In a phase with initial deficit K , row allocation L , and target g_* as in (3.2), every guarded prefix before completion satisfies*

$$\sum_{\text{removals}} w(R) \leq 2L(g_* - 1) + S^{-800}, \quad \#\{\text{blockers}\} \leq 2C(g_* - 1). \quad (4.14)$$

There are $O(N + Lg_*)$ expansions, the sum of contraction arities $|X|$ is $O(N + Lg_*)$, and the total tick increase is $\tau = O_a(N)$.

Proof. The first bound compares the debt added by repaired gains with the debt removed by projections; the counts of expansions and contractions follow from the bottom certificates, and the tick bound from (3.2).

Removed weight and blockers. Before the terminal g_* -th gain there are at most $g_* - 1$ gains whose ledgers are repaired. Top contractions preserve total debt, top expansions use available slack without adding other debt, and the positivity service of Lemma 3.5(iv) adds less than S^{-800} debt. By Lemma 3.4, each repaired gain adds at most $2L$ net debt. By Lemma 3.6, every removal by a projection decreases the debt of the top ledger by at least the removed weight. Hence

$$\sum_d p_d^0 + \sum_{\text{removals}} w(R) \leq 2L(g_* - 1) + S^{-800}, \quad (4.15)$$

and nonnegative remaining debt gives the first bound in (4.14).

The number of new blockers at a removal is at most $h(R) < Cw(R)/L$, by (3.15). Summing this strict inequality and using integrality gives the second bound in (4.14); the tiny positive remainder is less than one. In particular, before the first gain of a phase with $g_* = 1$, no removal of positive weight can occur.

Expansions and contractions. The bottom certificate of Lemma 3.2 bounds the weight removed by a gain by $2L$. A removed bin has $2w_d - 1$ physical vertices, and at most one assigned row per removed bin is materialized with it. Thus the number of vertices returned to U is bounded by a constant times the total removed weight. Every expansion takes a pair from U . The initial population of U is at most N , so expansions number $O(N + Lg_*)$. Initially there are at most N bins and each expansion creates one new bin. A contraction decreases their number by $|X| - 1$, and $|X| \geq 2$; hence the sum of $|X|$ over contractions is also $O(N + Lg_*)$.

Ticks. Expansions and contractions contribute one tick each, and a repaired gain contributes L , giving $\tau = O(N + Lg_*)$. Finally, the bound $Lg_* = O_a(N)$ stated after (3.2) follows from that display: if $K < 100C$, then $g_* = 1$ and $L = O_a(N)$; otherwise

$$Lg_* \leq \left(\frac{10000CN}{K} + 1 \right) \frac{K}{100C} = O(N).$$

This proves the tick bound. □

4.5.2 An improving direction at a stall

We call a settled state at which neither a distinct-bin contact (action 1 of Section 3.3) nor an eligible expansion (action 2) is available a *stall*; at a stall the main loop makes a top pricing call. In the next lemma, g and λ are the gradient and the absolute-derivative vector of the top objective (3.7) at the current point, and a direction is a nonzero circulation, as in Lemma 5.1(i).

Lemma 4.6 (Improving direction and progress when a perfect matching exists). *Let g_* be as in (3.2), assume that the subcubic graph has a perfect matching, and consider a settled phase state with fewer than g_* gains. If neither a distinct-bin contact nor an eligible expansion is available, the top objective has a circulation direction ξ with*

$$g^\top \xi > \frac{1}{2} \|\lambda \xi\|_1.$$

The state admits neither a valid LOW exit nor an unending sequence of pricing calls without another structural action.

Proof. We compare the current top point with an allocation that leaves at least $K/2$ bins with U -contacts free. Since such bins have slack at most $5C$ at a stall, the difference has ratio greater than one half. The final paragraph combines this with the step bound (4.20).

Augmenting paths reach U -contacts. Let $i_* \leq g_* - 1$ be the number of gains so far. The argument compares with the matching represented by the partition. Taking the symmetric difference with a perfect matching gives $K - i_*$ vertex-disjoint augmenting paths [2]. By (4.14), at most $2C(g_* - 1)$ of them meet a blocker.

Orient each remaining path from an unmatched endpoint. Its first bin is free and is entered at its exposed vertex. That bin has no other unmatched vertex and no crossing matching edge. At a stall there is no edge to a distinct bin. If the path exits to a row by a nonmatching edge, its next matching edge is that row's assigned edge, entering another bin at its exposed vertex. This new bin is assigned; its only possible crossing matching edge is the one just used on entry. A simple path must therefore next exit it by a nonmatching edge. Continuing this argument, the path must eventually reach U : it avoids blockers, cannot end inside an assigned bin, and cannot enter another free bin through an assigned row edge.

Every such entry uses the unique exposed vertex of its bin. Simplicity and vertex-disjointness consequently imply that the prefixes through their first U -contacts use distinct bins and rows, both

within a prefix and across different prefixes. Reassigning, in all prefixes at once, each prefix row to the preceding bin in its chain is therefore injective, and it frees the last bin of every prefix. If Z denotes all bins with a U -contact, one can thus leave simultaneously free a subset of Z of size

$$r \geq K - i_* - 2C(g_* - 1) \geq K/2. \quad (4.16)$$

The second inequality holds because $i_* \leq g_* - 1$ and, by (3.2), either $g_* = 1$ or $(2C + 1)(g_* - 1) \leq (2C + 1)K/(100C) \leq K/2$.

A comparison allocation. Scaling this conceptual assignment by L gives a candidate top allocation. The desired top column capacities are

$$L + p_d^0 - Cw_d,$$

which are nonnegative because the current top allocation and positive slack realize them. Clipping overloaded columns by removing mass gives a partial transportation flow. An assigned column initially carries L , so its removed mass is at most Cw_d ; the total removed is at most $Cw(\mathcal{D}) \leq CN$. The current top allocation proves that all rows can be filled under these capacities, so residual source-to-sink augmentations complete the clipped partial transportation flow; one can scale the dyadic data and use integral path decomposition for this existence argument. Simple augmenting paths do not leave the sink, so column sink flows do not decrease. The r initially empty specified columns receive in total at most the removed mass. The resulting allocation x' has nonnegative slacks s' and satisfies

$$s'(Z) \geq Lr - 2CN. \quad (4.17)$$

Ratio of the comparison direction. At a stall, the cache-based eligibility guarantee (3.10) of Lemma 3.5(v) gives $s_d \leq 5C$ on Z . Let the direction ξ be the difference between (x', s') and the current allocation and slack. Row sums and column equations are unchanged, so it is a circulation in the graph of the top primitive instance. Every negative individual logarithmic directional contribution has magnitude at most one, since $x', s' \geq 0$. Their total is at most $|E(\mathcal{B})| + |\mathcal{D}| \leq 4N$. The positive sum is at least

$$\frac{s'(Z)}{5C} - |Z| \geq \frac{Lr - 2CN}{5C} - N.$$

By (4.16) and $L \geq 10000CN/K$, this is greater than three times $4N$. As λ counts the absolute individual contributions, the ratio $g^\top \xi / \|\lambda \xi\|_1$ equals the difference of the positive and negative sums divided by their total; since the positive sum exceeds three times the negative one, the claimed ratio is greater than one half.

Exclusion of LOW and of indefinite pricing. Lemma 5.1(i) excludes LOW unless it explicitly reports failure or suspension. Its positive objective increase (Lemma 5.1(ii)), with the bound (4.20) on the number of successful pricing steps in Section 4.6, also excludes an unending sequence of pricing calls without another structural action. $\square$

4.6 Implementation and total phase cost

This subsection proves that a phase takes time $S^{1+O(1/a)+o_a(1)}$ on completed guarded prefixes; see (4.23). Time is counted per touched record and per notification of the primitive.

4.6.1 Data structures for bins, rows and contacts

The algorithm maintains physical vertices, bin and row identifiers, mates, assignments, incident lists, and edge classifications in dictionaries and linked lists. It maintains a set of distinct-bin contacts

and, for each bin, a list of contacts to U with its first member accessible. A refresh of a cached value changes a bin's eligibility flag without scanning its contacts.

At a contraction the algorithm retains a largest old bin's representative. It moves the smaller bins' member lists and the captive rows of the contracted set, updating their physical incidences and affected ledger records. Unchanged exterior incidence allocations retain their values and need only endpoint retargeting when necessary. Bin data are summed using $|X|$ point reads per retained ledger; the largest bin's members are not scanned. A moved old member's bin size at least doubles during its uninterrupted stay in bins. Initial membership, new expansions, and the entries of captive rows into the new bin begin new charges; members removed by deletions and later returns are bounded by Lemma 4.5. Deletions inspect only removed bins, their assigned rows, and their affected physical incidences, applying Lemma 3.4. Subcubicity therefore bounds all physical structural touches, including endpoint retargeting, by

$$O_a((S + Lg_*) \log S). \quad (4.18)$$

Assignment flips in the pair search require only abstract assignment updates. By Lemma 3.9, a successful contraction takes time proportional to its abstract arity $|X|$, a gain takes time proportional to L , both up to logarithmic factors, and a final abort takes at most the latter amount. All bin-pair work is consequently

$$O_a((S + Lg_* + L) \text{polylog } S), \quad (4.19)$$

where, as in Lemma 3.9, the constant may depend on σ through T_{rep} . Materialization uses Lemma 3.10 and is bounded by the same physical deletion work or by one final phase-boundary scan.

4.6.2 Top allocation values and the number of top pricing steps

At a structural edit the algorithm reads and resets only the affected top coordinates. The top column residual is available from the identity stated in Section 3.4

$$y_d^0 = s_d - p_d^0 + Cw_d;$$

it is therefore obtained from the stored values s_d , p_d^0 and w_d . The aggregation, redirection, restriction, and the 4δ transfers of the positivity repairs (Section 3.4) determine affected slacks exactly, preserving column equations without rounding. All final settings of a batch are computed before deletions, point changes, and insertions are submitted to the primitive. Both the old descriptors and those reset exactly then have positive arguments, and no pricing call occurs until the completed topology again satisfies its equations. Notifications arising during this service can be queued: there is no implicit motion during the service. All dirty small-argument and eligibility checks are settled before the next main action is chosen.

As stated in Section 3.4 after Lemma 3.5, the top primitive instance has nominal lifetime size SB , with an explicit check on the representation. Only topological insertions and retargetings use new incidence slots; repricing, motion, and positivity repairs on retained coordinates do not. By (4.18), this nominal bound is eventually sufficient. By (3.2) and (4.15),

$$L = O_a(S), \quad \sum_d p_d^0 = O_a(Lg_* + 1), \quad s_d \leq L + \sum_d p_d^0.$$

Thus at pricing states all top arguments lie in a range whose logarithmic width is $O_a(\log S)$, and there are only $O(S)$ coordinates. The top objective range is $O_a(S \log S)$. Each affected coordinate

at a structural edit decreases the objective by at most $O_a(\log S)$, so the cumulative structural loss is $O_a((S + Lg_*) \log^2 S)$. A small-allocation repair decreases the objective by at most $O(\delta)$ (Lemma 3.5(iii)). Each such repair adds 4δ at its destination bin (Section 3.4), and these additions total less than S^{-800} by Lemma 3.5(iv); so these decreases total $O(S^{-800})$. Each successful pricing step increases the objective by $\Omega(q^2)$. Hence their number is

$$O_a(Sq^{-2} \log^2 S). \quad (4.20)$$

Dirty-entry repairs and eligibility work take a constant number of point operations per notified or structurally touched entry. The notification bound of Lemma 5.1(iii) and (4.20) put the total top work in $S^{1+O(1/a)+o_a(1)}$.

4.6.3 Storage of the integral ledgers

Allocations are stored by physical incidence and y, p by bin in persistent point maps, for example path-copy balanced trees. An index and sum of positive live debts are maintained; deleted keys are cleared, and new bins are initialized locally. Retained levels receive their structural edits separately. At a projection from the integral parent $j - 1$, the levels are treated as follows.

- The new child j starts from the parent's roots as they were *before* a possible removal. The algorithm modifies this copy only along the obtained flow support and the deletions, including the exterior loads freed by removed rows. The new child's debt map is empty.
- The coarser levels $0, \dots, j - 1$, including the parent, are preserved through the removal: the deletion transformation of Lemma 3.4 is applied to each of them separately.
- The deeper levels $j + 1, \dots, a$ initially share the new child's roots.

Each reset thus takes work proportional to the size of the local projection, with logarithmic record overhead. Level 1 may build entire maps after its global static flow computation.

4.6.4 Time of the scheduled projections

Recall the reset periods $b_j = B^{a-j}$ of (3.11). Suppose an event selects level $j > 1$. Even if the event advances ticks by L , it crosses no b_{j-1} boundary: otherwise a coarser reset would have been selected. Its integral parent last reset to zero debt in this same b_{j-1} period interval; every crossing of such a boundary resets that parent or a coarser ledger. Its debt rises by at most C per expansion and $2L$ per gain, and never positively for another action. Since these actions advance ticks by $1, L$, respectively, the debt at the triggering event obeys

$$P \leq (C + 2)b_{j-1}. \quad (4.21)$$

The argument uses the period quotients (Section 3.5) both before and after the event, so it also applies when one event advances the tick counter past several period boundaries. In particular $P = O_a(S)$ for $j > 1$.

At most $\lfloor \tau/b_j \rfloor$ events select level j : each can be charged to a distinct crossed b_j boundary. By $b_{j-1}/b_j = B$, Lemma 4.1, and (4.21), all $j > 1$ projection costs sum to

$$\sum_{j=2}^a \frac{\tau}{b_j} ((C + 2)b_{j-1} + B) S^{O(1/a)+o_a(1)} = S^{1+O(1/a)+o_a(1)}. \quad (4.22)$$

Each of the $a - 1$ terms is at most $\tau((C + 2)B + B/b_j) S^{O(1/a)+o_a(1)}$, and $\tau = O_a(N)$, $N \leq S$ and $B \leq 2S^{1/a}$ give the right side. The small-instance padding B is included here. For level 1, the global

exact exchange computation takes time $S^{1+o(1)}$. Since $b_0 = B^a \geq S$ and $b_1 = b_0/B$, its total is at most

$$\frac{\tau B}{S} S^{1+o(1)} = S^{1+O(1/a)+o_a(1)}.$$

The bounds (4.18) and (4.19) are $S^{1+o_a(1)}$, since $Lg_* = O_a(N)$ and $L = O_a(S)$. Together with initialization, the top work, and the two reset sums just proved, the total time of a phase on completed guarded prefixes is

$$S^{1+O(1/a)+o_a(1)}. \quad (4.23)$$

4.7 Absence of guard failures and the perfect-matching theorem

Recall from Section 3.1 that the perfect-matching algorithm answers the empty graph by the empty matching and an input with an odd number of vertices negatively, since such a graph has no perfect matching; otherwise it starts with the empty matching and runs the phase construction of Sections 3 and 4.

Lemma 4.7 (The deterministic guards do not fail for large S). *For every sufficiently large fixed a and fixed reliability parameter σ with $0 \leq \sigma \leq a + 1$, all deterministic guards (Section 2) of the perfect-matching construction of Sections 3 and 4, including the checks (3.8) and the caps stated after them, are adequate, that is, none of them fails, for all sufficiently large S , on every computation path. The bounds proved in Sections 4.1–4.6 apply also to a prefix ending in a failed or interrupted attempted operation. These time and adequacy assertions hold whether or not the input graph has a perfect matching.*

Proof. The proof considers the prefix of a path through a hypothetical first deterministic interruption. It first derives the numerical bounds from the caps stated after (3.8) alone, then checks the nominal sizes and the iteration and operation caps using the bounds proved on that prefix, and concludes that no check fails for large S .

Initial checks. The elementary initial checks (3.8), such as

$$S \geq 2^{100}, \quad 100000C, \quad B \leq S^{1/20}, \quad q \geq S^{-1},$$

hold for all sufficiently large S once a is sufficiently large.

Numerical bounds from the caps. The S^{20} caps on main-loop actions and on pricing calls and the S^{100} external-operation cap suffice first to bound the additions of the positivity repairs and the local pads. A small argument increased and reset exactly does not require another increase without motion or a reset of its bounds. A large donor (Section 3.4) is chosen exactly from a bounded-degree row with sum L ; local repair does not create an unbounded chain of new donors that in turn need a small-allocation repair. Thus these numerical error estimates do not assume the sharper running-time bounds that they are used to prove.

Magnitudes of ledger data. All completed structural actions and removals preserve the deterministic ledger and matching invariants. Their effects on the debt (Lemmas 3.4 and 3.6) and the bottom certificates prove Lemma 4.5 independently of objective progress. These bounds also cover the structural action in progress, if any. For a coarse numerical bound, one has

$$L \leq 10000CS + 1, \quad g_* \leq S, \quad \text{removed weight} \leq 4Lg_* + 1,$$

and at most $N + 4Lg_* + 2$ expansions, allowing the next one as well. Every retained integral ledger has acquired at most

$$C(N + 4Lg_* + 2) + 2Lg_* \quad (4.24)$$

debt between its resets, by these counts alone. Column loads are at most $L|\mathcal{A}|$. Together with $100000C \leq S^{1/20}$, these estimates put all top data, exchange capacities, local return weights and scaled capacities, and argument upper bounds well inside S^{20} . Thus numerical adequacy does not depend on parameter-dependent rounding constants in the period bound.

Local threshold tests and nominal sizes. At a local threshold test, the current sink flow already bounds every lock-trigger expansion by (4.12), before that expansion is performed. Arguments before a step and at refreshes after motion have the required lower margins δ and $\delta/10$, and explicit edit batches use positive old and final settings. At the top, (4.18), with one extra physical-size edit, verifies the nominal lifetime bound SB at a possible first overflow. At a local test the nominal size is $8192(P+1)$; with this size in place of R , (4.21) eventually gives the condition (5.2), $R+B \leq S^{1+2/a}$. The same is true for the top nominal size.

Iteration caps. The structural and objective estimates (4.13) and (4.20) bound all completed iterations. Beyond these, at most one final Low, failed, or interrupted attempt occurs. Because $q^{-2} = O_a(S^{200/a})$, these bounds remain below S^{20} for every sufficiently large a and all sufficiently large S .

External-operation cap. Up to a hypothetical first external-operation cap, the notification bound of Lemma 5.1(iii) bounds the number of dirty coordinates in terms of the number of successful pricing steps. Client repairs add only constantly many point operations per dirty trigger, besides the structural edits, and without intervening motion a repair causes no further repair (Sections 3.4 and 4.3). Consequently, by (4.13) and (4.20) as for the top work in Section 4.6, even the prefix that reaches this cap has only $S^{1+O(1/a)+o_a(1)}$ external work, eventually smaller than S^{100} . Lemma 5.1(v) applies within these hypotheses: the deterministic checks of the primitive do not suspend for all sufficiently large S .

Uniformity over random outcomes. These arguments are uniform over successful verified states and rejected stochastic attempts. Lemma 5.1 bounds attempted operations on every path; its construction uses verified structures and capped failed attempts. The pair search likewise either returns a valid action within its time bound or terminates negatively. By the convention of Section 2, a detected stochastic failure ends the perfect-matching test and only a failed guard suspends the branch; the latter has just been excluded in the asymptotic range of the lemma. All local scans and updates, including a final pair abort or incomplete attempt, are counted by their represented work. This proves the claim. $\square$

Theorem 4.8 (Fixed-parameter subcubic perfect-matching test). *For integers $a \geq 1000$ and fixed reliability parameters $0 \leq \sigma \leq a+1$, the construction of Sections 3 and 4, run as described at the beginning of Section 4.7, defines a uniform randomized algorithm on loopless subcubic graphs with identified edge copies and $N \leq S$ vertices. Every computation halts with a checked perfect matching, a negative answer, or a suspension. It has no false positive. For each fixed input with a perfect matching, the probability of a negative answer is at most*

$$2^{-\sigma} S^{-5}. \quad (4.25)$$

This bound holds conditionally for adaptive invocations with fresh randomness and valid prerequisites.

For every sufficiently large fixed a and fixed σ with $0 \leq \sigma \leq a+1$, there is no suspension on any path at all sufficiently large S . The total time on every path is then

$$S^{1+O(1/a)+o_a(1)}, \quad (4.26)$$

where the exponent coefficients in $O(1/a)$ are absolute. All initialization, word-RAM data movement, arithmetic, random bits, and explicit witness writing are included.

Remark 4.9. The bound (4.25) concerns erroneous answers, not suspensions. Section 9 applies Theorem 4.8 to the parameter branches of its schedule.

Proof of Theorem 4.8. The proof has four stages. The number of phases times the phase bound (4.23) gives the time bound, and Lemma 4.7 makes it a bound on every computation path and excludes suspension. On an input with a perfect matching, the preserved invariants, Lemma 4.6 and the step bound (4.20) show that every phase reaches its target, and the final check excludes false positives on every input. A union bound over the randomized invocations gives (4.25), and the last stage justifies the conditioning used there.

Number of phases and time. The algorithm settles the trivial cases directly, starts with the empty matching, and runs the phase construction, as recalled at the beginning of Section 4.7. A phase starts with a matching M of deficit $K = N/2 - |M|$ (Section 3.1), and each gain increases $|M|$ by one. When $K \geq 200C$, a completed phase gains $g_* = \lfloor K/(100C) \rfloor \geq K/(200C)$ by (3.2), so it multiplies the deficit by at most $1 - 1/(200C)$. Since initially $K \leq N/2 \leq S$, the deficit is below $200C$ after $O(C \log S)$ completed phases. Below that threshold, at most $200C$ further phases are needed, each gaining at least one. As $C = a + 1$, there are $O_a(\log S)$ completed phases, and at most one final negative or suspended phase. Each completed phase has the bound (4.23) of Section 4.6, and so has the final phase, since by Lemma 4.7 the bounds of Sections 4.1–4.6 apply also to a prefix ending in a failed or interrupted attempted operation. Initialization, phase-boundary materialization, and the final edge-identifier and vertex-coverage check take near-linear time, which is within (4.26). Lemma 4.7 makes (4.26) a bound on every computation path and excludes suspension in the stated asymptotic range. On arbitrary checked sizes, the caps on main-loop actions and pricing calls stated after (3.8), the external-operation cap and the representation checks of Lemma 5.1(iv)–(v), and the caps of the randomized searches of Section 3.7 still give halting, with suspension permitted.

Correctness. Consider an input with a perfect matching. Unless the guarantee of a randomized subroutine fails or the branch suspends, valid pair searches and projections preserve the ledger invariant (Definition 3.1) and the matching invariants of the partition state (Section 3.1). Lemma 4.6 prevents a LOW negative answer before the phase target. The bound (4.20) on successful pricing steps prevents an infinite sequence of successful pricing steps, so each phase reaches its gains and the algorithm eventually constructs a perfect matching. Checking the proposed edge copies and coverage excludes false positives on every input, including inputs without a perfect matching.

Error probability. For the probability bound, invocations are counted using the caps stated after (3.8), rather than by conditioning on later noninterruption. There are fewer than S nontrivial phases, at most S^{20} main actions per phase, and only logarithmically many threshold tests per projection, including the tests that retrieve the regions used in Section 4.4. On guarded prefixes, the total deficit β of a projection (4.1) satisfies

$$\beta \leq CN + L|\mathcal{A}|,$$

where $\mathcal{A}$ is the set of rows, so the range of the binary search over thresholds in Section 4.4 has logarithmic bit length. Multiplying these counts, the total number of pair searches and primitive instances is $O(S^{21} \log S)$, less than S^{50} . Each reached pair search has the conditional error probability bound of Lemma 3.9; each reached primitive instance has the conditional failure probability bound of Lemma 5.1(v). Both are at most $2^{-\sigma} S^{-60}$. A union bound over the reached invocations gives at most $2^{-\sigma} S^{-10}$, which is smaller than (4.25).

Conditioning. The conditional bounds are applied when the invocation's input and prerequisites are fixed by its past history, and its new bits are fresh. They are not conditioned on whether the branch later suspends or wins a race in the schedule of Section 9. A deterministic suspension does

not by itself produce a negative answer. This proves (4.25) also at checked sizes where suspension remains possible, and completes the proof of the theorem. $\square$

5 Ascent steps along approximate minimum-ratio circulations

This section constructs the step primitive used by the matching algorithm (Section 3) and the local projection procedure (Section 4): Lemma 5.1 states its guarantees, and Lemma 5.2 states the dynamic cycle data structure from which it is built and which Sections 7 and 8 construct. The primitive maintains exact values implicitly, while exposing sufficiently accurate cached values of the coordinates to its client. Its progress guarantee depends on the length of the path representation returned by the cycle data structure. In particular, the motion attributed to a step or to a refresh of a cached value is measured before cancellation in the resulting circulation.

As in the almost-linear flow framework of Chen et al. [3], approximate minimum-ratio circulations serve as optimization directions. The step estimate below is proved here for the logarithmic objective and for motion measured by raw length.

5.1 Setting and the step primitive

Fix a size bound $S = 2^s$ and integer parameters $a \geq 1000$ and $\sigma \geq 0$, and recall the parameters (3.3): $B = 2^{\lceil s/a \rceil}$, $q = B^{-100}$, $\delta = S^{-1000}$, $\varepsilon = S^{-3000}$. The conventions of Section 2 on asymptotic estimates, suspension, detected failure, and guards apply. The guards of this section include $q \geq S^{-1}$.

Let a changing oriented multigraph, the *logical* multigraph of the client, have edge variables f_e on the grid $\varepsilon\mathbb{Z}$. Loops and individual parallel edge copies are allowed. Its separable objective $\Phi(f) = \sum_e \phi_e(f_e)$ has one of the following terms on each edge:

$$\log f, \quad \log(f - D_1) + \log(D_2 - f), \quad -W \log(T - f), \quad 1 \leq W \leq S^{20}, \quad (5.1)$$

where W is an integer. The endpoints D_1, D_2, T lie on the same grid as the values. The *descriptor* of an edge e is its term ϕ_e , that is, its form in (5.1) together with its endpoints and weight. Write $g_e = \phi'_e(f_e)$, and let λ_e be the sum of the absolute derivatives of the individual logarithmic terms on that edge, including their weights. Thus, for the two-sided term,

$$g_e = \frac{1}{f_e - D_1} - \frac{1}{D_2 - f_e}, \quad \lambda_e = \frac{1}{f_e - D_1} + \frac{1}{D_2 - f_e}.$$

All data and values have magnitude at most S^{20} , and the positive arguments at pricing and refresh states lie between $\delta/10$ and S^{20} . Immediately after a step an argument may leave this range by a small relative margin: it stays within relative distance $1.1q/200$ of its value before the step (Section 5.4), and the bounds of this section allow that margin. The clients in Sections 3 and 4 maintain stricter margins before each step.

The client may add, remove, or retarget edges and vertices, change a descriptor or value on one coordinate, request an exact point value, and make a pricing call. A *pricing call* asks the primitive for one step; it returns LOW, reports a detected failure, suspends, or performs an increment of f . In the last case the call is *successful*, and it is called a *pricing step*. The client's explicit value settings need not be circulations, and the values themselves need not have zero divergence. Between such edits, all increments performed by the primitive must be circulations.

For every coordinate e the primitive keeps a *cached value* $\hat{f}_e$; an *exact reset* of e sets $\hat{f}_e$ to the current value f_e . The primitive also resets coordinates itself when their values may have drifted from the cached values (Section 5.7); each such *refresh* is reported to the client by a *notification*,

and the refreshed coordinates are called *dirty*. The primitive services all pending refreshes and returns the list of dirty coordinates; the client can then process its local changes before the next pricing call, and further notifications arising in that processing are handled in the same way. The completion of this process is called *settlement*; Section 5.7 shows that it terminates.

A *nominal bound* R counts the logical vertex and edge births and endpoint insertions over one instance, including retargetings. Mere value or price changes on retained topology need not be counted as new incidences.

Lemma 5.1 (Dynamic ascent steps for separable logarithmic objectives). *Let S , a , σ , B , q , δ , and ε be as in Section 5.1. Consider an instance of the setting described there: a changing logical multigraph with values f_e , objective terms (5.1), and the client operations, cached values, refreshes, and settlement defined there. Let R be a nominal bound for the instance. Assume*

$$R + B \leq S^{1+2/a}. \quad (5.2)$$

There is a uniform implementation of the primitive with the following properties.

- (i) *A pricing call may return LOW. If some nonzero circulation d satisfies*

$$g^\top d \geq \frac{1}{2} \|\lambda d\|_1, \quad (5.3)$$

then the call does not return LOW, unless it reports a detected failure or suspends.

- (ii) *A pricing call that returns neither LOW nor a detected failure and does not suspend performs an exact increment $h \in \varepsilon \mathbb{Z}^E$ that is a circulation and satisfies*

$$\|\lambda h\|_1 \leq q/100, \quad g^\top h \geq q^2/1000. \quad (5.4)$$

It leaves all logarithmic arguments positive and increases Φ by $\Omega(q^2)$.

- (iii) *The implementation exposes cached values $\hat{f}_e$. After settlement and before the next pricing call, the descriptors are valid at the cached values and*

$$\lambda_e(\hat{f}_e) |f_e - \hat{f}_e| \leq q/100. \quad (5.5)$$

An explicit client setting of a coordinate can reset it exactly. Apart from such client settings, the number of refresh notifications is at most the number of successful pricing steps times $S^{O(1/a)+o_a(1)}$. Notifications are supplied as a list of dirty coordinates. Maintenance of the representation without a step leaves every value f_e unchanged.

- (iv) *With Q external pricing, point, and edge operations, total time, excluding the client's own work, is*

$$(R + B + Q) S^{O(1/a)+o_a(1)}. \quad (5.6)$$

The bound is on every computation path and includes initialization, arithmetic, random bits, and all representation changes. Polynomially many operations are allowed, with the guard $Q \leq S^{100}$.

- (v) *The conditional probability of detected failure in the entire instance is at most $2^{-\sigma} S^{-60}$, including when the instance is chosen adaptively. Deterministic size, precision, and representation checks may suspend, but none does so for all sufficiently large S at each sufficiently large fixed a and each fixed $\sigma \geq 0$ within the stated hypotheses.*

Lemma 5.1 is proved in Sections 5.3–5.8 from the cycle data structure of Lemma 5.2; the formal proof is at the end of Section 5.8. The mechanism is as follows. Each logical vertex is replaced

by a path of ports, which gives a graph of maximum degree three; the edges of these port paths are called connectors, and every other edge is priced from its cached value by an integer length ℓ_e and an integer gradient γ_e , see (5.11). While (5.5) holds, these prices approximate λ_e and g_e at the current values, see (5.13). A pricing call queries the cycle data structure, accepts its answer by the test (5.16), and moves along it by a dyadic multiple μ chosen by (5.18), so that the raw length of the step, measured in units of the prices, is at most $q/200$. The exact values are kept implicitly in a link-cut forest whose edge copies carry signed and absolute accumulators, see (5.20) and (5.23). The absolute accumulators detect when a value may have drifted from its cached value by the amount allowed in (5.5); the coordinate is then reset exactly and repriced, and each such refresh is assigned a disjoint part of the raw motion of the pricing steps (Lemma 5.4).

Accordingly, Section 5.3 builds the port graph and the prices and proves (5.13); Section 5.4 proves part (i) and the step bounds of part (ii); Section 5.5 gives the forest operations and their amortized time (Lemma 5.3, proved in Section 5.9); Section 5.6 keeps the exact values; Section 5.7 proves part (iii) through Lemma 5.4; and Section 5.8 proves the precision bounds, part (iv), and part (v).

5.2 A dynamic minimum-ratio cycle data structure with forest paths

The steps of Lemma 5.1 are computed with the following fully dynamic cycle data structure, which Sections 7 and 8 construct together with its bounds on raw path length and on forest recourse. Its minimum-ratio-cycle and hierarchical-routing viewpoint is related to the flat-forest cycle machinery of [4].

For a graph with positive lengths ℓ_e and signed gradients γ_e , put

$$\text{OPT} = \max_{d \neq 0, d \text{ a circulation}} \frac{|\gamma^\top d|}{\|\ell d\|_1}, \quad (5.7)$$

with value zero if there is no nonzero circulation. Recall from Section 2 that the raw length of a traversal list counts every traversal with its multiplicity, before cancellation. For a list of oriented input edges and oriented forest paths, as returned below, the raw length V is the sum of input lengths over all mapped edge traversals with repetition; a null forest traversal contributes zero.

The data structure reports each change of the forest it exposes, that is, each insertion or removal of a forest vertex (an *occurrence*) or of a forest edge, by a record; these records form its *forest logs*. Each occurrence and forest edge carries an identifier from its insertion to its removal, and this interval is its *lifetime*; the lifetime is *stable* if the identifier, the endpoints, and the mapping stay unchanged during it. A *transaction* is the batch of forest changes reported for one edit; it is *deletion-first* and *old-to-final* if all removals from the old forest are reported first, while the old mappings and input data are still available, and the insertions then produce the final forest. Section 8 constructs these records (see the opening paragraph of Section 8.5).

Lemma 5.2 (Dynamic approximate minimum-ratio cycles with forest representations). *Let S , a , and σ be as in Section 5.1, let OPT be defined by (5.7), and let raw length V , forest logs, stable lifetimes, and transactions be as defined after (5.7). Let $M = 2^u$, where u is divisible by $2a^2$, and suppose*

$$S^{1/a} \leq M \leq S^2. \quad (5.8)$$

Consider a dynamic undirected graph with reference orientations and fixed vertex slots, of total current size at most M and maximum degree at most three. It has positive integral lengths ℓ_e and integral signed gradients γ_e , both of magnitude at most M^{1200a} . Insertions and deletions are arbitrary; a price replacement is a deletion followed by an insertion. There is a fully dynamic data structure for this graph with the following properties.

- (i) *It exposes a forest. Every forest vertex maps to an input vertex slot; every forest edge either maps to a current input edge with specified endpoint correspondence, or is null and joins equal images. The multiplicity over each input slot is at most $M^{10/a}/4$.*
- (ii) *A query returns a list of oriented single input edges and oriented forest paths, the latter specified by endpoints. Each piece is used with unit multiplicity, with repeated pieces listed separately. Their signed projection to the input graph is a circulation c . If $\text{OPT} > 0$, the answer has raw length $V > 0$ and*

$$\frac{|\gamma^T c|}{V} \geq M^{-10/a} \text{OPT}. \quad (5.9)$$

An empty or missing answer is permitted when $\text{OPT} = 0$. There are $M^{O(1/a)+o_a(1)}$ pieces in an answer.

- (iii) *Initialization costs $M^{1+O(1/a)+o_a(1)}$. Amortized edit and query costs, including the cost of writing the forest logs that the structure exposes, are $M^{O(1/a)+o_a(1)}$, with bounds on every path.*
- (iv) *Retained forest occurrences and edges have stable lifetimes. Endpoint or mapping changes are replacements, and removals are exposed before the old mappings or input data are invalidated. Transactions permit deletion-first old-to-final updates with no intervening path query.*
- (v) *The bounds apply over the polynomial lifespans required by Lemma 5.1 (they are quantified in the stage Guards and checks of the proof of Lemma 8.10), and against adaptive clients. For the given σ , the reliability of the data structure can be chosen so that detected failure in an entire primitive instance has conditional probability at most $2^{-\sigma} S^{-60}$. A state with a reported failure has no quality requirement. Deterministic prerequisite checks may suspend, but none does so for all sufficiently large S at each sufficiently large fixed a and each fixed $\sigma \geq 0$ within the stated hypotheses.*

5.3 Degree reduction and integer prices

The first step replaces the logical multigraph by a graph of maximum degree three that represents the same circulations. For each logical edge insertion, create a fresh port at each endpoint; a loop receives two distinct ports. The ports belonging to one logical vertex lifetime form a path, extended on further endpoint insertions. Retargeting uses fresh ports. A retired port may remain on its path, and a retired vertex may leave a wholly inactive path after all its real incidences have disappeared. Distinct vertex lifetimes have distinct paths. Mere value or price changes retain the ports. The edges of these paths are *connectors*; the edges coming from logical edges are called *real*. Give every connector length one and gradient zero. The resulting port graph has maximum degree three and linear lifetime size in R .

Choose M by rounding $16(R+B)$ upward to a power of two whose exponent is divisible by $2a^2$; this multiplies $16(R+B)$ by less than 2^{2a^2+1} . Reserve $M/2$ vertex slots; the ports and any isolated logical-vertex representatives fit these slots, and slots plus edges fit the size bound M of Lemma 5.2. These counts and (5.8) are checked. They hold for all sufficiently large S by (5.2). Identifiers can be renumbered using dictionaries.

Circulations correspond in both directions. Summing conservation over one port path cancels its connectors and recovers conservation at the logical vertex. Conversely, a logical circulation d lifts to the port graph by routing its incidence imbalances along each path. The total absolute connector coefficient is at most

$$2M \sum_{e \text{ real}} |d_e|. \quad (5.10)$$

Indeed each real incidence contributes its coefficient to at most M connector positions, and each real edge has two incidences. Connectors carry no stored values; they represent conservation of increments only. In the other direction, a port-graph circulation gives a logical increment by deleting its connector coordinates.

The second step prices the real edges from their cached values. Set $H = S^{100}$ and price real edges by

$$\ell_e = \lceil H\lambda_e(\hat{f}_e) \rceil, \quad \gamma_e = \text{round}(Hg_e(\hat{f}_e)). \quad (5.11)$$

The sum or difference of at most two reciprocals of positive dyadic arguments, with the allowed integer weight, can be priced by exact bounded-length integer quotient arithmetic, so pricing is carried out in exact rational arithmetic. At exact resets,

$$S^{-20} \leq \lambda_e(\hat{f}_e) \leq S^{1022}. \quad (5.12)$$

This upper estimate also holds at the arguments reached immediately after a step. Since $S \leq M^a$, these prices fit the range M^{1200a} of Lemma 5.2. With the scale $H = S^{100}$, the lower bound in (5.12) gives $\ell_e \geq S^{80}$.

Suppose that the cached-value estimate (5.5) holds at a pricing call; Lemma 5.4 shows that it holds after every settlement. Put $\tau = q/100$. An individual log argument r with weight $W \geq 1$ changes by a relative amount at most $\tau/W \leq \tau$ between an exact reset and a pricing call. Its derivative magnitude changes by at most $\tau/(1-\tau)$ times its magnitude at the exact reset. Summing these errors term by term remains valid when the signed derivatives nearly cancel. Together with rounding error at most $1/(2H)$ and $\ell_e \geq S^{80}$, this gives, at every pricing call,

$$\left| g_e(f_e) - \frac{\gamma_e}{H} \right| \leq \frac{q}{10} \frac{\ell_e}{H}, \quad .9 \frac{\ell_e}{H} \leq \lambda_e(f_e) \leq 1.1 \frac{\ell_e}{H}. \quad (5.13)$$

By the guard $q \geq S^{-1}$ and $\ell_e \geq S^{80}$, the rounding error $1/(2H)$ is smaller than $(q/10)\ell_e/H$ by a factor at least $S^{79}/5$, for every coordinate. These inequalities are verified without reading the current vector: cached values are checked locally at their exact resets, and the displacement bound of the stage *Accuracy of the cached values* in the proof of Lemma 5.4 supplies the estimate between resets.

5.4 The acceptance test and the step

A direction satisfying (5.3) keeps a constant ratio after lifting and pricing. If (5.3) holds, its lift has ratio at least $1/4$ under the prices (5.11). Since dividing all prices by H does not change ratios, it suffices to compare γ/H and ℓ/H . By the first inequality of (5.13), the numerator in units of H is at least

$$\frac{1}{2} \|\lambda d\|_1 - \frac{q}{10} \sum_{e \text{ real}} \frac{\ell_e}{H} |d_e|, \quad (5.14)$$

whereas the second inequality of (5.13) gives $\sum_{e \text{ real}} (\ell_e/H) |d_e| \leq \|\lambda d\|_1 / 0.9$ for the real length sum. Hence the numerator is at least $(1/2 - q/9) \|\lambda d\|_1$. By (5.10), the additional connector length is at most $2M \sum_{e \text{ real}} |d_e|$, which is at most $2S^{-78}$ times the real length, since $2M \leq 2S^2$ while every real length is at least S^{80} . The ratio is therefore at least $0.9(1/2 - q/9)/(1 + 2S^{-78})$, and the bound $1/4$ follows for sufficiently large checked sizes. We also check

$$M^{-10/a}/4 > 2q. \quad (5.15)$$

It holds for all sufficiently large S : $M \leq S^2$, whereas $q = 2^{-100\lceil s/a \rceil} \leq S^{-100/a}$.

Compute the exact gradient sum and raw length of an answer of the cycle data structure using the path aggregates of Section 5.5. Accept it precisely when

$$V > 0, \quad |\gamma^\top c| \geq 2qV; \quad (5.16)$$

otherwise return LOW. This test cannot miss (5.3) unless the cycle data structure reports failure or a check suspends: if (5.3) holds, then $\text{OPT} \geq 1/4$ by the preceding paragraph, so by (5.9) and (5.15) the answer has $V > 0$ and $|\gamma^\top c|/V \geq M^{-10/a}/4 > 2q$.

An accepted answer is turned into a step as follows. Reverse every piece of an accepted answer if necessary so that $\gamma^\top c > 0$, and delete its connector coordinates from the logical increment. If N_e counts every real traversal of e in its pieces, then $|c_e| \leq N_e$. In particular, (5.13) and (5.16) imply

$$g^\top c \geq qV/H, \quad \sum_{e \text{ real}} \lambda_e N_e \leq 1.1V/H. \quad (5.17)$$

The first bound even holds with $1.9qV/H$ in place of qV/H . With c now oriented for ascent, choose a positive power of two μ with

$$q/400 \leq \mu V/H \leq q/200. \quad (5.18)$$

The interval has ratio two and therefore contains such a power. Its upper end bounds the motion of the step, and its lower end bounds the gain. Applying μ along all pieces gives $h = \mu c$, with

$$g^\top h \geq q^2/400, \quad \|\lambda h\|_1 \leq 1.1q/200. \quad (5.19)$$

These are stronger than (5.4). Moreover, the second inequality of (5.17) bounds the sum of the absolute partial motions over the entire list of traversals, counted before cancellation. Every individual argument consequently stays within relative distance $1.1q/200$ of its starting value, and within that distance divided by its weight for a weighted term. All arguments remain positive. The whole step is applied before repricing or servicing client changes; individual pieces need not themselves be circulations. The grid compatibility of μ is verified in Section 5.8, see (5.25).

5.5 Oriented forest operations

We implement path sums, path increments, and threshold detection with link-cut trees [31]. Signed path updates, separate absolute accumulation, and length-weighted threshold reporting are the dynamic-tree operations in [4, Lemma 2.7]. Its flow-maintenance construction also transfers implicit forest contributions into explicit edge values when mapped edges change [4, Theorem 3.6]. For Lemma 5.1, the signed values and the absolute motion stored on forest copies must remain correct when dormant copy slots (unassigned copy slots of the forest of Section 8) are reused and when coordinates are repriced in transactions. This subsection gives the augmented forest; Sections 5.6 and 5.7 give the copy transfers and the order of resets.

Subdivide each forest edge by an edge node. The other nodes are occurrence vertices, and represented roots are always occurrence vertices. A forest edge mapping to an input edge is a *copy* of that edge, and a copy of a real edge is a *real-variable copy*. On an edge node c mapping to an oriented input edge, let $s_c = +1$ when its represented-parent side is its tail and $s_c = -1$ when that side is its head. A null edge may receive either reference orientation. A real-variable copy stores two different accumulations:

- h_c is signed accumulation since its creation, in the fixed orientation of the mapped input edge;
- A_c is absolute accumulation since the coordinate's most recent exact reset or the copy's creation, whichever occurred later.

After everting an occurrence vertex v and exposing the path to w , the signs s_c describe traversal from v to w . A path increment uses

$$h_c \leftarrow h_c + \mu s_c, \quad A_c \leftarrow A_c + |\mu| \quad (5.20)$$

on real-variable copies.

Preferred-path aggregates store the signed sum $\sum_c s_c \gamma_c$ and the raw sum $\sum_c \ell_c$ on mapped nonnull edges. They also store a witnessed minimum of $\theta_c - A_c$ over enabled real-variable copies, that is, the minimum together with a node attaining it; the thresholds are specified in Section 5.7. Nodes other than enabled real-variable copies are omitted from this minimum. Disabled real-variable copies still receive the updates in (5.20).

Evert flips signs on edge nodes of the reversed exposed path, but does not negate their stored h_c . The reversal, signed-addition, and absolute-addition tags have the constant-size algebra

$$(s_c, h_c, A_c) \mapsto (rs_c, h_c + \alpha s_c, A_c + \beta), \quad r \in \{-1, +1\}, \quad (5.21)$$

together with reversal of path order when appropriate. Applying one such transformation and then another composes their signed offsets as $\alpha_1 + r_1 \alpha_2$. Thus a later reroot does not retroactively negate an earlier increment of f . The gradient aggregate changes sign on reversal, the raw length does not, and the enabled minimum shifts by the absolute-addition tag with its witness retained. Tags are pushed before modifying children or point fields. Each aggregate is taken over nodes of a single preferred path.

Access at a node does not change any represented-parent direction; it only changes which represented edges are preferred. Thus it does not alter the signs s_c . Point accesses without evert recover exact individual data. To insert an edge node, link it as a new child on its tail side, initializing sign $+1$; evert the head endpoint in its separate component and link that endpoint below the edge node. To delete an edge node, inspect its sign by point access, cut its child endpoint from its parent, and then cut the edge node from its parent. Edge nodes are never everted. Forest transactions give deletions before invalidation and insertions into the final forest, so every link joins different components.

Lemma 5.3 (Amortized time of the augmented link-cut forest). *On a forest of at most N_f subdivided nodes, the link-cut operations of Section 5.5, with the fields s_c, h_c, A_c and the preferred-path aggregates defined there, use $O(\log N_f)$ amortized basic tree operations per link, cut, evert, expose, point operation, or lazy path update (5.20) with the tags (5.21). Reporting and disabling a witnessed threshold violation costs the same amortized amount per reported node. Each basic tree operation uses a constant number of operations on the stored fields.*

The proof is the standard amortized analysis of link-cut trees [31, 32], checked for the augmented fields, for evert, and for recycling of a node pool; it is given in Section 5.9.

5.6 Preserving exact values under forest changes

Set

$$D = M^{10/a}. \quad (5.22)$$

There are at most D current forest copies of any input edge. Indeed, by Lemma 5.2(i) the two endpoint occurrence sets have at most $D/2$ vertices together, and the copies between them form a forest, which has fewer than $D/2$ edges. For a loop the same argument uses just one occurrence set.

Maintain a list of current copies for each input edge. For each real coordinate keep a signed base, with invariant

$$f_e = \text{base}_e + \sum_{c \text{ a current copy of } e} h_c. \quad (5.23)$$

An explicit single-edge traversal changes the base. Before deleting a copy, read its h_c and flush it into the base; a new copy starts with $h_c = 0$. This procedure is used also on price replacements and retargetings, with old mapped data retained until all removals have been processed. A client value setting to f_e^{new} sets the base to $f_e^{\text{new}} - \sum_c h_c$. Consequently a point read, point setting, or sweep of all copies costs $O(D)$ tree point operations, plus dictionary work.

The invariant is independent of the copy's present role in the cycle data structure. In particular, the construction in Section 8 of the *public forest*, the forest that Lemma 5.2 exposes, may reassign a dormant copy slot to a different child occurrence at the same label. The internal mapped edges of a reassigned slot retain their lifetimes and input images, so their accumulators h_c keep their values. A previous increment need not be a circulation in the current representation of the forest: the increment of f that it performed is recorded by (5.23). Every *new* step is a circulation in the current mapped graph. Thus representation maintenance alone neither changes a value nor loses an earlier increment.

5.7 Drift detection and the number of refreshes

The absolute counters detect when a value f_e may have drifted from its cached value $\hat{f}_e$ by the amount allowed in (5.5). At an exact reset of edge e , reset all its current copy counters A_c and one additional counter, the *bucket*. This bucket stores absolute motion of explicit single-edge traversals and absolute counts flushed from copies that subsequently die. Choose a power-of-two threshold θ_e with

$$\frac{qH}{800D\ell_e} \leq \theta_e \leq \frac{qH}{400D\ell_e}. \quad (5.24)$$

The interval has ratio two, so it contains such a power. Its upper end gives the accuracy (5.5), its lower end bounds the number of refreshes, and the power of two keeps θ_e on the grid (see Lemma 5.4 and Section 5.8). Each copy and the bucket use this same threshold. A new copy starts with $A_c = 0$ and the current threshold. Copy death transfers A_c to the bucket before removal, as well as transferring h_c to the base.

On an updated path, report every enabled copy reaching threshold using the witnessed minimum, and disable it pending service. A disabled copy continues to accumulate both signed and absolute motion. The whole circulation update is completed before any client change or reprice is serviced. An explicit traversal or a bucket flush is tested directly. Each threshold event sets a pending flag on the *variable* f_e of the copy; several events on one variable produce a single flag, and the flag survives deletion of the copy that caused it.

A pending variable is serviced by an exact point read, an exact reset and repricing, and a notification to the client, in the following order: first reset every current absolute counter and the bucket, reenable all current copies, and clear the old pending flag; then process the replacement in the cycle data structure and its forest logs. The current cached value, prices, and thresholds are installed consistently with this reset, which involves no motion. Its own old-copy removals therefore flush zero absolute mass for that coordinate. Other coordinates flagged by those logs are queued for later service, after the current transaction finishes. Explicit client settings and descriptor changes reset the coordinate exactly in the same way and either service or subsume an old pending flag. Stale queue entries are skipped or coalesced.

All old mapped edge data remain available through removals. New copies receive their current mapped prices and thresholds before a path can be used again. Once all pending flags are serviced, settlement proceeds as described in Section 5.1. No motion occurs during these transactions.

For a real coordinate e and an interval of time, the *raw priced motion* is ℓ_e/H times the sum of the absolute increments of f_e over all traversals during that interval, counted before cancellation. By (5.18), a pricing step creates raw priced motion at most $\mu V/H \leq q/200$ in total.

Lemma 5.4 (Accuracy of cached values and number of refreshes). *For the implementation described in Sections 5.3–5.7, with settlement as defined in Section 5.1 and with the refresh reports and copy-disable events of Section 5.7, the following holds. After settlement, (5.5) holds. The total number of refresh reports of the primitive and copy-disable events is $O(D)$ times the number of successful pricing steps, apart from explicit client operations. This holds for every sequence of forest changes (including reuse of dormant copy slots) and of exact resets by the client.*

Proof. The absolute counters bound the displacement from the cached value, which gives the accuracy. For the count, each refresh report and each disable event is assigned a set of raw priced motion, these sets are disjoint within each kind, a report is assigned at least $q/(800D)$, and pricing steps create at most $q/200$ of raw priced motion each. The last stage shows that settlement terminates.

Accuracy of the cached values. The signed displacement since the most recent exact reset is bounded by the sum of current absolute copy counters and the bucket. This is true initially, preserved by path updates and explicit traversals, and preserved by transfer on copy death. Dormant retained copies keep their counters. After settlement there are at most D copies and one bucket, each below threshold. Since $\lambda_e(\hat{f}_e) \leq \ell_e/H$, we obtain

$$\lambda_e(\hat{f}_e)|f_e - \hat{f}_e| \leq \frac{\ell_e}{H}(D+1)\theta_e \leq \frac{q(D+1)}{400D} \leq q/200.$$

This is stronger than (5.5).

Refresh reports. Partition the raw priced motion of each real coordinate into the intervals between its successive exact resets. Its price is fixed on such an interval. By the lower end of (5.24), a refresh report of the primitive is assigned raw priced motion of that interval of total at least

$$\theta_e \ell_e / H \geq q/(800D),$$

since the report requires a counter to reach θ_e . The assigned sets are disjoint for one coordinate, and different coordinates are assigned different real-edge motion. Absolute mass moves between copies and the bucket without duplication. It is reset before the forest logs of a price replacement are processed, so motion counted at the old price is not assigned again at the new price. An exact reset by the client may discard motion not yet assigned, but it creates no new motion; it is counted as an external operation. Since pricing steps create at most $q/200$ of raw priced motion each, while each report is assigned at least $q/(800D)$, the reports not caused by explicit client operations number $O(D)$ times the number of pricing steps.

Disable events. Similarly, a disable event is assigned the motion of its copy since creation or its latest reset. A copy is disabled at most once during that uninterrupted accumulation; reenabling at an exact reset resets it. These assigned sets are disjoint among disable events. Counting both reports and disable events only introduces a constant factor. This gives the claimed $O(D)$ amortized bound.

Termination of settlement. A no-motion refresh cascade is finite. A queued variable can use its extant absolute mass once; after its reset, all its absolute locations are zero and remain so until new

motion. Its own replacement logs cannot recreate that mass. Other variables may expose previously accumulated mass, but their reports are assigned the disjoint sets just described. Pending flags are per variable, so copy deletion loses neither the required service nor its assigned motion. This proves both termination of settlement and the notification bound. $\square$

5.8 Precision, running time, and the proof of Lemma 5.1

All quantities maintained by the primitive can be represented exactly with multiword arithmetic. The only nonintegral input values and increments are on the common dyadic grid. In particular, a positive logarithmic argument is $k\varepsilon$ for an integer $k > 0$ with $O(\log S)$ bits, since the argument is at most S^{20} and $\varepsilon = S^{-3000}$. Its reciprocal is ε^{-1}/k . Each price uses at most two such fractions and one bounded integer weight, so their products, sums, and quotient operations still have $O(\log S)$ -bit inputs and outputs. Rational comparisons of scores and scale choices likewise use bounded integer operations.

The implementation checks that a returned representation has at most S^{20} traversals in total, computing this bound from its piece count and the forest-size bound without expanding its paths. This check succeeds for all sufficiently large S at each sufficiently large fixed a , since $M \leq S^2$ and both piece count and occurrence overhead have exponent $O(1/a) + o_a(1)$. Each traversal has length at most $HS^{1022} + 1$ by (5.12) and (5.11), and (5.18) gives $\mu \geq (q/400)H/V$ with $q \geq S^{-1}$; hence

$$V/H \leq S^{1044}, \quad |\mu| \geq S^{-1045}/400. \quad (5.25)$$

For sufficiently large checked S , a power of two satisfying (5.18) is therefore an exact multiple of $\varepsilon = S^{-3000}$. Each piece has integral signed traversal coefficients, so every partial update and the final circulation are on the grid, with no rounding of conservation equations. Thresholds fit the same grid: even using $D \leq S^2$ gives $\theta_e \geq S^{-1025}/800$. These inequalities can be checked directly, with suspension at inadequate sizes.

Since $V \geq 1$, step scalars have magnitude at most S^{100} . There are at most $Q \leq S^{100}$ pricing steps, each with at most S^{20} raw traversals. Thus their total unweighted absolute motion is at most S^{220} . Transfers between copies and bases or buckets do not duplicate this motion; exact resets reset absolute counters, and client settings replace values of magnitude at most S^{20} . Signed and absolute tags, sums, buckets, and bases consequently have polynomial magnitudes on the common grid. Their bit lengths are $O_a(\log S)$; exact pricing and key operations add only logarithmic-power overhead. The arithmetic of the cycle data structure, including its static flow calls, is accounted for in Section 8. As in the uniform implementation of Section 9, larger logical integers in branches with varying parameters are digit arrays whose construction and operations are counted incrementally.

Reserve and recycle a pool of $N_f = O(M^{1+10/a})$ subdivided forest nodes. The initial pool and forest cost fit $M^{1+O(1/a)+o_a(1)}$. By Lemma 5.3, processing each exposed forest-log record or path piece costs logarithmic amortized overhead. A coordinate read, reset, or price change uses at most $O(D)$ point operations for its copy list. By Lemma 5.4, pricing steps cause only $O(D)$ reports and disables amortized. Repricing those reported coordinates invokes only that many replacements in the cycle data structure, each with the cost and recourse of Lemma 5.2. Thus the total of the work of the cycle data structure, forest-log processing, path operations, copy sweeps, and threshold service is

$$(M + Q)M^{O(1/a)+o_a(1)}. \quad (5.26)$$

By the choice of M in Section 5.3, $M < 2^{2a^2+1} \cdot 16(R + B)$, and $R + B \leq S^{1+2/a}$ by (5.2); hence $M + Q$ is at most a constant multiple of $R + B + Q$ for fixed a , and $M^{O(1/a)+o_a(1)} \leq S^{O(1/a)+o_a(1)}$. Thus (5.26) gives (5.6). A pricing call that returns LOW has the same query time, and explicit changes and all notifications are settled before the next query.

Proof of Lemma 5.1. The proof assembles the preceding subsections: the cycle data structure supplies directions, the accuracy of the prices gives parts (i) and (ii), Lemma 5.4 gives part (iii), the time accounting above gives part (iv), and the checks and the reliability of Lemma 5.2 give part (v).

Application of Lemma 5.2. The implementation runs the cycle data structure of Lemma 5.2 on the port graph of Section 5.3 with the prices (5.11); the hypotheses of that lemma (the choice of M , (5.8), maximum degree three, size at most M , and prices in the range M^{1200a}) are ensured in Section 5.3. Exact point operations and forest updates preserve (5.23) (Section 5.6). The refresh procedure of Section 5.7 proves (5.5) by Lemma 5.4, and therefore validates (5.13) before each pricing call.

Part (i). The lift (5.10), the ratio bound derived from (5.14), the quality margin (5.15), and the acceptance test (5.16) prove part (i), as shown in Section 5.4.

Part (ii). By (5.17) and (5.18), the step satisfies (5.19), which is stronger than (5.4). The bound on partial motion given by the second inequality of (5.17) keeps every argument positive (Section 5.4), and the grid check after (5.25) shows that h is an exact circulation in $\varepsilon\mathbb{Z}^E$.

Increase of Φ . The objective conclusion in part (ii) follows from its two inequalities once positivity is established. The sum of absolute relative argument changes is at most $\|\lambda h\|_1$. For a positive logarithm, $\log(1+z) \geq z - z^2$ when $|z| \leq q/100$; the sum of these squared relative changes is at most $q^2/10000$. A negative logarithm in (5.1) is convex and lies above its supporting tangent. Hence the total increase is at least $q^2/1000 - q^2/10000$.

Part (iii). Lemma 5.4 proves part (iii), with $O(D) \leq S^{O(1/a)+o_a(1)}$ by (5.22) and $M \leq S^2$.

Part (iv). The time accounting before this proof gives (5.26) and hence (5.6).

Part (v). Failures of the cycle data structure are propagated. By Lemma 5.2(v), detected failure of the cycle data structure in the whole primitive instance has conditional probability at most $2^{-\sigma}S^{-60}$; the remaining operations are deterministic. Representation, exact-reset-range, and precision prerequisites are checked on their local inputs, with counts maintained incrementally. During a whole step, the bound on raw motion proved in Section 5.4 supplies the required relative margins. The deterministic checks of this section include $q \geq S^{-1}$ (Section 5.1); the slot and edge counts and (5.8) (Section 5.3); (5.15) (Section 5.4); and the traversal bound, (5.25), the grid multiple, and the threshold bound (this subsection); together with the guard $Q \leq S^{100}$ of part (iv). The guards on operation and record counts are checked on prefixes of the computation path, and the preceding bounds hold up to and including the first operation, if any, at which one of these guards is reached. For each sufficiently large fixed a and each fixed $\sigma \geq 0$, these bounds fall below the polynomial thresholds of the guards, and the size and quality inequalities hold for all sufficiently large S . Together with the guarantees of the cycle data structure on guarded prefixes in Section 8, this proves the suspension clause of part (v) on every computation path, whether or not a stochastic routine ultimately reports failure. Such a failure is propagated promptly and is reported as a detected failure, not as a deterministic suspension. $\square$

5.9 Amortized analysis of the augmented link-cut forest

Proof of Lemma 5.3. The proof follows the standard analysis of link-cut trees: a splay potential bounds the rotations of an access, an additional count of heavy edges bounds the number of preferred-path switches, and evert, cut, and link increase both potentials by at most $O(\log N_f)$; the tags (5.21) add only bounded work per pointer or aggregate change.

Implementation. Use auxiliary splay trees in down-path order for preferred paths [32, Section 6]. Their roots keep path-parent pointers to the represented parents of their path tops. Access at v starts at v and repeatedly splays the current node y , replaces its right subtree by the path exposed so far, and advances to its previous path parent. The demoted path receives path parent y . A final

splay of v exposes the represented-root-to- v path. Evert reverses that path. A cut from parent accesses its endpoint and detaches the left ancestor subtree. A link accesses its endpoints and installs a path-parent pointer for a represented root. The stated tags require only bounded work during each pointer or aggregate change.

Rotations. For the amortization, give a node weight one plus the sum of represented subtree sizes of its nonpreferred children. Let $s(x)$ be its auxiliary subtree weight and $r(x) = \log_2 s(x)$, and use $\Psi = \sum_x r(x)$. Node weights stay fixed during a splay. We recall the rotation inequalities of the access lemma [32, Lemma 1]. In a double rotation with node x , parent y , and grandparent z , we have $r'(x) = r(z)$ and $r(y) \geq r(x)$. For zig-zag, $r'(y) + r'(z) \leq 2r'(x) - 2$ by disjoint child weights. For zig-zig, $r(x) + r'(z) \leq 2r'(x) - 2$ by the disjoint old- x and new- z subtrees, and $r'(y) \leq r'(x)$. Thus two rotations plus their change in potential cost at most $3(r'(x) - r(x))$; a final single rotation adds at most one.

Preferred-path switches. At a preferred-path switch, the total weight of a detached or attached auxiliary path equals the represented subtree size of the demoted or promoted child. The change in the splayed root's own weight compensates exactly, so the switch does not change Ψ . Along an upward access, the next node's initial auxiliary weight contains the preceding exposed path through its dashed-child weight. The rank differences therefore telescope to $O(\log N_f)$, apart from a constant per access iteration.

Number of access iterations. To bound the number of those iterations, call a represented edge heavy if its child's subtree has more than half its parent's size, and add a sufficiently scaled count of dashed heavy edges to the potential. A represented root path has at most $\log_2 N_f$ light edges. Access promotes every dashed edge it traverses. A heavy off-path edge is demoted only opposite a light preferred child on the accessed path, or at its endpoint. The iteration count plus the change in this additional potential is consequently $O(\log N_f)$.

Evert, cut, and link. It remains to check represented-tree changes. After access, write the weight of a path vertex as one plus the sizes of its off-path dashed subtrees. These weights are unchanged by reversing the complete exposed path. Recursive left-right swapping in its auxiliary splay tree preserves each auxiliary subtree's vertex set and total weight, hence preserves Ψ . If an off-path child becomes heavy after reversal, the next edge down the reversed path must be light, or the vertex is the path endpoint. There are only $O(\log N_f)$ such events. Cutting an accessed parent edge similarly leaves the path weights unchanged, does not increase Ψ , and charges newly heavy off-path children to light edges of the shortened path. Linking a root below an accessed vertex increases only that auxiliary root's weight sum; its rank grows by at most $\log N_f$. Existing off-path children of its ancestors cannot newly become heavy as ancestor subtrees grow, and the new dashed edge contributes at most one. All potentials are nonnegative and the initial singleton state has zero potential. This proves the bound, also under recycling of a fixed node pool.

Threshold reporting. For threshold reporting, expose the updated path, read a witnessed nonpositive minimum, access that node, and disable it at a point. Repeating these operations costs the claimed amount per disable. $\square$

6 A resource-preserving static routing sparsifier

The recursive cycle construction of Sections 7 and 8 needs a sparse set of representative edges through which all edges of an auxiliary graph can be routed. Its application also associates resources with the original edges: the load of a resource counts every traversal of a representative edge that carries it, in either direction, and repeated traversals count with their multiplicity. This section proves Lemma 6.1, which gives a static routing sparsifier preserving these loads: a sparse subset of

the edges together with a short walk in it for every edge. The bound on the size of this subset is a multiple of the number of incident vertex labels and holds even when the number of demands is much larger.

Throughout this section, a is an integer, $u > 0$, and

$$S = 2^s \geq 2, \quad M = 2^u, \quad S^{1/a} \leq M \leq S^2, \quad a \geq 10, \quad 2a^2 \mid u. \quad (6.1)$$

Define

$$b = M^{1/a^2}, \quad \Lambda = 2^{1000a^2}(1+u), \quad P_* = \Lambda^{400a^2}, \quad P_* \leq b. \quad (6.2)$$

The last inequality is a checked prerequisite. Since $2a^2$ divides u , b is an integer. Powers of Λ are logarithmic powers of the size bound S .

Lemma 6.1 (Static routing with resources). *Let S , M , a , and u be as in (6.1), and let b and P_* be given by (6.2), with $P_* \leq b$. Let F be an explicitly listed undirected multigraph with identified edge copies, and let v_+ be its number of incident vertex labels. For each edge f , an explicit list specifies resource counts $m_{xf} \in \{1, \dots, 10\}$; unlisted counts are zero. Suppose that*

$$\deg_F(v) \leq C_0, \quad \sum_{f \in F} m_{xf} \leq C_0 \quad \text{for every vertex } v \text{ and resource } x,$$

where $C_0 \geq 1$ is an integer, degrees count incidences, and both $2|F|$ and the total resource-list length are at most M^2 . Assume that identifiers and numerical input data have $(\log S)^{O_a(1)}$ bits. There is a uniform randomized algorithm that either reports detected failure or returns a set $F' \subseteq F$ and, for every $f = vw \in F$, an explicit walk Q_f from v to w in F' , such that

$$|F'| \leq P_* v_+, \quad |Q_f| \leq P_*, \quad \sum_{f \in F'} \sum_{e \text{ traversed in } Q_f} m_{xe} \leq P_* C_0 \quad \text{for every } x. \quad (6.3)$$

The inner sum counts traversal multiplicities and ignores orientation. A loop demand may be assigned the empty walk. Every returned result is verified. For each fixed reliability parameter $\sigma \geq 0$, the time on every computation path is

$$(1 + \text{input list size}) b M^{o_a(1)}, \quad (6.4)$$

and the probability of detected failure is at most $2^{-\sigma} S^{-800}$. The same statement holds conditionally when the input is determined by earlier computation and this call uses fresh bits.

Lemma 6.1 is proved in Section 6.5. Sections 6.1 to 6.4 describe one stage of the algorithm and establish its properties on good events: the stage structure and the sampling estimates (Section 6.1), the cuts and the expanding overlay (Section 6.2), routing on the overlay (Section 6.3), and the filtering and verification of a stage (Section 6.4); the proof in Section 6.5 uses these estimates to bound the failure probability of one attempt, combines the accepted stages into (6.3), and bounds the work on every computation path.

We build the representative set in stages, each routing at least half the remaining demands. An attempt samples edges according to their endpoint degrees and gives each retained edge its inverse-probability weight. Each edge incidence is represented by a distinct *stub*. A flow between two equal halves of the stubs either identifies a cut for recursion or pairs almost all the stubs by walks in the sampled graph. The sampling estimate ensures that the cuts are sparse with high probability. We defer their crossing demands to later stages and recurse on the two induced pieces with fresh samples. When almost all stubs are paired, completing the unpaired stubs with temporary fake

edges gives a perfect matching on the stubs. A piece that completes the prescribed rounds without a further cut obtains an overlay made from these matchings; the median rule (Section 6.2) makes it expanding with high probability. Each nonfake overlay edge stores its sampled-edge walk, called its *real embedding*.

We then route the demand stub pairs through this overlay. Partition its labels into groups and send each demand packet on a short random walk conditioned to reach its target's group. Additional independent walks, starting and ending within each group, supply the edges of smaller graphs on which routing continues. Recursion reaches singleton groups, and substituting the saved walks gives overlay routes with bounded length and congestion. Discard routes that touch a fake edge or an excessively long real embedding. Congestion bounds the number of lost demands, while the sampling weights bound resource use after the remaining real embeddings are substituted. The final verification checks these walks and their loads before accepting the stage.

The construction uses the following internal parameters:

$$\begin{aligned} r_m &= \Lambda^3, & K_s &= \Lambda^{230a^2}, & \chi &= \Lambda^{110a^2}, & \epsilon_0 &= \Lambda^{-100a^2}, \\ D_p &= \Lambda^{105a^2}, & H_* &= \Lambda^{70a^2}, & L_0 &= \Lambda^{200a^2}, & t &= \Lambda^{10}, & q_v &= \Lambda^3. \end{aligned} \tag{6.5}$$

They fall into four groups by role. For sampling and cuts, K_s sets the sampling scale, χ multiplies flow capacities, ϵ_0 sets the deficiency scale, and D_p caps the cut depth. For the overlay, r_m counts its rounds. For routing on the overlay, t is half the conditioned-walk length, q_v prescribes the same number of virtual jobs (Section 6.3) at each vertex, and H_* controls overlay routes. For filtering, L_0 controls the retained real embeddings.

None of the comparisons that the argument requires of these values is tight. Each comparison is stated where it is used: the bound on the number of cuts on a branch (Section 6.2); the sign-tail comparison in the proof of Lemma 6.3; the degree-ratio, conductance, and mixing comparisons in the proofs of Lemmas 6.4 and 6.5, including (6.22); the bound on the lost fraction of demands and the stage margins (6.26) (Section 6.4); and the comparison of the stage margins with P_* and the tail and counting comparisons of Section 6.5.

For an unweighted multigraph H , let $\partial_H X$ be the edge-copy cut of X , and let $\text{vol}_H(X) = \sum_{v \in X} \deg_H(v)$. Parallel copies count separately; loops count twice in volume and never in a cut. We suppress the subscript when the graph is clear. The conductance of H , defined whenever there is a cut with two positive volumes, is

$$\phi(H) = \min_{\substack{X \subseteq V(H) \\ \text{vol}_H(X) > 0, \text{vol}_H(V(H) \setminus X) > 0}} \frac{|\partial_H X|}{\min\{\text{vol}_H(X), \text{vol}_H(V(H) \setminus X)\}}.$$

6.1 Stages, pieces, and normalized sampling

Remove loop demands by giving them empty walks. Work on the remaining demands in stages. One successful stage must route at least half its input demands while using, separately for support, maximum hop count, and resource load, the bound Λ^{350a^2} in place of P_* in (6.3). All these properties, including endpoints and selected edge identities, are checked before a stage is accepted (Section 6.4). A stage gets at most

$$T = 2(\sigma + 1000 \log_2 S + 10) \tag{6.6}$$

fresh attempts (rounding up if necessary). Exhausting these attempts is a detected failure. There are at most $2u + 2$ successful stages, since $|F| \leq M^2/2$ and the residual number halves each time.

An attempt processes a recursive family of pieces. A piece is an edge list induced by the vertex side of each earlier cut; cut edges are discarded for this stage. Empty lists need no further processing.

Pieces at a fixed depth have disjoint edge lists and disjoint incident vertex sets. Process pieces by depth, using fresh samples at each depth. We cap the recursion depth at D_p and abort an attempt that would exceed the cap. Let a nonempty piece have s' edges and positive original degrees d_i . Retain each edge $e = ij$ independently with probability p_e , the smallest number of the form $2^{-\ell}$, $\ell \geq 0$, that is at least

$$\min\{1, K_s(1/d_i + 1/d_j)\}. \quad (6.7)$$

A retained edge receives integral weight $\omega_e = 1/p_e$. Before any flow call, require

$$\sum_{e \text{ retained}} \omega_e \leq 2s', \quad \#\{e \text{ retained}\} \leq 4K_s \#\{i : d_i > 0\}. \quad (6.8)$$

A violation aborts this attempt. Volumes in the following sampling estimate use the original piece degrees, not the sampled degrees.

Lemma 6.2 (Cut and resource-load estimates for edge sampling). *Let a nonempty piece with s' edges and positive original degrees d_i be sampled with the probabilities p_e of (6.7) and the weights $\omega_e = 1/p_e$. Except with probability at most $4s' \exp[-K_s/(100\chi^2)]$, every vertex set X satisfies*

$$\left| \sum_{e \in \partial X \text{ retained}} \omega_e - |\partial X| \right| \leq \frac{\min\{\text{vol}(X), \text{vol}(\bar{X})\}}{4\chi}. \quad (6.9)$$

This event implies the total-weight check in (6.8); the sample-count check fails with probability at most $\exp(-K_s \#\{i : d_i > 0\}/3)$. Conditional on the input pieces at a fixed depth, for each resource x ,

$$\sum_{\substack{e \text{ retained} \\ \text{over all pieces at this depth}}} \omega_e m_{xe} \leq 2C_0 \quad (6.10)$$

except with probability at most $\exp(-K_s/30)$.

Proof. Put $D = \text{diag}(d_i)$ on the nonisolates and set

$$A_e = D^{-1/2}(\mathbf{e}_i - \mathbf{e}_j)(\mathbf{e}_i - \mathbf{e}_j)^\top D^{-1/2}.$$

These are positive semidefinite rank-one matrices, $\|A_e\| = 1/d_i + 1/d_j$, and $\|\sum_e A_e\| \leq 2$. If ξ_e is the retention indicator, write $Y_e = (\xi_e/p_e - 1)A_e$. Deterministically retained edges contribute zero. For any other edge, $p_e \geq K_s\|A_e\|$, whence

$$\|Y_e\| \leq K_s^{-1}, \quad \mathbb{E}Y_e^2 = (p_e^{-1} - 1)A_e^2 \preceq K_s^{-1}A_e.$$

Thus the variance parameter is at most $2/K_s$. The self-adjoint matrix Bernstein inequality, applied to both signs [33, Theorem 6.1(ii)], states that independent centered self-adjoint summands of norm at most J and total variance norm at most V satisfy

$$\Pr \left\{ \left\| \sum_e Y_e \right\| \geq z \right\} \leq 2k \exp \left(-\frac{z^2}{2(V + Jz/3)} \right),$$

where k is the matrix dimension. Here $k \leq 2s'$, $J = 1/K_s$, $V \leq 2/K_s$, and $z = 1/(4\chi)$ give the claimed probability bound. Testing the quadratic form on $D^{1/2}\mathbf{1}_X$ gives additive error $\text{vol}(X)/(4\chi)$ in its cut weight. Testing the complement gives the other bound in (6.9). Testing a singleton bounds its weighted degree by $(1 + 1/(4\chi))d_i$; summing degrees proves the weight check.

Dyadic rounding gives

$$\sum_e p_e \leq 2K_s \sum_{ij=e} (1/d_i + 1/d_j) = 2K_s \#\{i : d_i > 0\}.$$

The scalar Chernoff inequality therefore proves the sample-count check with failure at most $\exp(-K_s \#\{i : d_i > 0\}/3)$. If the threshold exceeds the total possible edge count, failure is impossible.

For (6.10), condition on all lists at this depth. They are disjoint sublists of the input, so the expected sum for x is at most C_0 . A nondeterministic summand $\xi_e m_{xe}/p_e$ lies in $[0, 10C_0/K_s]$, since piece degrees are at most C_0 and $1/p_e \leq C_0/K_s$. After centering, its absolute value has the same upper bound. The sum of variances is at most $10C_0^2/K_s$, by bounding each second moment by its maximum value times its expectation. Deterministic summands have zero variance. For an upward deviation of C_0 , scalar Bernstein consequently gives

$$\Pr\{\text{sum} > 2C_0\} \leq \exp\left(-\frac{C_0^2}{2(10C_0^2/K_s + 10C_0^2/(3K_s))}\right) \leq e^{-K_s/30}.$$

□

6.2 Median cuts, integral matchings, and an expanding overlay

The median-projection and matching-overlay argument has the cut-matching structure of Khandekar, Rao, and Vazirani [20, Sections 3–3.2]. Here the vertices are incidence stubs. The analysis adds four controls to that structure: deficient flow, resource loads, exact arithmetic, and rejected attempts.

Recall that each original edge incidence has its own stub, so the piece has $N_p = 2s'$ distinct stubs. Run at most r_m rounds. Each round divides the stubs into two equal halves by the median rule specified before Lemma 6.3. Let l_i and h_i count the left and right stubs at original vertex i . Form a flow network with source capacities l_i , sink capacities h_i , and both directed arcs of capacity $\chi\omega_e$ for every retained edge. Its maximum possible value is s' .

If its deficiency $\Delta = s' - \text{maximum value}$ is at least $\epsilon_0 s'$, use the source-side vertex set X of a minimum cut to split the piece and stop its rounds. The cut formula gives

$$l(X) - h(X) = \Delta + \chi \sum_{e \in \partial X \text{ retained}} \omega_e. \quad (6.11)$$

The left side equals $h(\bar{X}) - l(\bar{X})$, so both original volumes are at least $\epsilon_0 s'$. It is also at most their minimum. On the event in Lemma 6.2,

$$|\partial X| \leq \frac{2 \min\{\text{vol}(X), \text{vol}(\bar{X})\}}{\chi}. \quad (6.12)$$

Indeed the weighted cut is at most the minimum volume divided by χ , and (6.9) adds only one quarter of that upper bound. Each child's internal edge count is at most $(1 - \epsilon_0/2)s'$, because the opposite side has volume at least $\epsilon_0 s'$. A nonempty branch thus has fewer than $1 + 4\epsilon_0^{-1} \log M < D_p$ cuts. Moreover, $\min\{\text{vol}(X), \text{vol}(\bar{X})\} \leq s'$, so summing (6.12) at each depth shows that all cuts discard at most the fraction

$$2(D_p + 1)/\chi \quad (6.13)$$

of the attempt's initial demands. The sum over the pieces at one depth uses that their edge lists are disjoint, so their edge counts sum to at most the attempt's initial demand count.

When $\Delta < \epsilon_0 s'$, extract the integral flow as individually identified pairs of left and right stubs with explicit sampled-edge walks. Pair all remaining left and right stubs arbitrarily and call these pairs *fake edges*. The result is a perfect matching on the stubs. Save this matching as one round of an overlay multigraph, together with real embeddings for its nonfake edges.

These flow calls use Lemma 2.1. Padding to $O((\chi + 1)s')$ makes every capacity polynomially bounded in the padded size, since the weight check precedes the call. Alternatively, Lemma 2.3 provides the same flow computation when local capacities are presented with longer bit strings. Paths are extracted as in Section 2, and their extraction time is bounded separately: pair incoming and outgoing flow units at each internal vertex, also pairing the units on source and sink arcs with their stubs, and trace from source units. Remaining cycles are omitted. There are at most

$$2\chi \sum_{e \text{ retained}} \omega_e \leq 4\chi s' \quad (6.14)$$

interior arc units per round. Lists of these units and their pairings therefore construct all paths within this bound plus the network size.

The *median rule* that chooses the halves is the following. For a perfect matching on the stubs, let A be its averaging matrix: on each matched pair it replaces two coordinates by their average. Start with $P_0 = I$, and after round j set $P_j = A_j P_{j-1}$, where A_j is the averaging matrix of the matching saved in round j . At the start of round j , let p_i be the rows of P_{j-1} and choose a fresh vector ζ of independent uniform signs. Order the numbers $\langle p_i, \zeta \rangle$, resolving ties by stub identifiers, and put their first and last halves on opposite sides.

The algorithm computes these projections from the saved matchings: to compute a new projection, it applies the earlier matching averages in chronological order to ζ , obtaining $P_{j-1}\zeta$. This takes $O(jN_p)$ elementary averages. All entries are dyadic rationals with denominator dividing 2^{j-1} , so exact projections, comparisons, and tie resolution use $O(r_m + \log N_p)$ bits per number.

Lemma 6.3 (Conductance of the matching overlay). *There is a median rule for which a piece completing all r_m rounds has an r_m -regular overlay of conductance at least $1/(4r_m)$, except with probability at most $4N_p^3 \exp[-r_m/(80\Lambda)]$. This guarantee holds for any choice of perfect matching across each chosen pair of halves.*

Proof. We show that the median rule specified before the lemma has the stated property. The potential Ψ defined below decays in expectation over the rounds; hence every entry of the final averaging product is at least $1/(2N_p)$ except with the stated probability; and the reverse-round walk converts this entrywise bound into conductance.

The averaging matrices A_j and their products P_j are doubly stochastic. Write $\bar{p} = \mathbf{1}/N_p$ and $\Psi = \sum_i \|p_i - \bar{p}\|_2^2$. The centered projected values $z_i = \langle p_i - \bar{p}, \zeta \rangle$ have mean zero over i , and $Z = \sum_i z_i^2$ satisfies

$$\mathbb{E}Z = \Psi, \quad \mathbb{E}Z^2 \leq 3\Psi^2.$$

For the second inequality, sign fourth moments give $\mathbb{E}\langle v, \zeta \rangle^4 \leq 3\|v\|_2^4$, and Cauchy–Schwarz bounds each cross term in Z^2 . The Paley–Zygmund inequality now gives $\Pr\{Z \geq \Psi/2\} \geq 1/12$ when $\Psi > 0$. Hoeffding’s inequality for sums of independent signs and a union bound over pairs give, with failure probability at most $2N_p^2 e^{-\Lambda/2}$,

$$|\langle p_i - p_j, \zeta \rangle|^2 \leq \Lambda \|p_i - p_j\|_2^2 \quad \text{for every } i, j.$$

Choose a median value m between the two central centered values z_i . For any pair i, j on opposite sides, $(z_i - z_j)^2 \geq (z_i - m)^2 + (z_j - m)^2$. Every stub occurs once in the matching, and the

sum of squared distances is minimized at the mean, zero. Consequently, for *every* matching across the halves,

$$\sum_{ij \text{ matched}} (z_i - z_j)^2 \geq \sum_i (z_i - m)^2 \geq Z.$$

The exact decrease under averaging is $\frac{1}{2} \sum_{ij \text{ matched}} \|p_i - p_j\|_2^2$. With conditional probability at least $1/20$ this is at least $\Psi/(4\Lambda)$. The parameter values ensure that the sign-tail failure above is smaller than $1/12 - 1/20 = 1/30$: since $N_p = 2s' \leq 2|F| \leq M^2$, $M = 2^u$, and $e^{-x} \leq 2^{-x}$, it is at most $2M^4 e^{-\Lambda/2} \leq 2^{1+4u-\Lambda/2}$, and $\Lambda \geq 2^{1000a^2}(1+u) \geq 8u+12$ makes this at most $2^{-5} < 1/30$. If $\Psi = 0$, it stays zero. Thus

$$\mathbb{E}\Psi_{r_m} \leq N_p \exp[-r_m/(80\Lambda)].$$

For this analysis one can continue with arbitrary median-crossing matchings after a piece has been cut; hence stopping a piece early does not introduce conditioning on successful mixing. If any entry of P_{r_m} is below $1/(2N_p)$, then $\Psi_{r_m} \geq 1/(4N_p^2)$. Markov's inequality proves the stated failure bound.

It remains to identify which walk this product describes. Since $P_{r_m} = A_{r_m} \cdots A_1$, its row starting at stub i describes averaging steps in the order $r_m, r_m - 1, \dots, 1$. Consider one unit of mass at each stub of a set X with $|X| \leq N_p/2$, moved by this *reverse-round* walk. All product entries are at least $1/(2N_p)$, so at least $|X|/4$ mass finishes outside X . At every intermediate time, double stochasticity keeps mass at each stub at most one. A matching edge $\{v, w\}$ at its single time step therefore carries total mass at most $(\text{mass}(v) + \text{mass}(w))/2 \leq 1$. The number of overlay edge occurrences crossing X is at least $|X|/4$. Every stub has degree r_m , which proves conductance at least $1/(4r_m)$. $\square$

6.3 Recursive routing on the overlay

The next task is to route the demand stub pairs through the overlay with bounded path length and total traversal load on each overlay edge. These two bounds control the lengths and resource loads after the saved real embeddings are substituted.

Pair the original stubs according to their demand edges and orient each pair arbitrarily, giving one packet with a specified source and final target. Both the initial sources and the final targets are distinct. At recursive calls current packet locations can coincide, but their final targets remain distinct. We route these packets through the overlay, temporarily permitting fake edges.

Every graph on which a walk runs retains distinct parallel edge copies and uses incidence degrees: a loop has two incidences. Its lazy walk holds with probability $1/2$ and otherwise chooses a uniform incident edge occurrence. Write P for its transition matrix, so that $P^h(x, y)$ is the probability that h lazy steps lead from x to y , and $P^h(x, A) = \sum_{y \in A} P^h(x, y)$. Holding incurs no traversal. An actual traversal of a loop is counted once. The stationary law is $\pi(v) = d_v / \sum_i d_i$, and $\pi_{\min} = \min_v \pi(v)$. For the lazy walk, the corresponding transition-kernel conductance is $\phi(H)/2$. We invoke conductance and mixing only on graphs with at least two vertices on which a walk runs; recursive calls on a single label are settled directly. The *degree ratio* of such a graph is its maximum degree divided by its minimum degree.

We use the standard conductance-to-mixing argument for reversible chains [30, Section 3]. We include the proof because it uses the normalization of $\phi(H)$ given before Section 6.1 and allows loops and parallel copies.

Lemma 6.4 (Mixing from conductance). *In a connected undirected multigraph with at least two vertices and conductance ϕ , the lazy walk has spectral gap at least $\phi^2/4$. Let Λ , r_m , and t be as in (6.2) and (6.5). If the graph has at most N_p labels, where $N_p = 2s'$ is the number of stubs of a piece as in Section 6.2, degree ratio at most $D_c \leq \Lambda^{1/20}$, and conductance at least $1/(4r_m)$, then for every*

$h \geq t$,

$$\frac{1}{2}\pi(y) \leq P^h(x, y) \leq 2\pi(y) \quad \text{for all } x, y. \quad (6.15)$$

Proof. Let f be a nonnegative vector supported on at most half the degree volume. Integrating the boundary sizes of the sets $\{i : f_i^2 > \xi\}$ over $\xi \geq 0$ (the co-area formula) gives

$$\sum_{ij \text{ edge}} |f_i^2 - f_j^2| \geq \phi \sum_i d_i f_i^2.$$

Cauchy–Schwarz, together with $\sum_{ij \text{ edge}} (f_i + f_j)^2 \leq 2 \sum_i d_i f_i^2$, implies

$$\sum_{ij \text{ edge}} (f_i - f_j)^2 \geq \frac{\phi^2}{2} \sum_i d_i f_i^2. \quad (6.16)$$

Both formulas remain valid with loops counted by their two incidences. Let g be an arbitrary vector of degree-weighted mean zero, and let m be a degree-weighted median of g . The positive and negative parts of $g - m$ each have support volume at most half. Their edge energies sum to at most that of g , while their squared degree norms sum to

$$\sum_i d_i (g_i - m)^2 = \sum_i d_i g_i^2 + m^2 \sum_i d_i \geq \sum_i d_i g_i^2.$$

Applying (6.16) to both parts proves the nonlazy gap bound $\phi^2/2$, by the Dirichlet quotient, and hence the lazy bound $\phi^2/4$. The nonlazy reversible operator is a contraction in the degree-weighted Euclidean norm; making it lazy places its spectrum in $[0, 1]$.

For the pointwise bound, the contraction of the lazy operator on mean-zero densities is applied to a point mass. Cauchy–Schwarz in the stationary inner product gives relative pointwise error at most

$$\frac{\exp(-h\phi^2/4)}{\sqrt{\pi(x)\pi(y)}} \leq \frac{\exp(-h\phi^2/4)}{\pi_{\min}}.$$

Here $\pi_{\min} \geq 1/(D_c N_p)$, $N_p \leq M^2$, $\phi \geq 1/(4\Lambda^3)$, and $t = \Lambda^{10}$. Thus the last expression is at most $D_c M^2 \exp(-\Lambda^4/64) < 1/2$, establishing (6.15). $\square$

The grouped recursive virtual-job construction of the next paragraph adapts the hierarchical routing idea of Ghaffari–Kuhn–Su [14, Lemma 3.1]; the recursive sampling of virtual graphs uses their group-conditioned walk construction [14, Section 3.1.2]. We prove the packet routing, congestion, and finite-computation estimates needed here, together with the sequential implementation, explicit caps, and resource-load bounds; these estimates allow nonuniform stationary distributions and count repeated traversals.

At a recursive call, a singleton label requires no routing. Otherwise divide its label set into $\min\{b, \#\text{labels}\}$ groups whose sizes differ by at most one. For each packet, draw a $2t$ -step walk from its present location conditioned to end in the group of its final target. For each vertex, also draw q_v independent *virtual jobs* with the same length, starting there and conditioned to end in that vertex’s own group. The start and endpoint of every virtual job form one undirected edge within its group; its sampled walk is saved as the edge’s embedding. Recurse on each of these virtual graphs with the packets at their newly sampled locations and with their original final targets. On return, substitute the stored embeddings, reversing them when necessary.

All job draws within a call are independent conditional on its input. A conditioned draw is implemented by ordinary walk trials, retaining the first whose endpoint is in the prescribed group. Allow at most $\Lambda^3 b$ trials per job. For a uniform choice among the d_v incident occurrences at a

vertex v , draw $\lceil \log_2 d_v \rceil$ fair bits and reject indices outside $\{0, \dots, d_v - 1\}$; allow at most Λ strings for one such choice. A cap violation aborts the attempt. These rules also cover graphs with loops and parallel edges. The group-size rule makes the maximum size after j recursive levels at most $\lceil N_p/b^j \rceil$. Hence the recursion depth is at most $2a^2$, since $b^{2a^2} = M^2 \geq N_p$.

Lemma 6.5 (Length and congestion of recursive routing on the overlay). *Let S , M , a , and u be as in (6.1), let b , Λ , and P_* be given by (6.2) with $P_* \leq b$, and let the internal parameters be given by (6.5). Conditional on an r_m -regular overlay of conductance at least $1/(4r_m)$, as in the conclusion of Lemma 6.3, the recursive routing procedure of Section 6.3 produces explicit routes for all packets with both maximum length and total traversal congestion on each overlay edge at most H_* , except with probability less than M^{-20} . All its data sizes and running times remain bounded by the caps specified in that procedure even when it fails.*

Proof. The proof is an induction over the recursive levels on a good event. On this event each child graph again has degree ratio within the allowed bound and conductance at least $1/(4r_m)$, so Lemma 6.4 applies at every level. A load bound for all jobs at one call then gives local edge loads and walk lengths at most Λ^{20} , and substitution multiplies these bounds over at most $2a^2$ levels. The last two stages collect the failure probabilities and prove the work bounds on every run.

Probability space and uncapped rejection. All probabilities in this proof are conditional on the recursively specified input of the current call. Endpoints chosen at a parent create the child graph, but at that child the new streams are fresh independent streams. This permits induction up to the first failed graph or packet invariant, even though the child graph may depend on its packet starts. The analysis treats uncapped rejection, in which a conditioned draw or a uniform choice repeats its trials without a cap, and stops when an inductive good event fails. Coupling capped rejection to uncapped rejection until the first cap justifies using the uncapped distributions in the bounds below, including the bounds on the probability of reaching a cap.

Degree and conductance invariants. The current degree ratio is allowed to be D_c , initially one, and this bound is multiplied by 20 per level. Thus

$$D_c \leq 20^{2a^2} \leq \Lambda^{1/20}.$$

The inductive conductance lower bound is $1/(4r_m)$, so Lemma 6.4 applies. A group A occupies at least $1/(2b)$ of the labels; hence $\pi(A) \geq 1/(2D_c b)$. An ordinary trial succeeds with probability at least $1/(4D_c b)$. The endpoint of a successful trial has distribution

$$\frac{P^{2t}(x, y)}{P^{2t}(x, A)} \in \left[\frac{1}{4}, 4 \right] \frac{\pi(y)}{\pi(A)} \subseteq \left[\frac{1}{4D_c |A|}, \frac{4D_c}{|A|} \right] \quad (y \in A). \quad (6.17)$$

Distinct final targets mean that at most $|A|$ packets seek group A . Packet arrivals at a specified child vertex therefore have expectation at most $4D_c$. For a sum Y of independent indicators and any $\mu \geq \mathbb{E}Y$ we use the Chernoff bounds

$$\Pr\{Y \geq R\} \leq 2^{-R} \quad (R \geq 6\mu), \quad \Pr\{Y \geq (1 + \delta)\mu\} \leq \exp\left(-\frac{\delta^2 \mu}{2 + \delta}\right) \quad (\delta > 0).$$

Since $q_v = \Lambda^3 \geq 24D_c$, the first bound shows that the arrivals at a specified child vertex are at most q_v except with probability at most 2^{-q_v} .

Exactly $q_v |A|$ virtual jobs seek A . If I_z counts their endpoints at z , then $\mathbb{E}I_z \leq 4D_c q_v$, and the second bound with $\delta = 3$ and $\mu = 4D_c q_v$ gives $I_z \leq 16D_c q_v$ for each z except with probability at most $\exp(-36D_c q_v/5) \leq e^{-q_v}$. The child degree is exactly $q_v + I_z$: a loop contributes its start incidence

and its endpoint incidence. Thus every child degree is at least q_v and at most $(1+16D_c)q_v \leq 20D_cq_v$, proving the claimed next degree ratio.

For any $X \subseteq A$ with $1 \leq |X| \leq |A|/2$, each of the $q_v|X|$ jobs starting in X leaves X with probability at least $1/(8D_c)$, by (6.17). Chernoff yields

$$\#\{\text{virtual edges from } X \text{ to } A \setminus X\} \geq \frac{q_v|X|}{16D_c} \quad (6.18)$$

except with probability $\exp[-q_v|X|/(64D_c)]$. A union bound over subsets, grouped by cardinality, bounds the failure probability for all such X by

$$\sum_{j \geq 1} N_p^j \exp[-q_v j/(64D_c)]. \quad (6.19)$$

For a nontrivial cut, let X be its smaller-cardinality side, to which (6.18) applies. Its volume is at most $20D_cq_v|X|$, and the smaller of the two cut volumes is no larger than this volume. The child conductance is at least $1/(320D_c^2) \geq 1/(4r_m)$. This establishes all graph and packet invariants required at the next level.

Expected loads at one call. We next bound loads of *all* jobs at one call before any substitution. Set $W = \sum_i d_i$ and $d_{\min} = \min_i d_i$. At most $2q_v$ jobs start at any vertex: there are q_v virtual jobs and at most q_v packets. At most $(q_v + 1)|A|$ jobs seek group A , because virtual jobs seeking A start in A and packet final targets are distinct.

Fix an oriented incidence $x \rightarrow y$ of a specific edge. At a step in the first half, the remaining suffix has at least t steps. Conditioning a job on its prescribed terminal group increases the probability of this step's traversal by at most four: the ratio is $P^h(y, A)/P^{2t}(z, A) \leq 2\pi(A)/(\pi(A)/2)$, where z is its start. Without conditioning, the aggregate initial density is at most $2q_v/\pi_{\min} = 2q_vW/d_{\min}$ relative to stationarity, and the walk preserves this bound. A stationary lazy step traverses this oriented incidence with probability $1/(2W)$. Therefore its expected conditioned load is at most

$$4 \frac{2q_vW}{d_{\min}} \frac{1}{2W} = \frac{4q_v}{d_{\min}}. \quad (6.20)$$

In a second-half step, the prefix before traversal has at least t steps. If j steps remain after this incidence and the job seeks A , its conditional traversal probability is at most

$$\frac{2^j(x) (1/(2d_x)) P^j(y, A)}{\pi(A)/2} = \frac{2P^j(y, A)}{W\pi(A)}.$$

Summing first over jobs seeking A and then over the groups gives

$$\begin{aligned} \sum_A \frac{2(q_v + 1)|A|}{W\pi(A)} P^j(y, A) &= 2(q_v + 1) \sum_A \frac{|A|}{\text{vol}(A)} P^j(y, A) \\ &\leq \frac{2(q_v + 1)}{d_{\min}} \sum_A P^j(y, A) = \frac{2(q_v + 1)}{d_{\min}}. \end{aligned} \quad (6.21)$$

This calculation cancels, group by group, the factor $1/\pi(A)$ introduced by conditioning on the terminal group. In particular it introduces neither the number of groups nor a degree-ratio factor. Adding the two oriented incidences of an undirected edge, including the two incidences of a loop, bounds its expected traversal load at any step by $24q_v$.

Concentration of local loads. Conditional on the current input, different jobs are independent. At one fixed step, their indicators of traversing a fixed edge are independent Bernoulli variables,

regardless of dependence between different steps of one job. The expected load is at most $24q_v = 24\Lambda^3$ and $\Lambda^6 \geq 6 \cdot 24\Lambda^3$, so the first Chernoff bound shows that this load is at most Λ^6 except with probability at most $2^{-\Lambda^6}$. A union bound over edges and the $2t$ steps gives total local edge load at most $2t\Lambda^6 \leq \Lambda^{20}$; local walk lengths are also at most Λ^{20} .

Substitution through the levels. Suppose that child routes have maximum length and edge congestion at most Z . Initial packet walks contribute at most Λ^{20} load to a current edge. Each virtual-edge embedding is then used at most Z times, so all substitutions contribute at most $Z\Lambda^{20}$. The same inequality bounds length, using the maximum embedding length. Starting at the leaves and iterating through at most $2a^2$ levels bounds both quantities by

$$(2\Lambda^{20})^{2a^2} \leq H_*. \quad (6.22)$$

Tail bounds. A conditioned job exceeds its trial cap with probability at most

$$\left(1 - \frac{1}{4D_c b}\right)^{\Lambda^3 b} \leq \exp[-\Lambda^3/(4D_c)],$$

and a uniform incidence choice exceeds its string cap with probability at most $2^{-\Lambda}$. The group-cut bound (6.19), the point-count and edge-step tails, and these cap tails are all below $M^{-40} = 2^{-40u}$. Indeed, $\Lambda \geq 2^{1000a^2}(1+u)$ and $D_c \leq \Lambda^{1/20}$ give $q_v/(64D_c) \geq \Lambda^2$ and $\Lambda^3/(4D_c) \geq \Lambda^2$. With $N_p \leq M^2$, the sum (6.19) is therefore at most $2M^2 e^{-\Lambda^2} \leq 2^{1+2u-\Lambda}$, and each of 2^{-q_v} , e^{-q_v} , $2^{-\Lambda^6}$, $\exp[-\Lambda^3/(4D_c)]$, and $2^{-\Lambda}$ is at most $2^{-\Lambda}$. Both $2^{1+2u-\Lambda}$ and $2^{-\Lambda}$ are less than 2^{-40u} , since $\Lambda > 42u + 1$.

Work bounds on every run. The bounds on the work of a failed run hold without any good event. Packets never duplicate; each vertex always initiates exactly q_v virtual jobs; group sizes shrink by the group-size rule; and every trial and walk has an explicit cap. At each recursion depth of the routing procedure there are at most $q_v N_p$ virtual edges, with incidence arrays for constant-index access. Even if congestion bounds fail, let $\mathcal{L}_k$ bound a fully expanded packet route with k levels remaining. The newly sampled prefix has at most $2t$ steps, and each edge of the child route expands to at most $2t$ steps. Thus

$$\mathcal{L}_0 = 0, \quad \mathcal{L}_k \leq 2t(1 + \mathcal{L}_{k-1}) \leq \sum_{j=1}^k (2t)^j \leq 2(2t)^k \quad (1 \leq k \leq 2a^2).$$

This is a fixed power of Λ . These deterministic size bounds also bound all elementary choice counts. Under $P_* \leq b$, the total number of pieces, calls, edges, steps, and capped choices in an entire attempt is less than M^{20} . A union bound over these positions, whose tails are below M^{-40} , thus proves the failure bound M^{-20} of the lemma. $\square$

6.4 Filtering real embeddings and verifying a stage

The good event of Lemma 6.5 bounds route lengths and congestion in the overlay, where fake edges are still allowed. This subsection discards routes using fake edges or excessively long real embeddings; the congestion bound limits how many demands this discards and controls the resource load of the remaining expanded routes.

At a *completed piece*, that is, a piece that completes all r_m rounds without a cut, discard every packet route using a fake overlay edge or an overlay edge whose real embedding has more than L_0 sampled-edge hops. This filtering is performed on overlay identities before any long real embedding is substituted. In one piece there are fewer than $\epsilon_0 s' r_m$ fake edges. By (6.14), the total length of all

real embeddings is at most $4r_m\chi s'$, so at most $4r_m\chi s'/L_0$ overlay edges have a long real embedding. Each such edge is used by at most H_* packets, since traversal congestion bounds the number of packets using it. The fraction discarded within completed pieces is therefore at most

$$H_*\epsilon_0 r_m + \frac{4H_*r_m\chi}{L_0}. \quad (6.23)$$

Adding (6.13), the total lost fraction is at most

$$2(D_p + 1)/\chi + H_*\epsilon_0 r_m + 4H_*r_m\chi/L_0 < \frac{1}{2}.$$

Indeed, by (6.5), $2(D_p + 1)/\chi \leq 4D_p/\chi = 4\Lambda^{-5a^2}$, $H_*\epsilon_0 r_m = \Lambda^{3-30a^2}$, and $4H_*r_m\chi/L_0 = 4\Lambda^{3-20a^2}$. Since $a \geq 10$, each exponent is at most $-5a^2 \leq -500$, so the sum is at most $9\Lambda^{-500} < 1/2$. Thus more than half the original stage demands survive all cuts and filtering on the good events.

Replace each remaining overlay traversal by its real embedding. Its direction specifies whether to use the embedding or its reverse. The resulting walks have correct endpoints and at most H_*L_0 hops. In each round, an undirected sampled edge e appears at most $2\chi\omega_e$ times among the real embeddings, by its two directed capacities. It is therefore traversed at most

$$2H_*r_m\chi\omega_e \quad (6.24)$$

times by the final walks from its completed piece.

Completed pieces are vertex-disjoint even when they stop at different depths: no completed piece has descendants, and sibling pieces use disjoint vertex sets. Their samples therefore contain at most $4K_s v_+$ distinct original edges. Combining (6.24) with (6.10), applied separately at each of the at most $D_p + 1$ depths, bounds every resource load by

$$4H_*r_m\chi(D_p + 1)C_0. \quad (6.25)$$

The parameters give each required stage margin:

$$4K_s \leq \Lambda^{350a^2}, \quad H_*L_0 = \Lambda^{270a^2} \leq \Lambda^{350a^2}, \quad 4H_*r_m\chi(D_p + 1) \leq \Lambda^{350a^2}. \quad (6.26)$$

These are the margins for the support, the maximum hop count, and the resource load required of a successful stage in Section 6.1.

The algorithm accepts a stage only after checking these three bounds with Λ^{350a^2} , checking that at least half the demands have been assigned, and checking every returned walk's edge identities and endpoint sequence. It counts total traversals of each selected edge and multiplies these counts into the edge's sparse resource list to verify resource loads. Consequently every accepted stage satisfies these bounds on every computation path, whether or not the overlay is expanding and the probabilistic estimates of the construction hold.

6.5 Failure probability and finite computation

Proof of Lemma 6.1. The proof first shows that one attempt fails with probability less than $1/2$. It then bounds the probability that some stage exhausts its T attempts by $2^{-\sigma}S^{-800}$ and assembles the accepted stages into (6.3). Finally it bounds the work on every computation path, including the finite-word implementation.

Failure probability of one attempt. At a fixed cut depth, condition on its input lists before making its fresh samples. Lemma 6.2 applies to these lists even though earlier cuts were random. Likewise the median potential and the recursive routing estimates are conditional on their respective

inputs. The sum of all bad-event probabilities in one attempt is less than $1/2$ when b , Λ , and P_* are given by (6.2), including the prerequisite $P_* \leq b$, and the internal parameters by (6.5). This bound follows from the following counts. There are at most $M^2(D_p + 1)$ pieces and at most M^2 resources having nonzero input data. The matrix and sample-count tails of Lemma 6.2, the resource tails in (6.10), and the product-mixing tails of Lemma 6.3 are each below M^{-40} , including their per-piece factors. Indeed, $s' \leq M^2/2$, $N_p \leq M^2$, $K_s/\chi^2 = \Lambda^{10a^2}$, a nonempty piece has $\#\{i : d_i > 0\} \geq 1$, and $r_m/\Lambda = \Lambda^2$, so these tails are at most

$$2M^2 e^{-\Lambda^{10a^2}/100}, \quad e^{-K_s/3}, \quad e^{-K_s/30}, \quad 4M^6 e^{-\Lambda^2/80},$$

and each of these is at most $2^{2+6u-\Lambda} < 2^{-40u} = M^{-40}$, because $\Lambda \geq 2^{1000a^2}(1+u)$. For the routing procedure, the group-cut union is bounded by (6.19); the point-count and edge-step tails use $q_v = \Lambda^3$ and threshold Λ^6 ; these and the rejection tails are shown to be below M^{-40} in the stage *Tail bounds* of the proof of Lemma 6.5. Their total number, including capped random-choice positions, is less than M^{20} . These comparisons follow directly from $\Lambda = 2^{1000a^2}(1+u)$, $a \geq 10$, and $P_* \leq b$: every polynomial count in the listed powers of Λ is absorbed by the latter inequality, while every displayed tail has an exponent dominating a constant multiple of u . All good events imply that the stage passes its verification (Section 6.4).

Attempts, stages, and assembly. After each unsuccessful attempt, the unchanged residual graph is tried again with fresh bits. Conditional failure probability is at most $1/2$ per attempt; hence failure of all T attempts has probability at most 2^{-T} . Taking a union bound over at most $2u + 2$ stages gives

$$(2u + 2)2^{-T} \leq 2^{-\sigma} S^{-800}.$$

Indeed, $u \leq 2 \log_2 S$ and (6.6) give $(2u + 2)2^{-T} \leq (4 \log_2 S + 2) 2^{-2\sigma-20} S^{-2000}$, which is at most $2^{-\sigma} S^{-800}$. This also proves the stated conditional guarantee for adaptive inputs. Taking the union of selected supports over accepted stages and retaining each demand's first accepted walk yields (6.3), because

$$(2u + 2)\Lambda^{350a^2} \leq P_*.$$

Support and resource loads add over stages; maximum hop count does not increase when supports are united.

Work on every computation path. The work bound includes rejected attempts. At any fixed cut depth, piece edge counts sum to at most the starting edge count. Sampling scans their lists once. For each piece there are at most r_m flow calls, each on padded size $O((1+\chi)s')$. Lemmas 2.1 and 2.3 bound their finite-word running time by padded size times $M^{o_a(1)}$, uniformly over these local sizes. In particular, the padded sizes are $O((1+\chi)M^2)$, and the subpolynomial overhead of these flow computations at these sizes is $M^{o_a(1)}$. The total padding over a depth is linear in its edge count times a fixed power of Λ . The weight check (6.8) bounds flow throughput by (6.14) even for a sample with inaccurate cuts. The time to construct all integral paths is therefore bounded by (6.14) plus the network size. By the projection computation of the median rule (Section 6.2), projecting sign vectors takes $O(jN_p)$ averages in round j , hence $O(r_m^2 N_p)$ operations per piece. Sorting projected values to choose each median takes $O(N_p \log N_p)$ comparisons, which adds only a logarithmic-power factor.

At a completed piece, the initial overlay has $r_m N_p/2$ edges. At each routing depth there are at most $q_v N_p$ virtual edges and $O(q_v N_p)$ jobs across its calls. Store all incident-edge arrays, so that each walk step accesses its chosen incidence by index. A job uses at most $\Lambda^3 b$ trials, each of $2t$ steps, and each incident choice has at most Λ bit-string trials. These caps supply the sole additional nonlogarithmic factor b . The deterministic recursion-depth and substitution bounds from

Lemma 6.5 control intermediate walk-list sizes even on bad random outcomes. Long real embeddings are filtered before substitution; all remaining expansions have a further factor of at most L_0 . Thus explicit output construction is also linear in the input size times b and a fixed power of Λ .

Finite-word implementation. Sparse dictionaries translate original labels, edge identifiers, and resource names to local indices and back. Verification first counts traversals by selected edge and then scans its resource list once, multiplying each entry by that count. All comparisons, capacity operations, resource-load products, and dyadic averages use bounded multiword integers. In particular, the projection denominator has at most r_m binary digits (Section 6.2); local path lengths and load counters have the deterministic size bounds just proved; and all input bit strings have the assumed logarithmic-power width. The normalized matrices, potentials, and stationary probabilities, together with their square roots and eigenvectors, appear only in the analysis. The sampling probabilities are found by exact integer comparisons, and an event of probability $2^{-\ell}$ is drawn using ℓ fair bits. Here $\ell \leq 2u$, since the positive piece degrees are at most M^2 . Each random sign uses one fair bit. Standard multiword implementations of all arithmetic operations add only logarithmic-power factors. Individual generated bits, initialization of arrays, allocation, and movement of these records are included in the same bounds. Multiplication by the capped numbers of cut depths, routing depths, stages, and attempts gives the bound (6.4). Since all caps are enforced independently of the good events, this is a bound on every computation path. $\square$

7 Approximation ratio of the cycle data structure

This section and Section 8 construct the cycle data structure of Lemma 5.2; this section proves its approximation guarantee (5.9). The structure is a recursion over levels $i = 0, \dots, d$ on graphs of decreasing size. At the bottom level a direct search suffices (Lemma 7.1). At a level $i < d$, updates are grouped into epochs (Section 7.2), and during an epoch the graph present at its start is kept fixed (frozen). For each epoch the argument has four steps:

- (1) verified random covers of the frozen graph (Lemma 7.2) and a finite family of forests built from them (Lemma 7.3);
- (2) random upward routes in these covers (Lemma 7.4), which decompose the part of every current circulation on the edges of the frozen graph that are still present into fundamental cycles of the forests of this family plus a vector on the shortest-path trees of the cells of the covers (Lemma 7.5);
- (3) local correction cycles, which replace the fundamental cycles opened by deletions with current circulations of bounded raw length, in the sense of Section 2 (Section 7.5);
- (4) one smaller child graph J , which carries what remains, in compressed form (Section 7.6 and Lemma 7.7).

Lemma 7.8 combines these steps: the ratio of every current circulation is at most b^2 times the larger of the best local correction score and the optimum of J . Corollary 7.10 applies this bound at every level. Section 8 supplies the path assignments used by the local corrections, maintains the child graph, and publishes a forest in which the returned walks have the required representation and recourse; its completion of the proof of Lemma 5.2 uses Corollary 7.10.

7.1 Parameters, certified scores, and the bottom level

We keep the parameters of Section 6; recall from (6.2) that

$$b = M^{1/a^2}, \quad \Lambda = 2^{1000a^2}(1+u), \quad P_* = \Lambda^{400a^2}, \quad M = 2^u. \quad (7.1)$$

Here u is a positive multiple of $2a^2$ and $S^{1/a} \leq M \leq S^2$. All powers in the following display are integers:

$$\begin{aligned} d &= 2a - 4, & k &= M^{1/a}, & n_i &= M^{1-i/(2a)}, & G_i &= 1200a + 2i \quad (0 \leq i \leq d), \\ r &= (1210a + 3)a^2, & \Gamma &= K_h = \Lambda^2, \\ A_* &= 64r\Gamma K_h b, & p_* &= (10P_*)^{a^2+1}, & Z_* &= 40(p_* + 1). \end{aligned} \tag{7.2}$$

The routing sparsifier (Lemma 6.1) has the checked prerequisite $P_* \leq b$ of (6.2). We impose it together with one further deterministic check:

$$P_* \leq b, \quad 20Z_*A_* \leq b^2. \tag{7.3}$$

For each fixed a , these inequalities hold for all sufficiently large M : every factor other than the displayed powers of M is a fixed power of $1 + u$. As in Section 2, a failed check suspends the parameter branch. Throughout this section the parameters satisfy (7.2) and (7.3).

The parameters in (7.2) have the following roles.

- *Recursion.* Level i handles graphs of size at most n_i whose prices have magnitude at most M^{G_i} . The bottom size $n_d = M^{2/a}$ is small enough for the direct search of Lemma 7.1. The exponent $G_0 = 1200a$ is the price bound of Lemma 5.2, and the increment $G_{i+1} = G_i + 2$ absorbs the factor $k + 1$ by which the edges of the smaller graph built in Section 7.6 can exceed the length and gradient bound of level i . The depth d determines the final ratio in Corollary 7.10.
- *Update batches and forest separation.* The quantity k sets the number $h_i = n_i/k$ of input edits in (7.4), and in Lemma 7.6 the size below which the components left after removing a sparse vertex set of a forest must fall.
- *Random partitions.* Section 7.3 uses r layers of random vertex partitions, with radii $R_j = b^j$; the number r is large enough that $R_r/2$ exceeds every finite component diameter (stage “Agreement at the top layer” of the proof of Lemma 7.4). In Lemma 7.2, Γ caps the random delays and hence bounds the radii of the cells by ΓR_j , K_h is the number of partitions per layer, and the choice $\Gamma = K_h = \Lambda^2$ makes the failure bound (7.6) hold.
- *Losses.* The constant A_* bounds the expected lengths in Lemma 7.4 and hence the coefficient bounds of Lemma 7.5; p_* bounds the number of hops of the paths Π_f of Section 7.5; and $Z_* \geq 17(p_* + 1)$ gives the length bounds (7.17) and (7.19). The second check in (7.3) turns the constants of Lemma 7.8 into the factor b^2 ; it also implies $b > 2\Gamma$, which is used in the stage “The route $P(v)$ ” of the proof of Lemma 7.4. The arguments of this section use the factors 40 and 20 only through $Z_* \geq 17(p_* + 1)$ and $14Z_*A_* \leq b^2$.

Level i accepts an undirected multigraph on fixed vertex slots, of total size at most n_i , counting slots and edge copies. Its lengths are positive integers and its signed gradients are integers, with

$$1 \leq \ell_e \leq M^{G_i}, \quad |\gamma_e| \leq M^{G_i}.$$

Each edge has an identifier and a reference orientation. The degree bound needed for the running time is a fixed-parameter logarithmic power; Section 8 proves that this bound persists under recursion. The outer input has degree at most three. Write OPT_i for the maximum circulation ratio on the current level- i graph, with the same zero convention as in Lemma 5.2. The level-0 graph is the input graph of Lemma 5.2, so OPT_0 is its optimum OPT of (5.7); below, OPT without a subscript or an argument always denotes this outer optimum.

A *certified score* is a nonnegative rational lower bound on the absolute gradient divided by the positive raw length of a returned traversal list. A zero score may accompany no candidate. An edge is *actual* if it is an input edge of level i , and a traversal is *actual* if it traverses such an edge, as opposed to the artificial null links introduced below. Raw length counts every actual traversal with multiplicity; in the constructions below, assigning length one to an artificial null link only enlarges the length used for an upper bound. A vector used solely in an existential comparison is measured instead by its length norm; recall from Section 2 that this is the sum of length times the absolute *net* coefficient on each edge.

Lemma 7.1 (Bottom-level search). *At level d , a query returns individual oriented edges forming a circulation with certified score at least $\text{OPT}_d/2$, whenever $\text{OPT}_d > 0$. Initialization and each graph operation or query cost $M^{O(1/a)+o_a(1)}$, and the answer has at most $n_d = M^{2/a}$ pieces.*

Proof. The query tests dyadic ratio thresholds by negative-cycle detection; since the optimum is attained by a directed cycle and lies in a bounded range, the largest successful threshold is within a factor two of it.

The search. Replace each undirected edge by its two possible traversals. For each dyadic threshold $\rho = 2^j$ with

$$-(G_d + 2)u \leq j \leq (G_d + 1)u,$$

give these two arcs costs $\rho\ell_e - \gamma_e$ and $\rho\ell_e + \gamma_e$, respectively. For every start vertex, tabulate minimum costs of walks with each number of arcs from zero through the number of vertices, retaining predecessor pointers. This detects a negative closed walk whenever one exists, because a negative closed walk contains a negative simple directed cycle. Keep a witness for the largest successful threshold, and return its exact absolute-gradient to raw-length ratio.

Approximation. Negating a nonzero signed circulation if necessary, we may assume that its gradient is nonnegative. Orienting its used edges along their coefficient signs gives a nonnegative directed circulation, which decomposes into simple directed cycles. Gradient and length add in this decomposition, so one cycle has ratio at least that of the circulation. Conversely, any positive-ratio directed cycle projects to a nonzero signed circulation; its projected length norm is at most its raw length. It follows that OPT_d is attained by a directed cycle. Integer gradients and positive integer lengths give

$$\frac{1}{n_d M^{G_d}} \leq \text{OPT}_d \leq M^{G_d} \quad \text{if } \text{OPT}_d > 0.$$

The tested thresholds cover this interval with room on both sides. The largest successful threshold is at least $\text{OPT}_d/2$, and its negative walk has still larger ratio. The witness has at most the number of vertices in the graph.

Running time. There are $O_a(1+u)$ thresholds. The elementary tables take at most $O(n_d^3)$ arithmetic operations per threshold, since both the vertex and arc counts are $O(n_d)$. All costs have $O_a(1+u)$ bits after a common dyadic scaling. This proves the stated bound, including multiword arithmetic. $\square$

7.2 Batched updates and the decomposition target

For $i < d$, level i works in epochs of

$$h_i = n_i/k \tag{7.4}$$

input edits, a price replacement counting as a deletion followed by an insertion. At the beginning of an epoch the current nonloop graph is frozen: it is recorded, including its prices and edge identities, as the *frozen graph* $\bar{G}$. An old edge keeps these data until its death; until then it is a *surviving*

frozen edge. Every later insertion is a *new* edge, even when its endpoints and prices coincide with those of a dead edge. A heap on all current loops supplies exact single-loop candidates. Loops can otherwise be omitted: their coordinates are independently circulations, and the optimum on a graph is the maximum of its loop optimum and its nonloop optimum. Throughout Sections 7.3–7.6 we fix a level $i < d$ and an epoch at this level, with frozen graph $\bar{G}$; states are states during this epoch.

Fix a current nonloop circulation Δ , and write $\Delta = \Delta_s + \Delta_n$ on the surviving frozen and new edges. The surviving part need not be a circulation: its imbalance is the negative of that of Δ_n , so it is supported at new-edge endpoints. We will express Δ as a linear combination of current circulation candidates plus the projection of one circulation in a smaller graph. The coefficient-weighted lengths and the child vector’s length norm must be bounded by a controlled multiple of $\|\ell\Delta\|_1$. Each candidate, however, also needs a bound on its own raw traversal length, because Lemma 5.2 measures a returned answer by its raw length V in (5.9).

7.3 Verified covers and a finite family of forests

The covers below adapt the shortest-path partitions with random shifts of Miller, Peng, and Xu [27, Algorithm 2 and Lemmas 4.1 and 4.4]. We prove the required guarantees for weighted distances, geometric delays, verified padding, and capped retries.

For each layer $j = 1, \dots, r$, let $R_j = b^j$.

Lemma 7.2 (Random partitions with every ball in one cell). *Assume (7.2) and (7.3), and let the level $i < d$, the epoch, and its frozen graph $\bar{G}$ be as fixed in Section 7.2. For every layer j there is a randomized construction of K_h partitions of the vertices of the frozen graph $\bar{G}$ into cells, with a shortest-path tree in each cell, such that:*

- (i) *every cell tree has radius at most ΓR_j about its center;*
- (ii) *for every vertex, at least one of the partitions has a cell containing its entire closed R_j -ball in $\bar{G}$.*

The construction checks properties (i) and (ii) for the partitions it returns. It takes $n_i M^{o_a(1)}$ time on every computation path for fixed reliability parameter σ , with detected-failure probability at most $2^{-\sigma} S^{-800}$ conditional on any valid frozen input. It uses integer operations and fresh fair random bits.

Proof. Each partition clusters the vertices by randomly delayed shortest-path distances: every vertex joins the center of largest score (7.5). The cells have small radius, a margin over the runner-up certifies that a whole ball lies in one cell, each partition gives this margin at a given vertex with probability at least $1/8$, and independent partitions with capped retries give the failure bound.

Cells and their trees. Independently for each partition, give every vertex c an integer delay X_c with $\Pr[X_c \geq t] = 2^{-t}$ for nonnegative integers t . At v , choose the center maximizing

$$R_j X_c - \text{dist}_{\bar{G}}(c, v), \tag{7.5}$$

breaking ties by center identifier. Reject the attempt if any delay exceeds Γ . A vertex’s own center has nonnegative score, so its winning center is at distance at most ΓR_j . Dominance of a center persists along a shortest path towards it: the triangle inequality preserves every score comparison, including the common identifier tie rule. Hence a shortest-path tree to the center lies inside the winning cell.

Checking the padding. For each vertex also compute the runner-up among distinct centers. Verify that at each layer every vertex has, in some partition, winning margin greater than $2R_j$. The difference between the scores of two centers changes by at most twice the distance moved.

Thus this check implies containment of the whole closed R_j -ball. A runner-up in another connected component has score minus infinity.

Success probability and retries. For unlimited geometric draws, fix a vertex and all delays except that of a proposed winner c . The event that c wins is a lower integer threshold on X_c , truncated below at zero. Exceeding that threshold by three raises its score by $3R_j$ and gives margin greater than $2R_j$. The geometric tail assigns probability at least $1/8$ of the winning-event probability to this overshoot. Summing over the disjoint winning events gives margin probability at least $1/8$. Independence of the partitions and a union bound therefore bound the probability that the delay or coverage checks fail by

$$rK_h n_i 2^{-\Gamma} + r n_i (1 - 1/8)^{K_h} < \frac{1}{2}. \quad (7.6)$$

The displayed inequality follows from the chosen scales in the checked parameter range: $n_i \leq M = 2^u$ and $r \leq \Lambda$, while $\Gamma = K_h = \Lambda^2$, $u < \Lambda$, and $(1 - 1/8)^{K_h} \leq 2^{-K_h/8}$, so each term on the left is at most $\Lambda^3 2^{\Lambda} 2^{-\Lambda^2/8}$, which is less than $1/4$ because $\Lambda \geq 2^{1000a^2}$. Repeat with fresh draws at most $2(\sigma + 1000 \log_2 S + 10)$ times. Exhaustion is a detected failure; its probability is at most $2^{-\sigma} S^{-800}$. This matches the failure bound of the routing sparsifier (Lemma 6.1), and the stage “Failure probability” of the proof of Lemma 8.10 sums these bounds over at most S^{500} randomized invocations.

Implementation and running time. Generate a geometric delay by fair bits, stopping and rejecting when its cap is exceeded. Shifted distances and the best two distinct center labels are computed by multi-source Dijkstra with initial offsets $-R_j X_c$, finalizing at most two distinct origins at each vertex. An origin among the best two at a vertex has prefixes among the best two along a shortest path; otherwise two strictly preferred origins at a prefix would remain preferred at the endpoint. Thus this truncation loses neither of the required origins nor a winning tree pointer. Initial offsets can be negative because edge lengths are positive. Each partition needs only near-linear integer graph work. The layer, partition, delay, and repetition counts are fixed powers of $1 + u$ for fixed a, σ , and their integer data have fixed-parameter logarithmic bit length. This proves the time bound on every computation path. $\square$

Let H_j be the disjoint union of copies of all the cell trees in the K_h partitions at layer j . We call the forests H_j the *raw forests*; they are kept separately from the copies of their trees inside the family forests T^0 below. A tree edge maps to its frozen actual edge, with the appropriate endpoint correspondence and reference orientation. Introduce additional vertices $V_j(v)$ for every frozen slot v and $0 \leq j \leq r$. The layered system has two kinds of artificial links:

- an OUT link from $V_{j-1}(v)$ to the copy of v in each partition of H_j ;
- an IN link from every cell root c to $V_j(c)$.

At layer j , the vertices $V_{j-1}(v)$ are the *lower slots*, and the K_h OUT links leaving a lower slot are its *OUT options*. Each artificial link maps to a null traversal at one level- i slot, that is, at one vertex slot of the level- i input graph, and is assigned length one and gradient zero for the construction.

The following finite family adapts the bit-selection construction of Chen et al. [4, Definition 4.30 and Lemma 4.31].

Lemma 7.3 (Finite family of routing forests). *Assume (7.2) and (7.3), fix the level $i < d$ and the epoch of Section 7.2, and let the cell trees and the layered system be those of the partitions of Lemma 7.2. There is a family of at most $(K_h^2(u+1))^{O(r)}$ forests T^0 with the following property. For every choice, at each layer, of two distinct lower slots and of an OUT option for each of them, some member of the family contains all the chosen OUT links simultaneously across the layers. Each*

member contains separate copies of all cell trees and IN links, and exactly one OUT link per lower slot.

Proof. At a given layer choose a bit position of the slot index and two partition options, using the chosen option as a function of that bit. Include constant functions as well. Two distinct indices differ in some bit, so any two prescribed options can be realized. Choose these functions independently at the r layers and include every choice in the family. There are at most $2K_h^2(u+1)$ choices per layer, giving the claimed bound.

With each cell tree oriented towards its root and the links towards higher layers, every vertex has outdegree at most one and there is no directed cycle. Such an underlying undirected graph is a forest: an undirected cycle would, by its edge count and outdegree bound, force a directed cycle. This also proves the asserted structure of each T^0 . $\square$

The chords of each T^0 are the frozen input nonloops on its V_0 vertices, and a null pair chord between every two distinct OUT-option targets of the same lower slot. The latter has length one and gradient zero, whether or not either OUT option was selected in T^0 . Retain only chords whose endpoints are connected in that forest. For such a chord f , let C_f^0 be its signed fundamental cycle in the forest T^0 as built at the start of the epoch, traversing f forward and the tree path backwards. Let w_f be its full length, including artificial links, and fix the bucket

$$W_f = 2^{\lceil \log_2 w_f \rceil}. \quad (7.7)$$

The bucket W_f is computed from the bit length of w_f by integer operations.

Whenever a frozen actual edge dies, delete every tree edge mapping to it, both in the T^0 copies and in the raw forests H_j . These deletions are called *holes*. An edge of one of these forests is *live* if it has not been deleted, and a chord is live if it is a surviving frozen edge or a null pair chord; live forests, live components, and live paths consist of live edges. For a surviving chord f , write C_f^{clip} for the chord together with the surviving signed tree traversals of C_f^0 . Null chords always survive. The clipped vector need not be a circulation.

7.4 Route distributions and the decomposition of a circulation

Let π denote projection onto the frozen actual edge coordinates, ignoring null links and adding signed coefficients of copies. Thus π maps vectors on the forests T^0 and H_j and on their chords to vectors on the frozen edges; Section 7.6 extends π to vectors on the child graph. We first define a distribution of an upward route for every vertex. These distributions are used only in the proof, and are not sampled or evaluated by the algorithm.

Two features of this construction, the normalized boundary-distance weights and the retention of each starting vertex's route law, adapt the origin-aware routing of Rozhoň et al. [29, Section 4]. Its dynamic development appears in Chen et al. [4, Section 4.3].

To account for the imbalance of Δ_s , we will compare a surviving edge with the difference of two upward routes, one from each endpoint. The same vertex must have the same route law when it appears at the endpoint of different edges. With this fixed marginal, the route contributions cancel wherever Δ_s is conserved; only its imbalance at new-edge endpoints remains. Thus the route law will be indexed by the starting vertex, even when the route has reached a different representative in a higher layer.

For a cell C at layer j , including its partition identity, define the cell weight

$$a_{j,C}(v) = \max \left\{ 0, \min \{ R_j, \text{dist}_{\bar{G}}(v, V(\bar{G}) \setminus C) \} - \frac{R_j}{2} \right\}. \quad (7.8)$$

Distance to the empty set, or to vertices in other components only, is infinity. Normalize these weights to a distribution $\mathcal{L}_j(v)$ on all cells at that layer. A verified covering cell gives a weight of at least $R_j/2$, so the normalizer is at least $R_j/2$. A positive-weight cell contains every vertex at distance at most $R_j/2$ from v .

Lemma 7.4 (Distribution of the random upward routes). *Assume (7.2) and (7.3), fix the level $i < d$ and the epoch of Section 7.2, and let the cells be those of Lemma 7.2. The independent layer choices $\mathcal{L}_j(v)$ define an upward path $P(v)$ in the layered system with all OUT options. For every frozen edge $e = vw$, its two path laws admit a coupling such that the closed walk $e + P(w) - P(v)$, after canceling a common final suffix, has expected full length at most $A_*\ell_e$. The marginal law of $P(v)$ is the same for every incident edge used in this assertion.*

Proof. The proof compares the cell laws of the two endpoints of e layer by layer. Neighboring vertices have nearly equal laws, so under a maximal coupling the two routes are retained at layer j only with probability $O(K_h\ell_e/R_{j-1})$. Since that layer has length $O(\Gamma R_j)$ and $R_j/R_{j-1} = b$, its expected contribution is $O(\Gamma K_h b \ell_e)$. At the top layer the laws coincide, and the common suffix after the last disagreement cancels.

Neighboring cell laws. The functions in (7.8) are one-Lipschitz on a connected component. At two vertices, at most $2K_h$ cell weights can be nonzero, since a positive weight requires membership in the cell. Comparing normalized weights along $e = vw$ yields

$$\text{TV}(\mathcal{L}_j(v), \mathcal{L}_j(w)) \leq \frac{4K_h\ell_e}{R_j}. \quad (7.9)$$

Indeed the sum of absolute unnormalized changes is at most $2K_h\ell_e$, and total variation of the normalized laws is at most that sum divided by the smaller normalizer.

The route $P(v)$. The route $P(v)$ starts at $V_0(v)$, uses the cell chosen at layer one, then its tree path to its center and the IN link to the corresponding V_1 vertex, and continues in the same way at the higher layers. At layer $j \geq 2$ the previous representative is the center of a cell containing v , so its distance from v is at most ΓR_{j-1} . The second check in (7.3) implies $b > 2\Gamma$, and therefore

$$\Gamma R_{j-1} < R_j/2.$$

Every positive next-layer choice contains that representative, so its OUT option exists. This defines $P(v)$ using independent choices across layers and a marginal law fixed by v alone.

Agreement at the top layer. For the endpoints of e , couple their cell choices maximally at each layer, independently between layers. At the top, $R_r/2$ exceeds every finite component diameter: such a diameter is at most $n_i M^{G_i}$, whereas $R_r = M^{1210a+3}$. A top-layer cell with positive weight at a vertex must consequently contain its whole connected component. All such cells have weight $R_r/2$ at every vertex of that component. The two endpoint laws therefore coincide at the last layer and can be coupled to agree, including the partition identity.

Expected length after cancellation. Retain the two routes through one layer after their last cell disagreement; if there is no disagreement, retain layer one. From then on their representative and choices coincide, so their suffixes are identical and cancel. Two routes at layer j , including their OUT and IN links, have total length at most $4\Gamma R_j$. For $j \geq 2$ this layer is retained only if a disagreement occurred at some layer $l \geq j-1$. By (7.9) and a geometric sum, that probability is at most

$$\sum_{l \geq j-1} \frac{4K_h\ell_e}{R_l} \leq \frac{8K_h\ell_e}{R_{j-1}}.$$

Including the original edge and the first layer, the expected length is consequently at most

$$\ell_e + 4\Gamma b + \sum_{j=2}^r 32\Gamma K_h b \ell_e \leq 64r\Gamma K_h b \ell_e = A_* \ell_e.$$

The coupling changes none of the prescribed per-vertex marginals. $\square$

For a surviving frozen edge $e = vw$, its *coupled walk* is the closed walk $e + P(w) - P(v)$ of Lemma 7.4, under the coupling constructed in its proof: the cell choices of v and w are coupled maximally at each layer, independently between layers. The two routes are *trimmed*: each is retained through one layer after the last cell disagreement, or through layer one if there is none, and the identical suffixes beyond are canceled. In the next lemma, $\|z\|_{\text{length}}$ denotes the length norm of Section 2, recalled before Lemma 7.1, of a vector z on the raw forests.

Lemma 7.5 (Fundamental-cycle decomposition after deletions). *Assume (7.2) and (7.3), and fix the level $i < d$ and the epoch of Section 7.2. Fix any state of the epoch and any circulation Δ on its current nonloops. Write $\Delta = \Delta_s + \Delta_n$ for its surviving frozen and new coordinates. Let C_f^0 and w_f be the fundamental cycle and full length of a chord f defined before (7.7), C_f^{clip} its clipped vector defined after it, and $P(v)$ as in Lemma 7.4. There are real coefficients α_f on live fundamental chords and a vector z on the raw forests H_j such that, before the frozen coordinates are restricted to the surviving ones,*

$$\sum_f \alpha_f \pi C_f^0 = \Delta_s + \pi z, \quad (7.10)$$

$$\sum_f |\alpha_f| w_f \leq A_* \|\ell \Delta_s\|_1, \quad (7.11)$$

$$\|z\|_{\text{length}} \leq A_* \|\ell \Delta_s\|_1. \quad (7.12)$$

Moreover z is the raw-tree part of

$$\sum_v (\text{in}_{\Delta_s}(v) - \text{out}_{\Delta_s}(v)) \mathbb{E}[P(v)]. \quad (7.13)$$

In (7.13) the coefficient of $\mathbb{E}[P(v)]$ vanishes unless v is an endpoint of a new edge. If z_L is the restriction of z to live raw forest edges, then

$$\sum_f \alpha_f \pi C_f^{\text{clip}} - \pi z_L = \Delta_s. \quad (7.14)$$

Proof. Each coupled walk splits into fundamental cycles of family forests and route parts on the raw forests; averaging over the coupling and summing over the edges of Δ_s gives the identity, the bounds follow from Lemma 7.4, and deleting dead coordinates gives the clipped identity.

Decomposing one coupled walk. This step uses the pair-chord construction of [4, Definitions 4.28 and 4.30 and Lemma 4.31]. Each coupled walk decomposes into fundamental cycles from the finite family. To see this, we split the two trimmed routes at their common representative vertices V_j . Before the first common vertex, the two lower slots at every layer are distinct. Lemma 7.3 therefore supplies a forest containing the paths that close the input chord e . Between successive common vertices, equal first OUT options give identical tree paths to the next layer and cancel. If those first options differ, replace the two null OUT traversals by the corresponding null pair chord. For the remaining routes in that piece, all simultaneous lower-slot prescriptions concern distinct slots;

another family member realizes them. The pair-chord endpoints are connected there, as the routes meet at the next common representative. Simplifying each tree walk preserves its signed tree vector and can only decrease length. Replacing two unit null links by one unit pair chord also cannot increase length. Thus the projected coupled walk is a sum of projected oriented fundamental cycles whose full lengths sum to at most its original length.

Summing over edges. We apply this decomposition to each surviving frozen edge, weight it by the edge's signed coefficient in Δ_s , and take expectations over the couplings; these distributions have finitely many outcomes. Collecting coefficients of the oriented family chords gives α_f ; collecting the signed route parts on the raw copies gives z . Only surviving input edges and null pair chords are used as chords, so every indexed chord is live. Summing the decompositions gives the identity (7.10). The triangle inequality and the expected-length bound of Lemma 7.4, applied to the fundamental cycles of the previous step and to the route parts, give (7.11) and (7.12).

Marginal cancellation. Although the couplings may differ between incident edges, the law of each endpoint path is fixed. Summing the route term $P(w) - P(v)$ over oriented edges therefore gives (7.13). Cancellation of identical final suffixes does not change this signed vector. Because Δ is a circulation, the displayed imbalance of Δ_s is the negative of that of Δ_n and vanishes away from new-edge endpoints.

Clipping. Finally, we delete every coordinate of every tree copy mapped to a dead frozen edge. Projection after this deletion is the same as restriction of the projected vector. Restricting (7.10) therefore proves (7.14), even when coefficients on a dead edge had canceled between different copies. $\square$

The identity (7.14) accounts for every surviving frozen coordinate. Its fundamental-cycle terms need not be current circulations, because holes may have opened them. We next extract current circulation candidates using short routed corrections and compress the remaining forest vectors. By Lemma 7.5, z involves only upward routes from endpoints of new edges; the endpoints of the cell-tree paths of these routes, together with hole endpoints, will supply the compression terminals for this raw-forest remainder (Section 7.6).

7.5 Local correction cycles

Lemma 7.6 (Few vertices whose removal leaves small components). *A forest admits a set of at most its number of vertices divided by k portals such that every component remaining after portal removal has fewer than k vertices. The set is constructed by one postorder traversal.*

Proof. Root each tree arbitrarily. Accumulate the number of still-connected unmarked vertices in the subtrees of a vertex, including the vertex itself. If the accumulated count is at least k , mark that vertex as a portal and pass count zero to its parent. Every mark charges a disjoint group of at least k vertices. In any component off the marks, the residual count at its highest vertex is exactly its size and is below k . $\square$

Place such a portal set, called the *grid*, on every T^0 and separately on each raw forest. Call the components off the grid *pockets*. Grids remain compression terminals for the whole epoch. Classify and assign each live fundamental chord separately in its forest and bucket W . A chord is either used directly, through its intact fundamental cycle, or it uses the leg scheme below; a chord using the leg scheme is called *active*. In the leg scheme, each endpoint of the chord, called an *incidence*, has a *leg*: a live tree path from that endpoint to a tree node, its *target*. A *class* is a group of incidences with a common target tree node, identified by a class label. The classes of a forest and bucket are the

vertices of their tree-bucket graph, defined before Lemma 8.1; a *logical path* is a path in that graph, and its edges, the *hops*, are active chords.

- (i) If its full frozen tree path touches the grid, call the chord *regular*. At each incidence choose the nearest portal to its endpoint in the full frozen tree metric, breaking ties by a fixed portal order. Use the leg towards that portal, truncated at the first hole and ending on the incidence side of that hole. The full path contains a portal at distance at most $w_f \leq W$ from each endpoint, so each such leg has length at most W .
- (ii) Otherwise the full tree path lies in one pocket. Root that pocket and measure weighted depth. For this bucket take connected clusters in half-open depth bands of width $4W$, using both the unshifted bands and their translate by $2W$. The depth range of the chord path has width at most its path length, hence at most W ; one of the two band systems contains that whole range. Assign the chord to its containing connected cluster there. Its tree diameter is at most $8W$, since the least common ancestor of two vertices in a connected band cluster is in that cluster.

Until any interior edge of that cluster is cut, use the intact fundamental cycle directly. At its first such cut, activate all surviving assigned chords. Give each of the two resulting sides a class with target its endpoint of the cut. A later cut splits the corresponding live fragment into two fresh classes, with targets the new cut endpoints. All resulting legs stay inside the original cluster and have length at most $8W$.

Regular incidences are initially grouped by their target portal; holes subsequently split those classes. Splits always create fresh class labels, including for an empty side. The class maintenance is given in Lemma 8.1.

For every active f , the construction in Section 8 supplies a logical path Π_f between its endpoint classes, using selected active chords of the same forest and bucket and containing at most p_* hops (Lemma 8.2 and (8.10)). For the selected chords, see Section 8.2 before Lemma 8.2. The assignments may use an empty path for a logical loop, that is, an active chord whose two endpoint classes coincide. If f is oriented from its tail to its head, let L_f traverse from its tail target to its tail, then across f , and then from its head to its head target. Such walks concatenate, also in reverse, to realize logical paths; write L_{Π_f} for the resulting walk. Every logical hop has full length at most

$$8W + W + 8W = 17W. \quad (7.15)$$

The local correction candidate is

$$B_f = \begin{cases} C_f^0, & \text{while the chord is used directly,} \\ L_f - L_{\Pi_f}, & \text{for an active chord.} \end{cases} \quad (7.16)$$

Here a negative walk is traversed backwards. The formula specifies a traversal list: both walks are traversed, and the raw length of the list counts every traversal of both. Its projection is a circulation, and its raw actual length is at most

$$17(p_* + 1)W_f \leq Z_*W_f. \quad (7.17)$$

The direct case satisfies the same bound because $w_f \leq W_f$. Hence

$$\frac{|\gamma^\top \pi B_f|}{Z_*W_f} \quad (7.18)$$

is a certified score. A positive numerator ensures positive raw actual length: null traversals have zero gradient. The list consists of actual individual chord edges and live forest paths; null chord traversals can be omitted. Its numerator is computed from integer path-gradient sums, with the positive denominator Z_*W_f .

7.6 The child graph

The smaller instance is one graph J incorporating all family forests and the raw forests. In each live component take the Steiner subtree of its terminals, retain the terminals and branching vertices, and suppress degree-two nonterminals: each maximal path of the Steiner subtree whose interior vertices are degree-two nonterminals, a *suppressed segment*, becomes one edge, its *compressed segment*. We call this construction *compression*. A one-terminal component contributes its terminal alone; an empty terminal set contributes nothing. The vertices retained by compression are the *compressed vertices*. The terminal set includes:

- (i) every initial portal and both endpoints of every hole;
- (ii) every class target and the *original tree-node* endpoints of every selected chord, that is, the forest nodes at which the chord is attached, as opposed to the targets of its legs;
- (iii) in the raw forests, both endpoints of every cell-tree path that an upward route from v can use, for every slot v that is incident to a new edge at some time in this epoch.

Terminals, once added, may remain terminals to the end of the epoch. For item (iii), enumerating all sequences of partition options suffices: it yields a superset of the endpoints required there, including those of routes with probability zero under the laws of Lemma 7.4. The enumeration lists only the endpoints of these paths, $K_h^{O(r)}$ per incident slot, a fixed-parameter logarithmic factor.

A compressed segment receives the sum of its constituent lengths and the oriented sum of their gradients; we write ℓ^J for the lengths of J . Include each currently selected chord on its original tree-node endpoints, with its actual or null prices; earlier selected chords may also remain while live. Introduce a hub for each level- i slot that is the level- i image of a compressed vertex or an endpoint of a current new nonloop. Join each compressed vertex to the hub of its level- i image by a unit-length, zero-gradient null edge, its *hub connector*. Include all current new nonloops directly between their hubs. As an input of level $i + 1$, the graph J has vertex slots, the *child slots*, which are assigned in the proof of Lemma 8.4 (stage *Child slots, hubs, and capacity*); unused child slots remain isolated.

Every child edge expands into a live parent walk between the level- i images of its endpoints, with exactly its gradient and with raw actual length at most its child length. We also write π for the induced projection from child coordinates to current parent coordinates; on surviving frozen edges it agrees with the earlier projection. A compressed segment has at most $k + 1$ constituent edges: it has no internal portal and its internal nodes lie in one pocket. Consequently its length and absolute gradient are at most

$$(k + 1)M^{G_i} \leq M^{G_i+2} = M^{G_{i+1}}.$$

The same bound holds for the other child edges. Lemma 8.4 bounds the number of child slots and edits and their degrees. For Lemma 7.8, the graph J and the expansion maps just described suffice.

Lemma 7.7 (Compression of a forest vector). *Let Steiner subtrees, suppressed segments, and compression be as defined at the start of Section 7.6. If a vector on a forest has divergence only at a specified terminal set, it vanishes off the terminal Steiner subtrees and has constant oriented coefficient along each suppressed segment, and compression preserves its gradient and its length norm exactly.*

Proof. Deleting an edge outside a terminal Steiner subtree leaves a side containing no terminal. The sum of divergences on that side is zero, so the edge coefficient is zero. At an internal degree-two nonterminal of a remaining segment, zero divergence forces the two consistently oriented coefficients

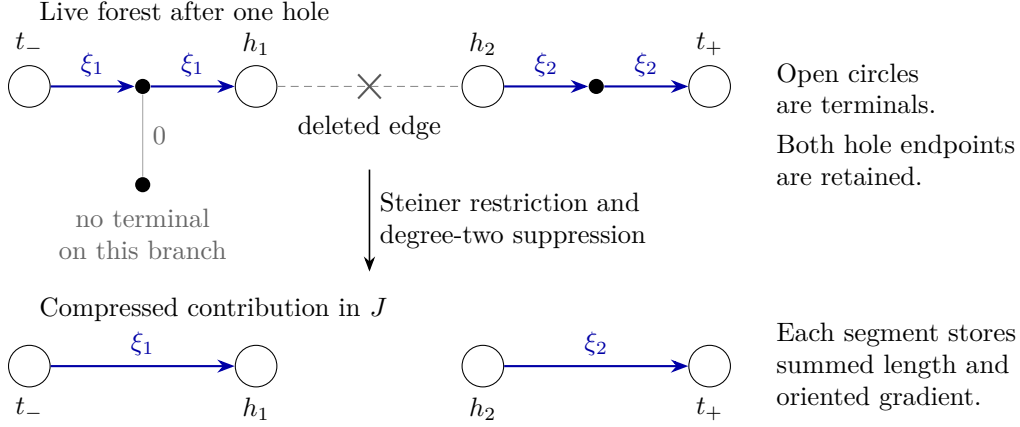


Figure 2: Compression preserves a net tree flow whose divergence is supported on terminals. The shown targets t_-, t_+ and hole endpoints h_1, h_2 are retained; the terminal-free branch has zero flow, and each surviving path becomes one segment with the same coefficient. This compression produces the child vectors J_f and $\hat{z}_L$ used in the proof of Lemma 7.8; the pictured contribution need not be closed. Vertices shown here are nodes of forest copies. Hubs, hub connectors, selected chords, and the contributions of every other family and raw forest join the same child graph J .

to agree. Replacing that segment by an edge with summed length and signed gradient consequently preserves both its gradient contribution and its length norm. $\square$

For an active f , subtracting its correction from its clipped vector leaves the clipped tree path minus the two legs of L_f , together with L_{Π_f} . The first part is a live forest vector with divergence only at hole endpoints and the two targets. The original chord-endpoint divergences cancel, also when a hole is adjacent to one of those endpoints. All remaining divergence points are terminals. The second part uses only selected chords and live paths between terminals. By Lemma 7.7 there is a child vector J_f , not necessarily a circulation, satisfying

$$\pi J_f = \pi(C_f^{\text{clip}} - B_f), \quad \|\ell^J J_f\|_1 \leq Z_* W_f. \quad (7.19)$$

For the length bound, the clipped tree part has length at most W_f , the two removed legs at most $16W_f$, and the routed selected path at most $17p_* W_f$. Their sum is at most $17(p_* + 1)W_f$. In the direct case the relevant cluster path is intact, so $C_f^{\text{clip}} = B_f$ and we take $J_f = 0$.

Likewise, the vector z_L in Lemma 7.5 compresses to a raw-forest child vector $\hat{z}_L$ with its length norm preserved. Indeed its description (7.13) is a combination of cell-tree paths whose endpoints were included in terminal item (iii). Clipping creates additional divergence only at hole endpoints. Compression is applied to the net vector z_L , whose divergence lies on terminals.

Figure 2 illustrates the compression of one net forest contribution to the child graph J .

7.7 The one-level reduction and the recursion

Lemma 7.8 compares the ratio of a current circulation with the best local correction score (Section 7.5) and the optimum of the child graph J (Section 7.6). Its hypothesis is the property of the path assignments Π_f stated in Section 7.5.

Lemma 7.8 (One-level ratio reduction). *Assume (7.2) and (7.3). In every state of an epoch at a level $i < d$, suppose the active logical path assignments use selected chords and have at most p_* hops. Let J be the child graph of Section 7.6 in this state, let $\text{OPT}(J)$ be the maximum circulation ratio of J , and let L_{best} be the largest local correction score (7.18), or zero if there is none. Then, for every nonzero circulation Δ on the current nonloops,*

$$\frac{|\gamma^\top \Delta|}{\|\ell \Delta\|_1} \leq 2Z_* A_* L_{\text{best}} + 12Z_* A_* \text{OPT}(J) \leq b^2 \max\{L_{\text{best}}, \text{OPT}(J)\}. \quad (7.20)$$

This conclusion is simultaneous for all current circulations and requires no independence between the state and the verified covers or path assignments.

Remark 7.9. Lemma 5.2 needs the scope asserted in the last sentence of Lemma 7.8 for adaptive clients, whose states may be chosen after, and depending on, the verified covers and path assignments.

Proof of Lemma 7.8. The proof subtracts the weighted local corrections $\alpha_f B_f$ from Δ and shows that the rest is the projection of a child circulation of length norm at most $12Z_* A_* \|\ell \Delta\|_1$; the gradient of Δ then splits between the local corrections, controlled by L_{best} , and this child circulation, controlled by $\text{OPT}(J)$.

The child vector. With the coefficients α_f and the vector z of Lemma 7.5, define on the child edges

$$\Xi_0 = \sum_f \alpha_f J_f - \hat{z}_L + \Delta_n, \quad (7.21)$$

with Δ_n carried by the hub-to-hub new edges. By (7.19) and (7.14), its projection is

$$\pi \Xi_0 = \Delta - \sum_f \alpha_f \pi B_f,$$

a circulation on the current level- i graph. The three terms of Ξ_0 are bounded separately. By (7.19), each J_f has length norm at most $Z_* W_f$, so, since $W_f \leq 2w_f$, (7.11) bounds the first term by $2Z_* A_* \|\ell \Delta_s\|_1$; compression preserves the length norm of z_L (Lemma 7.7), and this norm is at most $\|z\|_{\text{length}}$, which (7.12) bounds; the new edges carry Δ_n with their own lengths. These norm estimates give

$$\begin{aligned} \|\ell^J \Xi_0\|_1 &\leq 2Z_* A_* \|\ell \Delta_s\|_1 + A_* \|\ell \Delta_s\|_1 + \|\ell \Delta_n\|_1 \\ &\leq 4Z_* A_* \|\ell \Delta\|_1. \end{aligned} \quad (7.22)$$

Balancing at the hubs. The vector Ξ_0 need not balance at each compressed vertex. We route its imbalance there along the hub connector to the hub of its level- i image. The hub then balances because the sum of divergences over all compressed vertices with the same level- i image is zero. The total absolute imbalance at the compressed vertices is at most twice the sum of absolute coefficients of Ξ_0 , and hence at most twice its length norm, since every child edge has length at least one. The resulting child circulation Ξ therefore satisfies

$$\pi \Xi = \pi \Xi_0, \quad \|\ell^J \Xi\|_1 \leq 12Z_* A_* \|\ell \Delta\|_1. \quad (7.23)$$

Its gradient equals that of its projection because all hub connectors have zero gradient.

Combining the two parts. For the local part, (7.18) and (7.11) give

$$\left| \sum_f \alpha_f \gamma^\top \pi B_f \right| \leq L_{\text{best}} Z_* \sum_f |\alpha_f| W_f \leq 2Z_* A_* L_{\text{best}} \|\ell \Delta\|_1.$$

The child circulation contributes at most its norm times $\text{OPT}(J)$, with the same conclusion if it is zero. Adding these inequalities proves the first bound in (7.20). The second follows from $14Z_*A_* \leq 20Z_*A_* \leq b^2$, by (7.3).

Adaptive scope. Every distribution, coupling, coefficient, and comparison vector in this proof is existential. The algorithm constructs only integral graphs, terminal sets, path assignments, and rational scores. Once the covers and assignments have their verified properties, the argument applies to any state and any circulation chosen afterwards. This proves the asserted adaptive scope. $\square$

Corollary 7.10 (Approximation ratio at every recursion level). *Assume (7.2) and (7.3), and assume the path assignments and public forest lifting of Section 8. The cycle data structure at level i returns a certified score at least*

$$\frac{1}{2}b^{-2(d-i)} \text{OPT}_i. \quad (7.24)$$

At the outer level this is at least $M^{-10/a} \text{OPT}$.

Proof. At the bottom level the bound is given by Lemma 7.1. At a higher level the answer is the best of the local correction scores, the exact current-loop scores, and the recursive child score. Lemma 8.5 preserves the child gradient under lifting and does not increase raw actual length, so the child's certified score remains valid. If its approximation factor is $\eta \leq 1$, the maximum of its score and L_{best} is at least $\eta \max\{L_{\text{best}}, \text{OPT}(J)\}$. By Lemma 7.8, this maximum is at least ηb^{-2} times the ratio of every nonzero circulation on the current nonloops, so the approximation factor decreases by at most a factor b^2 at this level. Exact loop candidates give the same conclusion for the full graph. This proves (7.24) by induction.

Since $d = 2a - 4$, the outer factor is $\frac{1}{2}M^{-(4a-8)/a^2}$, which exceeds $M^{-10/a}$ in the checked range. When the optimum is zero, the bound is zero, and a zero score meets it. Section 8 bounds the representation size, the explicit forest logs, and the total operation cost; the quality argument itself makes a single recursive child call at each level. $\square$

8 Maintaining and exposing the cycle data structure

This section implements the one-level reduction of Section 7 and thereby completes the proof of Lemma 5.2. There are two quantities to control: the work performed locally and the number of edits passed to the one recursive child. The latter must be smaller, since it is multiplied through the recursion. We also construct the public forest, with explicit persistent edge identities, so that a child path can be lifted without traversing all its edges. Recursive vertex and edge sparsification for dynamic minimum-ratio cycles is developed in [4, Sections 5 and 7]. Here we establish the snapshot, edit-count, and forest-lifetime guarantees required by Lemma 5.2.

The setting is that of Section 7. Every lemma of this section except Lemma 8.10 assumes four displays: the parameters (7.1), where $M = 2^u$ and u is a positive multiple of $2a^2$; the parameters (7.2); the range (5.8) for M ; and the checks (7.3). Lemma 8.10 concerns these checks and assumes only the first three. Except in Lemmas 8.9 and 8.10, which cover all levels, we work at one recursion level $i < d$. Input slots and input edges are those of the level- i input graph, as opposed to their copies in the frozen forests, in J , and in the public forest. The following objects of Section 7 are used.

- *Forests.* The frozen forests T^0 of Lemma 7.3, with their OUT and IN links; the raw forests H_j ; the width buckets W of (7.7); and the holes, which are the tree edges deleted when a frozen actual edge dies.

- *Portals and legs.* The portal grid of Lemma 7.6 and its pockets, which are the components off the grid. The rules stated after Lemma 7.6 classify chords as regular or nonregular, group nonregular chords into depth-band clusters, and assign to each incidence a leg ending at its target.
- *Classes and logical paths.* The classes and the active chords defined after Lemma 7.6, and the logical paths Π_f and corrections B_f of (7.16).
- *The child graph.* The graph J of Section 7.6, with its terminals, compressed segments, hubs, and projection π .

Section 8.1 maintains the classes and legs, Section 8.2 maintains the paths Π_f through a hierarchy of snapshots, Section 8.3 maintains J , Section 8.4 builds the public forest through which child paths are lifted, and Sections 8.5 and 8.6 bound its changes, the total work, and the failure probability.

As in Section 2, asymptotic statements are made with a and σ fixed; in this section *eventually* means for all sufficiently large M at fixed parameters. Throughout this section a *logarithmic factor* means $\log^{O_a(1)}(2 + M)$. Different occurrences may denote different factors. In particular P_*, p_* , the number of forest copies, and the number of width buckets are logarithmic factors. Work bounds keep logarithmic factors separate from powers of b . Recall from (7.4) that at recursion level $i < d$ an epoch has $h_i = n_i/k$ single input edits. Queries do not advance this epoch counter.

8.1 Maintaining endpoint classes

For each frozen tree T^0 and width bucket W we use one logical graph, the *tree-bucket graph* of (T^0, W) . Its vertices are the classes formed for the chords of T^0 in bucket W , and the logical paths Π_f of these chords are paths in it. It groups all components of that tree, so neither rebuilding nor taking the best score incurs a separate overhead for every component. Classes are identified by *class labels*, and every split creates fresh class labels for both sides (Section 7). Let D_T bound the combined tree and chord degree at a frozen tree vertex, and set

$$D_\ell = (k + 1)(D_T + 1)^2. \quad (8.1)$$

This is the logical degree bound; the subscript ℓ stands for “logical”. The maximum can be measured when the tree is built. Cell-tree edges use actual incidences; a null pair chord only joins copies of the same input slot in different partitions of one layer. At a vertex $V_j(v)$ there are only the appropriate OUT and IN links, and, at $V_0(v)$, the old actual chord incidences. Hence D_T is bounded by the input degree plus $O(K_h)$ and is a logarithmic factor whenever the input degree is.

The changes to the classes caused by input edits are counted in *logical edit units*: a unit is a class split, a pair of fragment-class births at a cluster activation, or a single chord death. The next lemma maintains the classes and legs of every tree-bucket graph and bounds the number of units; this number controls the rebuild work in Lemma 8.2 and the support in Lemma 8.3.

Lemma 8.1 (Maintenance of endpoint classes). *Assume (7.1), (7.2), (7.3), and (5.8). The leg assignments of regular and nonregular chords defined after Lemma 7.6 can be maintained with logical degree at most D_ℓ . One input edit induces a number of logical edit units, across all tree-bucket graphs, that is at most a logarithmic factor. Each unit involves at most $O(D_\ell)$ incidence work. Every change to the leg of a surviving active incidence entails a split of its class.*

Proof. Regular and nonregular chords are treated separately; in each case a cut splits one class into two classes with fresh class labels. The units are then counted, and finally an implementation within the stated work is described.

Regular incidences. Recall from the rule for regular chords after Lemma 7.6 that a chord is regular when its full frozen tree path touches the initial portal grid, and that each of its incidences uses the leg toward the nearest portal. The nearest-portal regions are connected: along a shortest tree path toward the chosen portal, the distance comparison with every competing portal persists, as does the global index tie rule. A region consists of its portal and parts of the adjacent pockets, and therefore has at most $1 + kD_T$ vertices.

The target of a regular incidence is the incidence-side endpoint of the first hole on its fixed path toward its original portal, or the portal if there is no such hole. In each width bucket, all current legs crossing a newly cut edge belong to one old class. Indeed that edge lies in one nearest-portal region, and their earlier truncation, if present, is the first hole on their common suffix toward the portal. Split this class by moving the crossing incidences to the incidence-side endpoint of the new cut. This target is determined by the newly cut parent edge toward the portal, so no merging is needed. Both children receive fresh class labels, also when one is empty or one retains the old target.

Nonregular chords. For a nonregular chord, the classes come from the two depth-band clusterings defined for nonregular chords after Lemma 7.6. The first interior cut activates the cluster and creates two fresh fragment classes. Every newly active chord has both incidences in these new classes, including chords whose two incidences lie on the same side. Thereafter cuts split fragment classes, again with fresh class labels on both sides. Classes in different clusters remain distinct even if their targets are the same tree node. Each cluster is contained in an initial pocket. Its number of incidences, and that of any regular region, is bounded by (8.1). The two incidences of a logical loop are counted separately.

Number of units. An old actual-edge deletion creates a number of holes and chord deaths in all copies that is at most a logarithmic factor. In one width bucket, a hole is interior to at most one cluster in each of the two shifted band partitions, and it causes at most the regular split just described. A newly inserted input edge creates no frozen chord. These facts give the asserted number of units.

Implementation. The units are listed and processed as follows. At freezing time, compute tree ancestor tables, length and oriented-gradient prefix sums, portal mark counts, nearest portals, pockets, and cluster membership. The band index at a pocket vertex is an integer quotient of its weighted depth; connected components of equal band index give the clusters. No record is allocated for an empty band. These computations take near-linear tree work up to logarithmic factors and determine the relevant path sums without scanning each fundamental path. Initial regular legs have at most $k + 1$ tree edges, so their membership lists on tree edges can also be built explicitly. On a hole, inspect the region-bounded lists, discard dead incidences and legs already stopped before this hole, and update the remaining class lists. A cluster cut can inspect the whole small cluster and its chord incidences.

Before activation, keep each direct fundamental-cycle score in a heap, computed from its frozen sums. Delete its entry when the chord dies or its cluster activates. For an input death, the removals of the dead chords from the input graph can be processed before the splits and activations that it causes. Afterwards every surviving active leg is the unique live tree path to its stored target. A leg whose class has not split retains that path and its gradient. This also proves the last assertion. $\square$

8.2 Maintaining the logical paths

The logical paths Π_f required by Lemma 7.8 are computed with the routing sparsifier of Lemma 6.1. In the terminology of that lemma, the active chords whose paths are assigned are the *demands*, and the chords used by the returned paths form the *support*. We maintain these assignments by a hierarchy of snapshots. The schedule is described by the index t of an input edit within the epoch,

the number z_0 of levels, the periods s_j , the level j rebuilt at edit t , and its parent time t_0 . The invariants (8.4) below bound the hops $p(j)$ and the class visits $C(j)$ of a level- j snapshot in terms of P_* from (7.1) and D_ℓ from (8.1).

At edit h_i rebuild the whole outer epoch on the resulting input graph, without updating the obsolete child. For edits $1 \leq t < h_i$, put

$$z_0 = \lceil \log_b h_i \rceil \leq a^2, \quad s_j = b^{z_0-j} \quad (0 \leq j \leq z_0). \quad (8.2)$$

These are integral powers of two under the divisibility assumption on u stated after (7.1). Level zero is the initial frozen assignment. At edit t , choose the smallest $j \geq 1$ for which $s_j \mid t$. Rebuild from the parent snapshot at

$$t_0 = \left\lfloor \frac{t}{s_{j-1}} \right\rfloor s_{j-1} < t, \quad t - t_0 \leq b s_j, \quad (8.3)$$

and let all finer levels share the resulting snapshot. Minimality of j ensures $s_{j-1} \nmid t$, so the strict inequality in (8.3) holds. If a coarser level was rebuilt at t_0 , its shared snapshot also supplies the indicated parent.

An assigned path is simple; an empty path visits its sole vertex once. The invariants for a level- j snapshot are

$$p(j) = (10P_*)^{j+1}, \quad C(j) = D_\ell(100P_*)^{j+1}, \quad (8.4)$$

bounding its path hops and the number of assigned paths visiting any one class, respectively. The factors 10 and 100 are convenient choices, large enough that the hop bound (8.8) and the visit bound (8.9) reproduce $p(j)$ and $C(j)$ from $p(j-1)$ and $C(j-1)$.

At initialization only regular chords are active and their classes are portals. The initial snapshot is computed by Lemma 6.1, applied with endpoint incidence counts as resources, followed by simplification of the returned walks. For nonempty walks, vertex visits are bounded by the endpoint incidence counts on their traversed edges; empty paths add at most the demand degree. This proves (8.4) for $j = 0$. All finer levels initially share this snapshot.

A selection is recorded by the identifier of the chord's input edge. A selected chord remains in support for the rest of the epoch while alive, even if its logical endpoint classes later split. Repeated selection stores it just once in the child graph.

The next lemma is the inductive step of the hierarchy: a rebuild at edit t starts from the parent snapshot at t_0 , and its work and its new support are proportional to the number of logical edit units in the edits after t_0 .

Lemma 8.2 (Rebuilding a snapshot from its parent). *Assume (7.1), (7.2), (7.3), and (5.8). Let a level- j snapshot of one tree-bucket graph, $1 \leq j \leq z_0$, be rebuilt at edit t from its parent snapshot at t_0 , as in (8.3), and suppose the edits in the interval $(t_0, t]$ contain E_0 logical edit units in that tree-bucket graph. The rebuild can restore (8.4) using one instance of the routing sparsifier of Lemma 6.1 on $O(E_0)$ incident class labels. It adds at most $O(P_*E_0)$ selected chords to the support. The work is*

$$O(1 + D_\ell E_0) b M^{o_a(1)}, \quad (8.5)$$

in addition to scanning the interval's edit log with logarithmic overhead per edit and explicitly handled incidence.

Proof. The rebuild keeps every old assigned path that avoids the class labels marked by the edits of the interval. For every other surviving demand it keeps the old prefix up to the first marked

label and the old suffix from the last one, and it routes the remaining middle part by one call of the routing sparsifier on a core graph whose vertices are the anchors defined below. The proof first identifies the marked labels and the anchors, then shows that unmarked paths keep their corrections, reinterprets the prefixes and suffixes of marked paths as current paths, bounds the resources of the core graph, derives both invariants from the sparsifier's guarantees, and finally describes the persistent representation and bounds the work.

Marked labels and anchors. Mark class labels of the snapshot at t_0 as follows. Every split marks its ancestor at t_0 , if one exists; every chord death marks the ancestors of both endpoint classes. Let Y be the marked set. Then $|Y| \leq 2E_0$. The *anchors* are all current descendants of Y and all current classes born from activations after t_0 , including their later descendants. There are $O(E_0)$ anchors: initially marked labels contribute at most $2E_0$, and every subsequent split or activation adds only constantly many class labels. Ancestors absent at t_0 belong to the latter activation families.

Unmarked paths keep their corrections. Mark every old assigned path that visits Y , including at an endpoint. In particular the path of a deleted demand is marked, as is any path using a deleted support edge. Every unmarked path still uses the same live oriented chords on unchanged labels. Moreover its demand and all support edges on the path retain the same legs and leg gradients, by Lemma 8.1. Thus its correction $B_f = L_f - L_{\Pi_f}$ has exactly the stored gradient. A hole outside these legs can change the clipped vector C_f^{clip} of Section 7.3, but does not change this correction. Inactive direct-cycle candidates are handled separately by their activation lists.

Prefixes and suffixes of marked paths. In this stage the oriented chords of an old path, as input edges, stay fixed, while their endpoints pass from labels of the snapshot at t_0 to current classes: an unsplit label is still a current class, and a marked label is replaced by the current descendant that the incidence of the adjacent edge determines. Consider a marked old path whose demand e survives, and its prefix from the tail to the first marked label. Every preceding label is unsplit. Every edge of this prefix is live: if one had died, its earlier endpoint's old label would also have been marked. The final edge now ends at a current descendant of the first marked label. Interpreting the same oriented edges therefore gives a current path A_e from the current tail to an anchor. If the old path starts marked, A_e is the empty path at the current tail. Reversing this argument gives a suffix Z_e from a current anchor to the current head at the last marked label. Repeated splits cause no ambiguity, since the current incidence of the last prefix edge specifies its current descendant. For newly activated demands use empty A_e, Z_e ; both endpoints are anchors by the activation rule.

The core graph and its resources. For each such demand put an edge in a core graph on the anchors, associated with the direct logical walk

$$-A_e + e - Z_e. \quad (8.6)$$

We call e the *central chord* of this core edge. Its orientation is from the end of A_e to the beginning of Z_e ; a minus reverses a path. For every current class of the tree-bucket graph use its number of visits on this direct walk as a resource count. Each count is at most two. Indeed the prefix and suffix each come from a simple old path, with at most their final marked label replaced by a descendant; the two resulting vertex lists concatenate across e . This also covers empty prefixes, suffixes, and a loop demand. The total resource count at a class, over all direct walks, is at most

$$C_0 = 2C(j-1) + D_\ell. \quad (8.7)$$

Prefix visits and suffix visits of old demands are bounded separately by the visits at that class's old ancestor; newly activated endpoint incidences are bounded by its current degree. The same bound controls the core demand degree, because its endpoint incidences are among these resource visits.

The sparsifier call and the invariants. The rebuild applies Lemma 6.1 to the core with the resources just defined; the counts lie in $\{1, 2\}$, and C_0 of (8.7) bounds both the degrees and the total resource counts. It lifts the assigned walks by substituting (8.6), and for demand e it prepends A_e and appends Z_e . It then simplifies the resulting walk to a simple path, erasing the whole walk if its endpoints coincide. Cycle erasure never increases hops or resource visits. The hop count before simplification is at most

$$2p(j-1) + P_*(2p(j-1) + 1) \leq p(j). \quad (8.8)$$

The substituted sparsifier walks contribute at most P_*C_0 visits at each class. The added prefixes and suffixes contribute at most $2C(j-1) + D_\ell$, including the empty paths; the unchanged assignments add at most $C(j-1)$. Thus a valid upper bound is

$$P_*(2C(j-1) + D_\ell) + 3C(j-1) + D_\ell \leq 100P_*C(j-1) = C(j). \quad (8.9)$$

All edges in the prefixes and suffixes were already live support edges. Only the central chords of the selected core edges must be added to support. The core has $O(E_0)$ incident class labels, regardless of how many old paths met Y , so the sparsifier selects at most $O(P_*E_0)$ such chords. When $E_0 = 0$ the old snapshot can simply be retained.

Persistent representation of snapshots. Snapshots are stored persistently. Each held snapshot consists of three parts: immutable balanced-dictionary roots for the paths, keyed by demand; an index from each class to the paths visiting it; and an ordered collection of correction-score keys. Persistence duplicates only the search paths of modified dictionary entries. A rebuild begins with the parent roots, replaces marked or dead records, and adds the newly active records. The class index enumerates at most $C(j-1)|Y|$ old path occurrences before their lists are read; a dictionary removes duplicates. Activation lists enumerate new demands. Every such path has logarithmic-factor length, and the number of newly activated demands is at most $O(D_\ell E_0)$.

Store split-parent pointers with the index of the edit at which each class was born, and jump pointers. To find the ancestor at t_0 , follow this history to the class alive after edit t_0 . All splits within one edit belong to that edit and snapshots are taken only after settlement, so, even when birth indices are equal, the class found is never one that has received only some of the splits of one edit. The input-edge identifiers of the chords, the two current incidence classes, and their targets are explicit records. They determine the oriented lifted prefixes and suffixes. Frozen tree prefix data gives each live leg's gradient and length; summing over the new short logical paths recomputes their correction-score keys. Unmodified keys remain valid by the unmarked path argument.

Work and prerequisites of the sparsifier. Building the direct lists and the sparsifier's resource lists, simplifying its output, and changing the indices costs $O(1 + D_\ell E_0)bM^{o_a(1)}$ by Lemma 6.1. Comparing the maintained best keys over the tree-bucket graphs, whose number is a logarithmic factor, has only logarithmic overhead. The sparsifier's input prerequisites can be checked: twice the demand count and the nonzero resource-list length must each be at most M^2 . They hold eventually, since even all frozen chords number at most $O(n_i)$ times a logarithmic factor, and every direct walk has logarithmic-factor length. This proves the work assertion and both snapshot invariants. $\square$

Summing Lemma 8.2 over the rebuilds of all levels gives the cumulative bounds for an epoch.

Lemma 8.3 (Cumulative assignment work and support). *Assume (7.1), (7.2), (7.3), and (5.8). The number of support chords used by the initial assignments, that is, by the level-zero snapshot, is at most h_i times a logarithmic factor. On every epoch prefix of $t' \leq h_i$ edits, the number of additional support selections is at most bt' times a logarithmic factor, and local assignment work is $t'M^{O(1/a)+o_a(1)}$ after initialization. Initialization costs $n_iM^{O(1/a)+o_a(1)}$.*

Proof. Initially the number of portal classes, summed over all tree-bucket graphs, is at most (n_i/k) times a logarithmic factor. The sparsifier selects at most P_* times this many chords. At inner level j there are at most $\lfloor t'/s_j \rfloor$ rebuilds, each using an interval of at most bs_j edits. By Lemma 8.1, the sum of their interval edit counts across all tree-bucket graphs is at most bs_j times a logarithmic factor per rebuild. Consequently the cumulative support additions at that level are at most bt' times a logarithmic factor by Lemma 8.2. There are at most a^2 inner levels. Summing (8.5) similarly bounds the local work by $t'D_\ell b^2$ times a logarithmic factor, in addition to smaller constant-per-rebuild and log-scanning costs. Since $D_\ell = (k+1)\log^{O_a(1)}(2+M)$, this has the asserted exponent with an absolute coefficient. Initial tree preparation, the explicit short regular-leg lists, and initial sparsifier calls fit the stated initialization bound. A removal of an input edge lists the deaths of selected copies in logarithmic-factor work. Keeping earlier live selections therefore needs no support selections beyond those counted above. $\square$

Every level- j snapshot satisfies (8.4): for $j = 0$ by the initial routing above, and for $j \geq 1$ by Lemma 8.2, whose hop bound is (8.8). Since $j \leq z_0 \leq a^2$ by (8.2), the assigned paths therefore have at most

$$p(j) \leq (10P_*)^{a^2+1} = p_* \quad (8.10)$$

hops, with p_* as in (7.2).

8.3 Small changes to the compressed child graph

This subsection bounds the number of edits needed to maintain the compressed representation of the selected-support paths in the child graph J of Section 7.6.

Lemma 8.4 (Compressed graph edits). *Assume (7.1), (7.2), (7.3), and (5.8). The single child graph J of Section 7.6 can be initialized nonempty. Throughout an epoch the number of active child slots and edges it uses, including child slots assigned earlier in that epoch, is at most $h_i b$ times a logarithmic factor. On an epoch prefix of t' further input edits it receives a number of individual edits at most bt' times a logarithmic factor. Its degree is a logarithmic factor, and its integral prices obey the level- $(i+1)$ bounds. The local construction work per parent edit is $M^{O(1/a)+o_a(1)}$ amortized.*

Proof. The terminals of J are promoted one at a time; the proof counts the promotions, shows that each promotion changes only $O(1)$ compressed vertices and edges with local work, and then checks the capacity, degree, and price bounds.

Number of promotions. Initial terminal promotions number at most h_i times a logarithmic factor. Subsequent promotions come from hole endpoints, new class targets, endpoints of selected support chords, and, for new incidences at input slots, the endpoints of raw-forest cell-tree paths that the terminal set of Section 7.6 includes for these input slots. The first three counts follow from Lemmas 8.1 and 8.3. For the fourth, enumerate the $K_h^{O(r)}$ possible sequences of partition options from the given input slot. At each step look up the entering copy and its frozen cell root in stored tables. These lookups list the needed endpoints without traversing the raw legs, in work at most a logarithmic factor per incidence. Thus the cumulative additional promotions on a prefix number at most bt' times a logarithmic factor. Retain all promoted terminals until the epoch ends.

Maintaining the Steiner subtrees. Build each initial Steiner subtree in bulk, using subtree terminal counts. To promote one additional tree node, split its current compressed segment if it is already in the represented subtree. If it is outside, explore its hanging component to the unique attachment and add the joining path, splitting the segment at that attachment if needed. If the live component has no represented subtree, create an isolated terminal. The explored exterior component contains no initial portal, and hence lies within one original pocket of fewer than k

vertices. Existing segments have at most $k + 1$ constituent edges. Membership and segment pointers on these nodes, together with local forest adjacency, therefore suffice for $O(k + 1)$ times degree and logarithmic-factor work. Only $O(1)$ compressed vertices and edges change per promotion. No whole live component needs to be relabeled.

Before cutting a frozen tree edge, promote both endpoints. The edge is now itself a compressed segment and can be deleted. On either side, every other represented edge still separates terminals: a terminal formerly reached across the deleted edge is replaced for this purpose by its now-terminal endpoint. Thus there is no pruning or merging cascade. Keep each segment's full constituent list with orientations. A changed segment or changed realization is removed and replaced by a fresh edge, with deletion first.

Child slots, hubs, and capacity. Assign each represented frozen tree or raw-forest node its own child slot, without reuse in this invocation, and assign at most one hub per input slot. A selected chord remains attached at its original tree-node endpoints. Its logical leg targets can therefore change in bulk without migrating its child edge. New actual nonloops are inserted directly between hubs and removed at their deaths; actual loops have already been handled by the heap on current loops (Section 7.2). These rules establish the claimed size and edit counts.

Reserve $\lfloor n_{i+1}/2 \rfloor$ child slots, with unused child slots isolated, and check both assignment overflow and total child slots plus current edges against the child's capacity. These checks hold eventually, because

$$n_{i+1} = h_i M^{1/(2a)}, \quad b = M^{1/a^2}, \quad \frac{1}{a^2} < \frac{1}{2a}. \quad (8.11)$$

The bounds above on active child slots and edges are consequently eventually smaller than either reserved half.

Degrees. At a compressed node, segment degree is at most its frozen tree degree; add its chord degree and one hub connector. At a hub, the connector degree is bounded by the number of frozen tree and raw-forest copies of its input slot, plus the current new actual incidences. Both are logarithmic factors. Deleting old segments before adding replacements preserves these degree bounds even during the child edit sequence. This proves the degree hypothesis inductively from the degree-three outer input.

Prices and local work. A segment sums prices over at most $k + 1$ edges, so, as in Section 7.6, its positive integral length and absolute gradient are at most $(k + 1)M^{G_i} \leq M^{G_i+2} = M^{G_{i+1}}$ whenever $k + 1 \leq M^2$. Other child prices are inherited or are $(\ell, \gamma) = (1, 0)$. The local work includes an $O(k + 1)$ factor per promotion or changed segment and the assignment work already bounded. Its exponent is therefore $O(1/a)$ with an absolute coefficient. $\square$

8.4 Prebuilt path copies and the lifting of child forest paths

This representation uses the flat-embedding viewpoint of [4, Definition 2.3 and Theorem 3.5]. The preallocation and reassignment of unused tree copies follow the flat-forest lifting construction of Kyng, Meierhans, and Probst Gutenberg [21, Section 3.3, Lemma 3.21 and Algorithm 5]. Their construction already provides path-length domination and bounds on changes to the lifted forest. Here the copied trees are copies of the constituent paths of compressed segments, which we call *path copies*. The lifetimes of mapped edges and the transaction rule of Section 8.5 preserve the step primitive's signed and absolute accumulations when an unused copy slot is reassigned.

Put

$$D_i = b^{2(d-i)} \quad (0 \leq i \leq d). \quad (8.12)$$

These are the occurrence bounds of the recursion levels. The bottom forest is empty. At a higher level include full live copies of all T^0 and raw H_j forests as separate public components, with their

holes and their actual or null mappings. They represent local correction paths and individual compressed segments. The following additional components, disjoint from the full copies, represent the public forest F^J of the recursive child, which we call the *child forest*.

For each active child slot q , including hubs, write $\varphi(q)$ for its input image. Prebuild a pool of D_{i+1} copy slots. A copy slot has a center mapping to $\varphi(q)$. For each tree segment of J incident to q , attach an independent copy of its entire constituent path, oriented outward from q , with a fresh far tip and all constituent vertex and edge maps; this is the path copy of that segment in the copy slot. Different path copies in one copy slot meet only at its center; different copy slots are disjoint. Retain the whole pool, including unassigned, or *dormant*, copy slots. Assign a distinct copy slot to each occurrence of q in F^J . Occurrences over an inactive child slot are ignored publicly: such a child slot is isolated in J , so all its forest components have only null edges and project to that one child slot.

Add one connector for every child forest edge in the active part:

1. If its map is a segment qz , choose a fixed endpoint side, say q . Use the corresponding path copy in the copy slot assigned to its occurrence over q , and add a null connector from that path copy's far tip to the center assigned to the other occurrence over z .
2. If its map is a nonsegment child edge, join the two assigned centers with that edge's single actual-edge or null map.
3. If the child forest edge itself is null, join its assigned centers with a null connector.

The endpoints of each null connector have equal input images. Centers have indexed identifiers, and tips are indexed by copy slot and incident segment identifier. Connector endpoints can thus be looked up without scanning a path copy.

Lemma 8.5 (Lifting child forest paths). *Assume (7.1), (7.2), (7.3), and (5.8). The public forest of Section 8.4, consisting of the full copies, the pools of copy slots, and the connectors added for the child forest edges, is a forest whose edge maps are as required in Lemma 5.2(i): every edge maps to a current actual edge, with specified endpoint correspondence, or is null and joins equal input images. A child forest-path piece lifts to at most one public forest-path piece, preserving its signed projection and with parent raw length at most its child raw length. An individual child-edge piece lifts to one path or one actual-edge piece, or can be omitted if null. The lifted sum of a child circulation is a circulation with the same gradient. The number of returned pieces is $M^{O(1/a)+o_a(1)}$.*

Proof. Contracting copy slots recovers the child forest, so the construction is a forest; the expansion of a child path is then reduced, by cancelling backtracking, to the forest path with the same signed projection and no larger raw length.

Contracting every copy slot to its center gives the connector quotient; on assigned copy slots it is exactly the active subforest of F^J . Dormant copy slots have no connectors. A cycle in the expanded graph would give a cycle in this quotient, since each contracted part is itself a tree. Together with the disjoint full copies, the result is a forest.

Expanding each edge on an oriented child path by the prescribed path copy and connector, or by its nonsegment connector, gives a concatenation with the expanded signed projection of that child path. Segment length is the sum of its constituent parent lengths; null child edges contribute zero parent raw length, and other child null realizations can only decrease it. Thus the concatenation's parent raw length is at most the sum of child-edge lengths counted with their traversal multiplicities.

This concatenation may backtrack, even when the child path is simple, because several connectors can use the same path copy in one copy slot. In a forest, deleting a consecutive edge and its reverse preserves the signed count of every oriented edge and decreases nonnegative raw length. Repeating this deletion leaves the unique forest path between the assigned endpoint centers. It therefore has

exactly the needed projection and no larger raw length. We return its endpoints; the cancellations are not performed by expanding the walk. If the child path has an inactive endpoint, its whole component is one of the ignored null components, so the piece can be omitted.

For a piece given as a single child segment, use its endpoints in the appropriate full live forest copy. For a nonsegment piece, return its actual edge or omit its null realization. Each edge expansion has divergence equal to its mapped endpoint divergence. Summing a child circulation therefore gives zero divergence in the input graph and preserves its gradient. In particular a positive child score gives positive gradient magnitude after lifting and therefore positive parent raw length. Its score remains a valid lower bound for the lifted ratio.

These conversions use only endpoint and copy-slot lookups; they neither traverse nor reprice a long lifted path. A local correction uses $O(p_*)$ pieces in full copies and individual actual chords, with opposite traversals explicitly reversed and null chords omitted. An actual loop uses one piece. A child answer gives at most as many parent pieces as it had child pieces. At each level we return one best candidate. The bottom answer has at most $n_d = M^{2/a}$ pieces. Taking the maximum of these bounds proves the last assertion. Every listed piece has unit multiplicity; repetitions, if present, are listed and counted. $\square$

Figure 3 shows a lift through a shared path copy and a dormant copy slot whose internal edge lifetimes persist through reassignment.

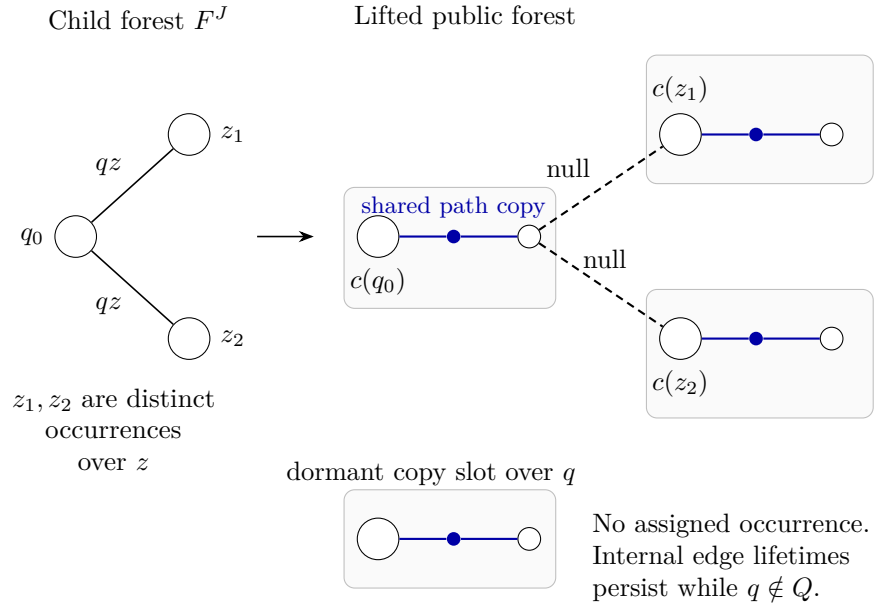


Figure 3: A lift through a shared path copy in Lemma 8.5. The vertex q_0 is an occurrence over the child slot q , and $c(\cdot)$ denotes the center of the copy slot assigned to an occurrence. Both child edges map to the same segment qz , but their endpoint occurrences over z use different copy slots. Each edge adds one null connector from the far tip of the chosen path copy to the other assigned center. Contracting all copy slots recovers the child forest. In the child path from z_1 to z_2 , the two traversals of the shared path copy cancel when the concatenated realization is reduced to the unique forest path; signed projection is preserved and raw length cannot increase. Dormant copy slots retain their internal edges, so reassignment at a child slot outside Q (Lemma 8.7) changes assignments and connectors without rebuilding those edges.

Lemma 8.6 (Occurrence multiplicity). *Assume (7.1), (7.2), (7.3), and (5.8). For all sufficiently large M at fixed parameters, level i has at most D_i public forest vertices over each input slot. In particular the outer multiplicity is at most $M^{10/a}/4$.*

Proof. Full copies contribute a logarithmic factor. For one compressed frozen forest, segment interiors are disjoint. Thus a node in such an interior occurs in at most two path copies per pool index, one from each endpoint of its segment. At a compressed endpoint, its own centers and the far tips of path copies from adjacent endpoints contribute at most one plus its tree degree per pool index. These bounds include dormant copy slots. The number of frozen-tree and raw-forest copies of an input slot is at most a logarithmic factor, and the input slot has at most one hub. Consequently its total multiplicity is at most

$$(1 + D_{i+1}) \log^{O_a(1)}(2 + M). \quad (8.13)$$

The bound (8.13) is independent of the segment lengths. The b^2 gap between D_i and D_{i+1} eventually absorbs the logarithmic factor. Induction starts with the empty bottom forest. Finally

$$D_0 = M^{(4a-8)/a^2} \leq M^{10/a}/4$$

eventually. Both the inductive multiplicities and the last inequality may be checked as deterministic prerequisites. $\square$

8.5 Explicit recourse and persistent edge lifetimes

An occurrence or forest edge has an immutable lifetime identifier. Changing an occurrence's label, an edge's endpoints, or its map requires a replacement with a fresh identifier. A vertex death explicitly logs all incident edge removals first. Additions use present endpoints. The client retains old input labels, prices, and maps until all old public removals have been processed, then installs the new input state and public additions. No path call occurs inside this transaction. A whole epoch boundary can remove the old public forest and install the new one with these rules.

Lemma 8.7 (Forest recourse). *Assume (7.1), (7.2), (7.3), and (5.8). For one parent input edit within an epoch, let Q be the set of child slots assigned or changed, or incident to a changed child edge or changed segment realization. Apart from local full-copy edits, the public forest can be updated in*

$$|Q|D_{i+1}(k+1) \log^{O_a(1)}(2 + M) + \mathcal{L} \log^{O_a(1)}(2 + M) \quad (8.14)$$

work and public log records, where $\mathcal{L}$ is the number of child log records processed. Retained path-copy edges at child slots outside Q keep their lifetimes and maps, even if their copy slot is reassigned to a different child occurrence.

Proof. Child slots in Q are rebuilt from scratch, which gives the first term of (8.14). At all other child slots the pools and retained assignments persist, and every change has its own child log record, processed with logarithmic overhead, which gives the second term. The parent publishes the result as one old-to-final transaction.

The child batch and its logs. First compute the batch of individual J edits for this parent operation, without a child query, and determine Q . Its size is bounded by a constant times the exogenous child edit and assignment counts of Lemma 8.4. Retain access to the old occurrence assignments and old child forest records. Run the child edit sequence while maintaining a *passive* duplicate of its logs, indexed by lifetime identifier and child slot, with individual incidence lists. Track which records were added or removed. Each log changes only its own dictionary and incidence-list

entries and costs logarithmic overhead. In particular, deleting an occurrence needs no separate degree scan: its incident edge deletions have their own records. Persistent roots or saved old records retain the old information until the parent transaction is complete.

Child slots in Q . At child slots in Q , remove all old pools of copy slots and construct the final active pools with final occurrence assignments. Sweep their old and final incident child edges as well, to remove and reconstruct the corresponding connectors. To bound these sweeps, fix a child slot q in either the old or the final state and let $N_J(q)$ be its set of *distinct* other neighbors. Every child forest edge incident to an occurrence over q belongs to the forest induced on the occurrence sets over $\{q\} \cup N_J(q)$. That forest has at most

$$D_{i+1}(1 + |N_J(q)|) \leq D_{i+1}(1 + \deg_J(q)) \quad (8.15)$$

vertices, and hence no more than this many edges. Parallel child-graph edges only increase the degree bound on the right of (8.15). A child-null edge has both ends over q and is already included. This proves the bound separately for the old and final sweeps; old lists can be saved before applying the child batch. Constructing each path copy takes at most $k + 1$ constituent steps. The degree bound on J now gives the first term of (8.14).

Child slots outside Q . At every active child slot outside Q , retain the pools of copy slots and the assignments of retained occurrences. First free assignments of deleted old occurrences, then assign each final new occurrence from a free-slot stack. There is sufficient space because the final occurrence count is at most D_{i+1} . Remove the old connectors for changed child edge records and add the necessary final ones; also handle the connectors found in the sweeps at the child slots in Q . Additions can wait until all final assignments exist. Deduplicate swept records by identifier. Transient child records that are neither old nor final need not be published by the parent, although their individual processing is counted.

If a child edge and both its assignments survive unchanged at child slots outside Q , its connector retains the same map. Every other change at these child slots has an explicit child edge record: an occurrence replacement logs its incident edge removals and new edges, and an edge endpoint or map change is itself a replacement. Thus the remaining work is logarithmic per child record, giving the second term in (8.14). A high-degree child occurrence can have many incident records, but each is processed once, without a degree sweep per record. Child lists over inactive child slots are likewise kept passively until activation, which places their child slot in Q .

Removals and the order of insertions. Full-copy hole edits are performed separately. Every public map to a dying parent input edge is removed: it lies in a full copy that receives the cut, in a path copy of a changed or removed segment whose endpoints are in Q , or in a connector realizing a removed nonsegment child edge. Remove connectors before their old pool vertices, and all old edges before their vertex deaths. After these removals the retained public graph is a subforest of the final graph, so its final pools, vertices, and connectors can be inserted in a valid order.

The child's view and the single transaction. Three child states occur: the old state, the intermediate states of the child edit sequence, and the final state. During this computation the child sees only its abstract child slots, edges, and integral prices. An intermediate child state may still contain the old realization of a parent segment whose input edge is about to die. Its abstract graph remains an in-range input, and it never asks for data of the parent's input graph. Only the final child state must be the reduction of the new parent input. The parent therefore publishes one old-to-final transaction, using retained old data for all removals, rather than a lifted public state for every intermediate child edit.

Persistence of dormant copy slots. Finally, a dormant copy slot at a child slot outside Q retains all its internal edges when assigned to another occurrence. Their identity is determined by the child

slot, pool index, and incident segment realization, not by the assigned occurrence identifier. Existing signed and absolute accumulations in the client's path tracker consequently remain attached to the same mapped edges. Hence the processing of general child-internal occurrence changes carries no factor D_{i+1} or k . $\square$

Remark 8.8. The marking rules of Lemmas 8.2 and 8.7 serve different purposes. A terminal promotion or segment subdivision may merely reencode an unchanged path of the input graph, so its logical correction need not be marked in Lemma 8.2. It nevertheless marks the endpoint pools of copy slots in Lemma 8.7. Conversely, a logical class split changes its leg data while selected central chords remain attached at their original tree nodes. For this reason the logical snapshot update and the public forest update have the separate bounds stated in Lemmas 8.3 and 8.7.

8.6 Total work, finite arithmetic, and reliability

Because the two terms of (8.14) are bounded separately, the small child edit count of Lemma 8.4 controls the recursive term of the next lemma.

Lemma 8.9 (Recursive work bounds). *Assume (7.1), (7.2), (7.3), and (5.8). For a prefix of N_{op} graph operations, counting queries and edits and including an operation that terminates partway through, the level- i cycle data structure has total work and explicit public-log size at most $n_i A_i + N_{\text{op}} U_i$, where*

$$A_i, U_i = M^{O(1/a) + o_a(1)}.$$

All coefficients in the $O(1/a)$ exponents are absolute.

Proof. The proof bounds the initialization and the local incremental work at one level, combines them with the child's work into two recurrences for A_i and U_i , solves these recurrences over the d levels, and finally extends the bounds to unsuccessful attempts.

Initialization. The child receives its arbitrary nonempty initial graph directly. The parent's own initialization costs $n_i M^{O(1/a) + o_a(1)}$. This includes tree preparation, initial assignments and compression, full forest copies, and initial pools of copy slots. Initially the number of active child slots is at most h_i times a logarithmic factor; each has D_{i+1} pools and path copies of at most $k + 1$ steps. Since $D_{i+1} \leq M^{4/a}$, this cost has the required absolute exponent. Epoch teardown and its public logs satisfy the same bound. Allocations and later deallocation are counted; the time to reclaim a discarded memory arena is at most proportional to the time of its earlier allocations.

Local incremental work. Lemma 8.3 bounds the local assignment work, Lemma 8.4 the local construction work for J , and Lemma 8.7 the public forest updates. In the first term of (8.14), $|Q|$ is at most a constant times the child edit and assignment counts of Lemma 8.4, so the cumulative size of the sets Q in an epoch prefix is at most b times a logarithmic factor per parent edit. Together with $D_{i+1} \leq M^{4/a}$ and $k = M^{1/a}$, all local incremental work is $M^{O(1/a) + o_a(1)}$ amortized per input operation. Queries use the maintained best keys and lift one answer, with logarithmic overhead per child piece. They need not alter the public forest.

The recurrences. Let $L = M^{C/a + o_a(1)}$, for an absolute constant C , dominate these local factors and the bottom-level costs. Such an absolute C exists: all nonlogarithmic local factors above are fixed powers of b , k , and D_{i+1} ; at the bottom the elementary negative-cycle computation is a fixed-degree polynomial in $n_d = M^{2/a}$. Factors involving r, a^2, P_*, p_* and degree bounds are logarithmic factors.

Initialization of the child contributes $n_{i+1} A_{i+1}$, with logarithmic overhead on its logs, and $n_{i+1}/n_i = M^{-1/(2a)}$. At most $1 + N_{\text{op}}/h_i$ parent epochs start. After initialization the child receives at most $b M^{o_a(1)}$ times the parent operation count, including the one recursive query used for a parent

query. Only a logarithmic factor multiplies child log-processing work, by Lemma 8.7. Consequently it suffices to choose dominating upper estimates satisfying

$$A_i = L + M^{-1/(2a)+o_a(1)} A_{i+1}, \quad (8.16)$$

$$U_i = L + kA_i + M^{1/a^2+o_a(1)} U_{i+1}, \quad (8.17)$$

with $A_d, U_d \leq L$. The term kA_i bounds the work of the rebuild every $h_i = n_i/k$ edits, including its child initialization, amortized over those edits.

Solving the recurrences. For fixed a , the multiplier in the first recurrence tends to zero, so $A_i \leq 2L$ eventually after enlarging L if necessary. Unrolling the second recurrence through at most $d \leq 2a$ levels gives

$$U_i \leq (d+1)(L + 2kL)M^{d/a^2+o_a(1)} = M^{(C+3)/a+o_a(1)}.$$

Thus the coefficient $C+3$ of $1/a$ in this exponent is absolute. Logarithmic overhead accumulated over the fixed number of levels remains $M^{o_a(1)}$.

Unsuccessful attempts. Each attempted cover or sparsifier reconstruction (Lemmas 7.2 and 6.1) has its own work bound, valid on every computation path, whether it succeeds or is rejected. The work of such a reconstruction is counted in the bound for the attempted operation, also when failure or a guard (Section 2) stops it before publication. All previously published states satisfy their verified bounds. The same recurrences therefore hold for these prefixes and unsuccessful computations. $\square$

The arithmetic of the construction is finite. All maintained data are finite integers, dyadics, explicit lists, or identifiers. Frozen tree prefix sums and segment sums use at most the number of constituent edges times the level price bound. Width buckets are integer powers of two. Correction scores compare ratios of integral gradient magnitudes to the positive integral proxies Z_*W_f by cross multiplication; bottom scores and inherited scores are likewise represented by bounded integral numerator–denominator pairs. The conceptual hierarchy-law probabilities, the real flows of the comparison argument, and barrier logarithms occur only in the analysis; the construction evaluates none of them. Exact geometric delays, dyadic thresholds, and the randomness of the routing sparsifier are generated by the finite procedures already specified. Fixed-parameter data have $\log^{O_a(1)} S$ bit length, and ordinary dictionary and forest record operations have multiword logarithmic-factor overhead. The integral-flow calls inside the routing sparsifier use the simulation of Lemma 2.3; its logical addresses are not assumed polynomially bounded in their numerical values.

Lemma 8.10 (Deterministic checks and failure probability). *Assume (7.1), (7.2), and (5.8). The cycle data structure constructed in Sections 7 and 8 can enforce all size, multiplicity, arithmetic, and sampling prerequisites by finite checks. For every sufficiently large fixed a and every fixed $\sigma \geq 0$, those deterministic prerequisites hold for all sufficiently large sizes in the permitted range (5.8), on every random path. Conditional on any adaptively specified history before an invocation, the detected sampling failure probability over an entire primitive instance is at most $2^{-\sigma} S^{-60}$.*

Proof. Every bound in this proof other than the probability bounds holds on every random path. No deterministic guard, in the sense of Section 2, is violated; the stage *No guard is reached* derives this, applying Lemma 8.9 up to a putative first guard violation. The probability bound is a union bound over at most S^{500} stochastic invocations.

Guards and checks. Share counters throughout the tree of recursive instances of the cycle data structure within one primitive instance. Allow S^{200} outer graph operations of the cycle data

structure, and guard total records, fresh identifiers, and stochastic invocations by S^{500} . The cap S^{500} exceeds the count $S^{200+O(1/a)+o_a(1)}$ derived below for sufficiently large fixed parameters and size. Fresh identifiers can be sequential. Count allocations and record creations as they occur; check child capacity, input-list sizes of the routing sparsifier, public multiplicity, price ranges, and quality inequalities before using the corresponding state. Deletion-first transactions maintain the capacity and degree bounds during replacement. The checks $P_* \leq b$ and $20Z_*A_* \leq b^2$, the inequalities in (8.11), the multiplicity checks, and the price bounds all hold eventually by their positive power gaps over logarithmic factors.

Bounds on every random path. Published covers satisfy their verified coverage and radius requirements, and published outputs of the routing sparsifier satisfy the verified support, path, hop, and resource bounds. Every rejected attempt separately has bounded work: sampled piece lists partition their input edges at a capped recursion depth; sample size and weight checks precede flow extraction; the router has a fixed number of jobs per label, shrinking label groups, no packet duplication, and capped walk trials and substitution depths. Long real embeddings are filtered before expansion. These properties are proved in Section 6. Covers likewise have capped delays, attempts, and finite graph computations.

No guard is reached. Lemma 8.9 at the outer level, where $n_0 = M$ by (7.2), applies to a prefix ending just before a putative first violation of these guards, with the partially attempted operation counted as one operation. With $S^{1/a} \leq M \leq S^2$ by (5.8), the initial construction and at most S^{200} outer operations together cost at most

$$M^{1+O(1/a)+o_a(1)} + S^{200} M^{O(1/a)+o_a(1)} = S^{200+O(1/a)+o_a(1)}$$

elementary records, invocations, and local operations. This is strictly below S^{500} for sufficiently large fixed parameters and size, including the attempted guard-reaching operation. The motion and refresh accounting of the step primitive (Section 5.7), which is independent of changes of the public forest, keeps its outer graph-operation count below S^{200} under its client bounds, including threshold-driven reprices. Thus the first supposed violation is impossible.

Suspension and detected failure. A failed deterministic prerequisite causes suspension (Section 2). This includes a shared guard that interrupts a stochastic invocation before it has completed its prescribed attempts. For an in-range call of the routing sparsifier, Lemma 6.1 gives conditional failure probability at most $2^{-\sigma} S^{-800}$. A cover draw passes jointly with probability greater than $1/2$ under its checked parameters. Using fresh draws for the prescribed $2(\sigma + 1000 \log_2 S + 10)$ attempts makes its exhaustion probability at most $2^{-2\sigma-20} S^{-2000} \leq 2^{-\sigma} S^{-800}$; the number of attempts is chosen to give this inequality. The input to either invocation may depend on all earlier outputs: the stated bounds are conditional on that history, and the new randomness is fresh.

Failure probability. There are at most S^{500} such invocations. Summing their conditional probabilities, or equivalently summing probabilities of the first exhausted invocation, bounds the probability of any sampling failure by

$$S^{500} 2^{-\sigma} S^{-800} \leq 2^{-\sigma} S^{-60}.$$

The exponent -60 is the one required in Lemma 5.2. Exhausting the full allowed retries is reported as detected failure. Until such failure, all returned states satisfy the deterministic representation, length, and recourse guarantees. $\square$

Completion of the proof of Lemma 5.2. The assertions of Lemma 5.2 are collected in three groups: quality, representation, and cost and reliability.

Quality. By Lemma 8.2 and (8.10), the snapshot hierarchy supplies the simple selected-support paths required by Lemma 7.8, with at most p_* hops. Lemma 8.4 constructs its one child with the

required capacity, integral prices, and degree. Lemma 8.5 preserves certified child scores in the public representation. These are the path assignments and the public forest lifting assumed in Corollary 7.10, which therefore gives at level i a certified score at least

$$\frac{1}{2}b^{-2(d-i)} \text{OPT}_i.$$

At the outer level this is at least $M^{-10/a} \text{OPT}$ for sufficiently large parameters, as required by the checked quality inequalities. If $\text{OPT} > 0$, the chosen pieces have positive raw length and achieve their certified score. If $\text{OPT} = 0$, an empty answer is allowed.

Representation. The pieces are unit-multiplicity actual edges or public forest paths, their signed sum is a circulation, and their number has the required bound by Lemma 8.5. Vertex multiplicity is at most $M^{10/a}/4$ by Lemma 8.6. Lemma 8.7 gives explicit forest edits and retained immutable lifetimes, with removals exposed before the old maps are invalidated.

Cost and reliability. Finally Lemmas 8.9 and 8.10, together with the finite arithmetic described before Lemma 8.10, give initialization time $M^{1+O(1/a)+o_a(1)}$, amortized operation and log cost $M^{O(1/a)+o_a(1)}$ on every path, finite arithmetic, and the stated conditional bound on failure in an entire instance. These are all the assertions of Lemma 5.2. $\square$

9 Explicit output and one uniform program

We first reduce exact cardinality to subcubic perfect matching, retaining all witness information. We then combine the fixed-parameter procedures into the single program of Theorem 1.1. Throughout this section $p = n + m$ refers to the size of the original input graph G .

9.1 Witness-preserving reductions

The algorithm first converts the adjacency lists into edge records. It reads adjacency-list boundaries and neighbor occurrences once, and emits an edge record, with its endpoints and a fresh identifier, only at the occurrence (u, v) with $u < v$. Input validity and simplicity imply that this produces each edge exactly once, regardless of list order. All vertex records are retained, including isolates. This takes $O(n + m)$ word operations, and the resulting read-only data can be shared among the parameter branches of Section 9.2. If $n = 1$, the algorithm returns the empty list.

Two reductions connect the required output to the subcubic procedure. First, a rearrangeable network turns a cardinality threshold into a perfect-matching instance. Second, a path gadget reduces the degree to three. Both reductions come with witness decoders. For a target cardinality k , adding $n - 2k$ new vertices adjacent to every original vertex would turn a size- k matching into a perfect matching, but could add quadratically many edges. The network replaces these complete connections by $O(n \log(n + 2))$ positions and arcs while allowing any $n - 2k$ original vertices to reach the required new vertices.

Lemma 9.1 (A linearithmic rearrangeable network). *For every power of two $s \geq 1$, there is a directed acyclic network with s distinct input positions and s distinct output positions, of size $O(s \log(s + 2))$, in which every permutation of inputs to outputs is routable by vertex-disjoint directed paths. A position may be both an input and an output when $s = 1$. The network can be constructed uniformly within its size bound.*

Proof. The construction follows Beneš's recursive construction of rearrangeable networks [1]. For $s = 1$, take a single position and its length-zero route. For $s > 1$, take s new input positions, s new output positions, and disjoint upper and lower networks on $s/2$ positions. Connect each input in

pair $2i - 1, 2i$ to input i of both copies. Connect output j of either copy to both positions in output pair $2j - 1, 2j$. The size recurrence is $Q(s) = 2Q(s/2) + O(s)$, and all arcs respect the resulting input-to-output order.

Given a permutation, form a bipartite multigraph between its input pairs and output pairs, with one edge per assigned input–output pair. Every vertex has degree two, so the components are even cycles, allowing a cycle of two parallel edges. Color each cycle alternately upper and lower. Every input pair and output pair has one edge of each color. The upper edges specify a permutation through the upper network, and the lower edges one through the lower network. By induction, these two permutations are routed by vertex-disjoint paths in the upper and lower networks, and attaching the corresponding outer arcs extends the routes to the whole network. The two copies are disjoint and each outer position occurs once, giving vertex-disjoint routes. The network itself is generated by the recursion above, from the lists of input and output positions and their indices within the pairs, which gives the uniform construction within the size bound. $\square$

Lemma 9.2 (From a cardinality threshold to perfect matching). *Given G and an integer $0 \leq k \leq \lfloor n/2 \rfloor$, one can construct in $O(m + n \log(n + 2))$ time a loopless graph H_k of size $O(m + n \log(n + 2))$ such that H_k has a perfect matching if and only if G has a matching of size k . A perfect matching of H_k yields an explicit size- k matching of G in linear time in the constructed size.*

Proof. Construction. Put $b = n - 2k$ and choose the least power of two $s \geq n$. Use the network of Lemma 9.1 and select b distinct output positions. Retain all original vertices and edges. Split every network position v into vertices $v_{\text{in}}, v_{\text{out}}$ joined by an *idle edge*; we call v a split position. An arc $u \rightarrow v$ becomes the *transition edge* $u_{\text{out}}v_{\text{in}}$. Join original vertex i to the input endpoint of network input i . Finally, add one sink vertex adjacent to the output endpoint of each selected output. Every constructed vertex is required to be covered by a perfect matching.

A perfect matching of H_k gives a matching of size k . In any perfect matching, the two endpoints of a split position are either joined by their idle edge or both matched externally. The numbers of externally matched input and output endpoints are therefore equal. Selected transition edges contribute once to each number and cancel. Original-to-network edges contribute one input endpoint, and edges to sinks contribute one output endpoint. All b sinks must be covered, so exactly b original vertices use network edges. The remaining $n - b = 2k$ original vertices are covered by original edges, which give the decoded matching.

A matching of size k gives a perfect matching of H_k . Conversely, let a size- k matching leave b original vertices uncovered. Map them bijectively to the selected outputs and extend to a permutation of all s positions. By Lemma 9.1, this permutation is routed by vertex-disjoint paths; we keep the b paths that start at the inputs of the uncovered vertices. On each retained path, select its original-to-input edge, all transition edges, and the edge to its sink. Each split position on a retained path is covered once at each endpoint. Use idle edges on all other positions and the original matching on its covered vertices. This is a perfect matching of H_k . For a length-zero route, the two outer edges cover the one split position. For $b = 0$, every split position uses its idle edge.

Size and decoding. The number of added positions and edges is $O(n \log(n + 2))$. Store each edge’s type and original identifier. Decoding selects the original edges from the returned perfect matching. $\square$

The reduction of Lemma 9.3 uses the path replacement for matching described by Dahlhaus and Karpinski [5]; we record the construction and the witness conversion needed here.

Lemma 9.3 (Degree reduction). *A loopless graph with v vertices and e edges either has an isolated vertex, certifying absence of a perfect matching, or can be replaced in $O(v + e)$ time by a graph of*

maximum degree three and size $O(v + e)$ that has a perfect matching if and only if the given graph has one. Witnesses can be converted in either direction within the same bound.

Proof. The algorithm first checks for isolated vertices. Otherwise, replace a vertex of degree $r \geq 1$ by a path on $2r - 1$ vertices, attaching its r old incidences to successive odd positions. Retain the original edge identifiers at both ends. There are r odd and $r - 1$ even positions. Every even position must be matched internally to an odd one. Exactly one odd position therefore remains for an external edge. Hence each gadget selects exactly one old incidence, and any perfect matching projects to one in the original graph.

Conversely, the odd attachment of a selected original matching edge leaves two even-length paths on its sides, and each of them has a perfect matching along the path. This also covers $r = 1$, whose gadget is a single vertex. Each odd position has at most two path neighbors and one external edge, and each even position has degree two. In terms of the numbers v of vertices and e of edges of the given graph, the number of gadget vertices is

$$\sum_u (2 \deg(u) - 1) = 4e - v.$$

A degree scan, the assignment of incidences to odd positions, and the path construction give the time bound and the explicit conversions. $\square$

The reduced instance and its size bound. For a threshold $k > 0$, the algorithm applies Lemma 9.2 to obtain H_k and then Lemma 9.3 to H_k . Unless absence of a perfect matching is already certified by an isolate, this gives a loopless subcubic instance; let N denote its number of vertices. Then

$$N = O(m + n \log(n + 2)), \quad N \geq n, \tag{9.1}$$

where the first bound holds because H_k has size $O(m + n \log(n + 2))$ and Lemma 9.3 increases the size by at most a constant factor. The latter inequality holds because every original vertex has at least one gadget vertex. On a simple input the construction can itself use simple graphs, since network arcs, attachments and path edges join their specified distinct positions. Parallel edges would in any case be admissible, since Theorem 4.8 applies to subcubic graphs with identified edge copies.

For the perfect-matching procedure, S is set to the first power of two at least $16 + n + m + N$. Thus

$$n \leq S = O(p \log(p + 2)). \tag{9.2}$$

The upper bound follows from (9.1). This choice also gives $N \leq S$, as Theorem 4.8 requires, and $S \geq n + 1$, which is used for (9.4) below. None of these consequences needs the summand 16. Every proposed perfect matching is checked by edge identifier and by covering each gadget vertex exactly once. Decoding then reverses the two reductions: Lemma 9.3 converts the perfect matching back to one of H_k , and Lemma 9.2 extracts the original size- k matching, in time linear in the size of these auxiliary instances.

9.2 Threshold search and its error probability

A *branch* is one complete run of the threshold search below. For $t \geq 1$, branch t answers its tests with Theorem 4.8 at the parameters (9.3); branch zero, described at the end of this subsection, uses Lemma 2.4. The quantities in play are the input size p ; the reduced size N and the size bound S of Section 9.1; and the branch index t , which determines a and σ .

Each branch runs its own integer binary search. Initially the feasible lower endpoint is zero, with the stored empty matching, and the exclusive upper endpoint is $\lfloor n/2 \rfloor + 1$. An accepted test stores its decoded witness and raises the lower endpoint; a negative test lowers the upper endpoint. There are $O(\log(n+2))$, and certainly at most $n+1$, tests. Exact-size feasibility is monotone because edges may be dropped from a matching. Consequently, if no feasible test receives a false negative, the final stored witness is maximum.

For each branch index $t \geq 1$, use

$$a = 1000 + t, \quad \sigma = t + 20 \tag{9.3}$$

in Theorem 4.8. These values satisfy its hypotheses $a \geq 1000$ and $0 \leq \sigma \leq a+1$, since $t+20 \leq 1001+t$. The offset 1000 meets the lower bound $a \geq 1000$; a grows with t so that the $O(1/a)$ term of the exponent can be made small in the proof of Theorem 1.1; and the offset 20 makes the error bounds of the branches sum to $2^{-20} < 1/3$ in (9.5). Recall from Section 2 that a failed guard suspends the entire branch with no output, whereas a detected failure (exhaustion of a capped randomized routine) ends only the current perfect-matching test, with an ordinary negative answer. All invocations use fresh independent bits.

By (4.25), on a feasible perfect-matching instance the probability of a false negative is at most $2^{-\sigma} S^{-5}$, conditional on its input and the earlier history, and not conditional on whether the branch is interrupted later. Each reached randomized invocation with valid prerequisites has its declared conditional bound. The union bound over these invocations is part of (4.25): it is the stage *Error probability* of the proof of Theorem 4.8, which counts the reached invocations with the caps stated after (3.8). Each instance of the cycle data structure already includes its capped internal invocations in its error estimate. There are no false positive perfect-matching answers. Hence a branch completes with an incorrect matching only if some feasible test returns a false negative. There are at most $n+1$ adaptively chosen tests, and $S \geq n+1$ at all nontrivial tests, so the union bound gives at most $(n+1) \cdot 2^{-\sigma} (n+1)^{-5} \leq 2^{-\sigma}$; with $\sigma = t+20$,

$$\Pr[\text{branch } t \text{ completes with an incorrect matching}] \leq 2^{-(t+20)}. \tag{9.4}$$

Inequality (9.4) holds on every input, including inputs on which branch t suspends.

Branch zero performs the same reduction and search, using the deterministic procedure of Lemma 2.4, which applies to the explicit loopless graphs of maximum degree three produced by Lemma 9.3. Every threshold answer is exact, so this branch always completes with a maximum matching. It uses a fixed polynomial number of instructions after finite-word simulation. Indeed every simple input satisfies $m \leq n(n-1)/2$, and (9.1) makes every fallback instance polynomial in $n+2$.

9.3 Running time of a single branch

Lemma 9.4 (Absence of suspension and running time of branch t). *Let $p = n + m$, and for $t \geq 1$ let branch t be the binary search of Section 9.2 whose threshold tests apply Theorem 4.8, with S as in (9.2) and with the parameters a, σ of (9.3), to the instances produced by Lemmas 9.2 and 9.3. For every sufficiently large fixed t , branch t avoids deterministic suspension on all sufficiently large inputs, on every computation path. Its total instruction count, including unsuccessful tests, threshold search, and explicit witness handling, is*

$$p^{1+O(1/a)+o_a(1)}.$$

The coefficient in $O(1/a)$ is absolute.

Proof. Absence of suspension comes from the estimates on guarded prefixes (Section 2) proved in Section 4; the time bound multiplies the time bound (4.26) of Theorem 4.8 by the number of tests and adds the reductions.

Absence of suspension. Each test applies Theorem 4.8 to a loopless subcubic graph with $N \leq S$ vertices, with S satisfying (9.2) (Section 9.1) and with parameters (9.3) that satisfy its hypotheses (Section 9.2). A branch suspends only when a guard fails. The guards of a test are those of the perfect-matching construction of Sections 3 and 4, including the initial checks (3.8), the S^{20} iteration caps, the S^{100} external-operation cap, and the nominal-size conditions such as (5.2), together with the guards of the cycle data structure: its S^{200} and S^{500} caps and the deterministic prerequisite checks of Section 8 (Lemma 8.10). At fixed sufficiently large a, σ , Lemma 4.7 shows that the guards of the construction are adequate for all sufficiently large S , on every computation path, independently of matchability and of which verified constructions or sampling failures occurred; this is the no-suspension clause of Theorem 4.8. Part (v) of Lemma 5.2 gives the same for the guards of the cycle data structure, on every computation path: every accepted sampled structure satisfies its checked deterministic invariants, and rejected attempts obey their own unconditional caps (stage *Bounds on every random path* of the proof of Lemma 8.10). It remains to see that S is large enough: since $S \geq 16 + n + m + N > p$ by the choice of S , this holds on all sufficiently large inputs, on every computation path.

Running time. By (4.26), each test takes $S^{1+O(1/a)+o_a(1)}$ instructions; there are $O(\log(n+2))$ tests, and the reductions take $O(m+n \log(n+2))$ time by Lemmas 9.2 and 9.3. Together these give the displayed exponent: logarithmic and fixed-parameter subpolynomial factors absorb the substitution $S = O(p \log(p+2))$ from (9.2).

Computation of the parameters. All parameters and adequacy checks are computed by finite integer loops: bit lengths, rounded powers of two, products, quotients, and comparisons of scaled dyadics or ratios. Parameters that are too large for the input may fail their checks. The program evaluates each check directly and stores no table of thresholds; the input sizes beyond which fixed-parameter logarithmic factors fit within the size, multiplicity or quality margins appear only in the analysis. Fixed-parameter bit lengths and the instruction counts of arithmetic and dictionary simulation are bounded by powers of logarithms, or are already counted in the static integral-flow algorithm of Lemmas 2.1 and 2.3. Each random bit is counted as one instruction. $\square$

9.4 An instruction-level schedule

Weighted dovetailing, familiar from universal search [23], lets one program run every branch t with a slowdown that depends only on t . This subsection defines the schedule and its finite-word implementation; the error bound of the combined program is (9.5) at the end of the subsection.

A *scheduled step* is one bounded word-RAM instruction of the machine of Section 1, with constant control overhead. A whole graph routine, an arbitrary-precision operation, or an allocation request therefore consists of many scheduled steps. Here and below, *machine-level* refers to this machine, with its w -bit words, addresses and instructions, as opposed to the logical values and addresses of the simulated branch computations.

Run rounds $j = 0, 1, 2, \dots$. In round j , give branches $t = 0, \dots, j$, in that order, at most 2^{j-t} scheduled steps each, resuming a branch at its interruption point. A suspended branch stays inactive. Stop when the first branch completes with its answer. The first branch to complete constructs its decoded output privately, within the steps counted for it; copying the list of at most $n/2$ original labeled edges adds work linear in the input size.

The branches are built from one fixed program of graph, list, tree and integer subroutines, the fallback of Lemma 2.4, and one integral-flow implementation of Lemmas 2.1 and 2.3, the same for

all branches. Store each branch as a coroutine with a program counter and stack. Use digit arrays, whose operations are counted instruction by instruction, for longer logical values; the digits may even be single bits. Long virtual indices and addresses are keys in dictionaries of touched cells with a default initial value. Comparisons, initialization, recursion, and dictionary access are themselves interruptible implementations.

Allocate and initialize records one word at a time. A paused computation retains its state without copying it; its machine-level storage is proportional to the allocations it has executed plus the shared input. Allocation areas may retain cells that are no longer used. In particular, computing a power of a parameter that depends on the branch index, however large the result, is counted instruction by instruction and can pause inside its arithmetic or allocation loop, so no memory interval of that size is reserved in a single step. For each branch t of Lemma 9.4, these implementations add only overheads bounded by powers of logarithms, which the bound of that lemma already absorbs. The simulation of the fallback still has a polynomial bound.

Lemma 9.5 (Halting, space, and word length of the schedule). *The schedule of Section 9.4, which in each round $j = 0, 1, 2, \dots$ gives each branch $t \in \{0, \dots, j\}$ at most 2^{j-t} scheduled steps, is realizable by one finite program with a word-length constant c_0 , the constant in $w = \lceil c_0 \log_2(n+2) \rceil$. On every computation path, whatever its outcome, it halts after polynomially many instructions and uses polynomially many addressed words, with constants independent of the input.*

Proof. Branch zero finishes within $O(\log(n+2))$ rounds, which bounds the total work and storage by a fixed polynomial; c_0 is then chosen so that this polynomial fits in w -bit words.

Work bound. We first bound the work while ignoring machine-level address overflow. Fix the fallback's finite digit-array and touched-cell simulation. The instruction count of the simulated fallback, its bit lengths, and its dictionary operations give a fixed polynomial bound $F(n)$ on its work and records. This estimate concerns branch zero alone, and it may use unit-bit data and machine-level pointer operations.

Unless the schedule has already stopped, the fallback receives 2^j steps in round j , hence $2^{J+1} - 1$ steps through round J . It therefore finishes by round J , the least integer with $2^{J+1} - 1 \geq F(n)$, and $J = O(\log(n+2))$ with an implied constant independent of the input. The total number of steps given to all branches through round J is

$$\sum_{j=0}^J \sum_{t=0}^j 2^{j-t} < 2^{J+2}.$$

Starting and switching branches, status checks, and loop control add only polylogarithmic overhead; input handling, setup and output are polynomial as well. Since every allocation is counted among the scheduled steps, the machine-level records touched before halting are bounded by a fixed polynomial in $n+2$, even if every allocation area retains its unused cells.

Choice of the word constant. Each branch uses its own logical indexing, and machine-level addresses serve only as pointers; they may be assigned by sequential allocation. Hence the interleaving leaves the logical computation of each simulated branch the same as in a separate run. Choose c_0 once, large enough that this polynomial bound covers machine-level addresses, the step counters of the schedule, program control, and input indices in the prescribed w -bit words. No overflow then occurs on any path. Long logical integers remain digit arrays. Smaller logical words can be masked or simulated; long shifts, divisions and other operations use the corresponding loops, whose instructions are counted. Thus the choice of c_0 is determined by this polynomial bound. $\square$

For the probability calculation we couple the schedule to independent infinite fair-bit streams for its branches, revealing bits only when requested. This is equivalent to generating fresh random

bits during execution, each counted as one instruction. Consider each branch's own complete computation on its stream, including a possible suspension. The scheduled output is the output of one such complete computation, namely that of the first branch to finish, so interleaving and the identity of that branch impose no conditioning on the error estimates (9.4). Branch zero is exact, so the scheduled output is incorrect only if some branch $t \geq 1$ completes with an incorrect matching, and (9.4) gives

$$\Pr[\text{the scheduled output is incorrect}] \leq \sum_{t \geq 1} 2^{-(t+20)} < \frac{1}{3}. \quad (9.5)$$

Here $1/3$ is the failure probability allowed by Theorem 1.1.

9.5 Proof of Theorem 1.1

Proof of Theorem 1.1. The algorithm is the schedule of Section 9.4. The proof first obtains halting and correctness from Lemma 9.5 and (9.5). It then compares the schedule with a fixed branch t : the schedule runs branch t with a slowdown depending only on t , branch t runs in $p^{1+O(1/a)+o_a(1)}$ instructions by Lemma 9.4, and $a = 1000 + t$ can be taken as large as needed; so for every $\xi > 0$ the single program runs within $p^{1+\xi}$ instructions on all large inputs. The function η is then defined from the exponents that the schedule attains, and the sparse and dense bounds follow from its monotonicity.

Halting and correctness. Halting and the computation model follow from Lemma 9.5; explicit output and success follow from the reductions of Section 9.1 (Lemmas 9.2 and 9.3), the threshold search of Section 9.2, and (9.5).

Comparison with a fixed branch. The program, including the parameters (9.3), the schedule and c_0 , is fixed before the analysis and does not depend on ξ . The analysis chooses, in this order, a number $\xi > 0$, then a branch index t depending on ξ , and then an input-size threshold p_0 depending on ξ and t ; it shows that for every $\xi > 0$ there is p_0 such that every computation path on every input with $p \geq p_0$ halts within $p^{1+\xi}$ instructions.

Fix a sufficiently large branch index t . If it needs T_t steps, round $j = t + \lceil \log_2 \max(1, T_t) \rceil$ alone gives it $2^{j-t} \geq \max(1, T_t) \geq T_t$ steps, unless another answer has already terminated the schedule. By the double-sum bound in the stage *Work bound* of the proof of Lemma 9.5, with j in place of J , the total number of scheduled steps through that round is less than $2^{j+2} = 2^{t+2} \cdot 2^{\lceil \log_2 \max(1, T_t) \rceil} < 2^{t+3}(T_t + 1)$, a constant times $2^t(T_t + 1)$, in addition to input and logarithmic control overhead. Thus the single schedule runs within a factor depending only on t of every branch t .

Now let $\xi > 0$. Choose t large enough that the $O(1/a)$ term in the exponent of Lemma 9.4, whose coefficient is absolute, is smaller than a suitable fraction of ξ ; this is possible because $a = 1000 + t$ by (9.3). For all sufficiently large p , its $o_a(1)$ term, the constant factor and the logarithmic overhead are absorbed by the remaining part of the exponent ξ . Consequently the *same program* halts within $p^{1+\xi}$ instructions on every computation path for all sufficiently large inputs; the index t is chosen only in the analysis.

The exponent function η . Let $T(G, \omega)$ be the total instruction count of the schedule on input G and random path ω . For $x \geq 2$ define

$$e(x) = \sup_{G, \omega: n(G)+m(G) \geq x} \max \left\{ 0, \log_{n(G)+m(G)}(\max\{1, T(G, \omega)\}) - 1 \right\}.$$

The polynomial bound of Lemma 9.5, which comes from the fallback, bounds this supremum, and the comparison with a fixed branch above, valid on every computation path, implies $e(x) \rightarrow 0$. Put $\eta(q) = e(\max\{2, q\})$ for integers $q \geq 1$. This function is nonnegative and nonincreasing, and it tends

to zero. If $p \geq 2$, the term for (G, ω) in the supremum defining $e(p)$ gives

$$T(G, \omega) \leq p^{1+\eta(p)}.$$

The only input with $p = 1$ is the one-vertex empty graph, which is handled directly. Choose $C \geq 1$ large enough to cover its instruction count. The all-input bound in Theorem 1.1 follows.

Sparse and dense inputs. For the sparse specialization, let $\eta^* = \sup_{q \geq 1} \eta(q) < \infty$. If $m \leq \kappa n$, then $n \leq p \leq (1 + \kappa)n$, and monotonicity gives $\eta(p) \leq \eta(n) \leq \eta^*$, so

$$T(G, \omega) \leq Cp^{1+\eta(p)} \leq C(1 + \kappa)^{1+\eta^*} n^{1+\eta(n)}.$$

Thus the same algorithm and the same function η apply for every κ , with $C_\kappa = C(1 + \kappa)^{1+\eta^*}$. When $m = \Theta(n^2)$, one instead has $p = \Theta(n^2)$, giving the stated $n^{2+o(1)}$ dense specialization.

Completeness of the count. Every operation, random bit, intermediate construction and output copy is counted in T , so the bound covers the whole computation. The input conversion of Section 9.1 applies to every simple graph in any list order. This proves all assertions of Theorem 1.1. $\square$

10 Prescribed-degree factors

Recall that an f -factor of $G = ([n], E)$ is a set $F \subseteq E$ with $\deg_F(v) = f(v)$ for every vertex v . We reduce its existence and construction to one maximum-matching computation, adapting Tutte's factor-to-perfect-matching reduction [35] with the construction in the stage *Construction* of the proof of Lemma 9.2, in which every network position is split into an input and an output vertex. The underlying rearrangeable network is the Beneš construction proved in Lemma 9.1 [1]. In Tutte's reduction, each vertex has one port vertex for each incident edge, and $\deg_G(v) - f(v)$ of its ports must be matched locally. Complete connections to that many auxiliary vertices would use $\deg_G(v)(\deg_G(v) - f(v))$ edges. A separate rearrangeable network at each vertex v permits any such subset of ports with $O(\deg_G(v) \log(\deg_G(v) + 2))$ vertices and edges. The input consists of the graph and n degree records, so its length is within a constant factor of $p = n + m$.

Corollary 10.1 (Almost-linear f -factor search). *There exist a single uniform randomized word-RAM algorithm, constants $c_f \geq 2$ and $C_f \geq 1$, and a nonincreasing function $\eta_f : \mathbb{N}_{\geq 1} \rightarrow \mathbb{R}_{\geq 0}$ with $\eta_f(q) \rightarrow 0$ as $q \rightarrow \infty$ such that the following holds. The input is a simple graph $G = ([n], E)$ with $n \geq 1$, $m = |E|$, and $p = n + m$, in the adjacency-list representation of Section 1, together with one-word integers $f(v)$ satisfying $0 \leq f(v) \leq \deg_G(v)$ for all v . With the instruction set and costs of Section 1 and word length $w_f = \lceil c_f \log_2(n + 2) \rceil$, the algorithm halts on every computation path in at most*

$$C_f p^{1+\eta_f(p)}$$

instructions. Every output is either an explicit edge list of a valid f -factor or NO. If an f -factor exists, it returns one with probability at least $2/3$; otherwise it always returns NO.

Lemma 10.2 (Linearithmic factor reduction). *For the input of Corollary 10.1, one can construct a possibly empty simple graph H_f with $|V(H_f)| + |E(H_f)| = O(m \log(n + 2))$ such that H_f has a perfect matching if and only if G has an f -factor. The construction takes $O(n + m \log(n + 2))$ word instructions and records. A perfect matching of H_f yields an explicit f -factor within the same bound.*

Proof. The graph H_f has one cross edge for each host edge, joining two port vertices, and at each vertex v a split copy of the network of Lemma 9.1 that connects the ports at v to $\deg_G(v) - f(v)$ sink vertices; correctness follows from the two correctness stages of the proof of Lemma 9.2, applied at every vertex.

Construction. For every host edge $e = \{u, v\}$, create distinct port vertices $x_{u,e}$ and $x_{v,e}$ and join them by a *cross edge* labeled by e . For a vertex v write $d_v = \deg_G(v)$ and $b_v = d_v - f(v)$. If $d_v = 0$, then $f(v) = 0$ and no local construction is needed. Otherwise, let s_v be the least power of two at least d_v and take a separate copy of the recursive network in Lemma 9.1. Assign distinct input positions to the d_v incidences at v , and select b_v distinct output positions.

Split every network position z into vertices z_{in} and z_{out} joined by an idle edge. For every directed arc $z \rightarrow z'$ of the network, add the transition edge $z_{\text{out}}z'_{\text{in}}$. Join $x_{v,e}$ to the input endpoint of its assigned input position. At each selected output position, add a new sink vertex adjacent only to that position's output endpoint. All these local constructions use disjoint new vertices. The recursive network has distinct arcs, and the edge types have distinct endpoint pairs, so H_f is simple. This also covers $s_v = 1$, when the single position is both an input and an output and there is no transition edge.

A perfect matching of H_f gives an f -factor. The network at v , with the ports at v attached to its inputs and b_v sinks attached to its selected outputs, is the construction of the stage *Construction* of the proof of Lemma 9.2 with the ports in place of the original vertices and b_v in place of b . The counting argument of the next stage of that proof, which derives a matching from a perfect matching (input and output endpoints matched by non-idle edges are equally many, and all sinks have degree one), shows that exactly b_v port attachments at v are used. Every port has only its attachment and its cross edge. Hence exactly $d_v - b_v = f(v)$ ports at v use cross edges, and the labels of all selected cross edges form a set $F \subseteq E$ with $\deg_F(v) = f(v)$ for every v .

An f -factor gives a perfect matching of H_f . Conversely, suppose that F is an f -factor. At v , exactly b_v ports belong to edges outside F . The stage of the proof of Lemma 9.2 that derives a perfect matching from a matching, applied to the network at v with these b_v ports in place of the uncovered original vertices and b_v in place of b , selects edges that cover these ports, all split endpoints, and all sinks at v exactly once, including when $b_v = 0$ or $s_v = 1$. The cross edges of the edges $e \in F$ cover each remaining port exactly once, since every port has exactly one cross edge. The local constructions are disjoint, so these edges form a perfect matching of H_f .

Size and decoding. For $d_v > 0$ we have $s_v \leq 2d_v$, so the local network has $O(d_v \log(d_v + 2))$ positions and arcs. Ports, cross edges, attachments, and sinks add $O(m)$ more records. Since $d_v \leq n - 1$ and $\sum_v d_v = 2m$, their total is $O(m \log(n + 2))$. The uniform network construction and a scan of the adjacency lists take $O(n + m \log(n + 2))$ instructions. Scanning the occurrence with the smaller endpoint label assigns an identifier to each host edge. Two passes over the constructed edge list count degrees and fill the adjacency-list representation of H_f within the same bound. Every identifier uses $O(\log(n + 2))$ bits because the constructed size is polynomial in n . Store each cross edge's host identifier; scanning a returned perfect matching and selecting these labels gives the stated decoder bound. $\square$

Proof of Corollary 10.1. If $m = 0$, validity forces every $f(v) = 0$, so return the empty edge list. Otherwise H_f is nonempty; construct it and apply the single algorithm of Theorem 1.1. Check the returned edge identifiers and one-time coverage of every vertex, returning NO unless the list is a perfect matching. Section 9's output implementation copies at most $|V(H_f)|/2$ edges on every path, so the check takes $O(|V(H_f)| + |E(H_f)|)$ instructions, including initialization. If it passes, decode the cross-edge labels by Lemma 10.2, checking their host identifiers and prescribed degrees.

When the matching call returns a maximum matching, the test accepts exactly when H_f has a perfect matching and then supplies the factor by the reduction. This maximum-matching event has probability at least $2/3$. The checks ensure that every accepted edge list is a valid factor, regardless of the matching call's outcome. Thus a failed call can cause a false negative, whereas an input with no factor always yields NO.

Put $P = |V(H_f)| + |E(H_f)| = O(p \log(p + 2))$. Since $|V(H_f)|$ is polynomial in n , the matching algorithm's virtual word length $\lceil c_0 \log_2(|V(H_f)| + 2) \rceil$ is at most $\lceil c_f \log_2(n + 2) \rceil$ for a constant c_f . Choose c_f large enough also for the construction's indices. Simulate each virtual word instruction by a constant number of machine-level instructions, masking to the virtual width for modular arithmetic and Boolean operations. Quotients, remainders, comparisons, shifts, and memory addresses then have the same semantics as in the matching call; each fair random bit is one instruction, as in Section 1. The masks and input conversion take at most the reduction's work bound. Thus one finite program, with the word constant c_f , handles all input sizes and degree prescriptions.

For fixed $\xi > 0$, Theorem 1.1 gives $O_\xi(P^{1+\xi/2})$ instructions on every computation path, absorbing small P into the constant. Since $P = O(p \log(p + 2))$, the logarithmic factor is absorbed as well: $P^{1+\xi/2} = O_\xi(p^{1+\xi})$. Together with all reduction, simulation, checking, and output work, this gives $O_\xi(p^{1+\xi})$ for large p , for this same program for every $\xi > 0$. Boundedness of η also gives a polynomial bound on every computation path. The construction of η by a supremum in the proof of Theorem 1.1 (Section 9.5), applied to this program, now gives a nonnegative nonincreasing $\eta_f(q) \rightarrow 0$; a constant C_f covers the small input sizes, including $p = 1$. $\square$

References

- [1] V. E. Beneš. Optimal rearrangeable multistage connecting networks. *Bell System Technical Journal*, 43(4):1641–1656, 1964. doi: 10.1002/j.1538-7305.1964.tb04103.x.
- [2] Claude Berge. Two theorems in graph theory. *Proceedings of the National Academy of Sciences of the United States of America*, 43(9):842–844, 1957. doi: 10.1073/pnas.43.9.842.
- [3] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 612–623. IEEE, 2022. doi: 10.1109/FOCS54457.2022.00064. arXiv:2203.00671v2; full version.
- [4] Li Chen, Rasmus Kyng, Yang P. Liu, Simon Meierhans, and Maximilian Probst Gutenberg. Almost-linear time algorithms for incremental graphs: Cycle detection, SCCs, s - t shortest path, and minimum-cost flow, 2023. arXiv:2311.18295v1.
- [5] Elias Dahlhaus and Marek Karpinski. Perfect matching for regular graphs is AC^0 -hard for the general matching problem. *Journal of Computer and System Sciences*, 44(1):94–102, 1992. doi: 10.1016/0022-0000(92)90005-4.
- [6] Ran Duan, Seth Pettie, and Hsin-Hao Su. Scaling algorithms for weighted matching in general graphs. *ACM Transactions on Algorithms*, 14(1):8:1–8:35, 2018. doi: 10.1145/3155301. arXiv:1411.1919.
- [7] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. doi: 10.4153/CJM-1965-045-4.
- [8] Jack Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of Research of the National Bureau of Standards, Section B: Mathematics and Mathematical Physics*, 69B(1–2):125–130, 1965. doi: 10.6028/jres.069B.013.
- [9] L. R. Ford, Jr. and D. R. Fulkerson. Network flow and systems of representatives. *Canadian Journal of Mathematics*, 10:78–84, 1958. doi: 10.4153/CJM-1958-009-1.
- [10] Harold N. Gabow. The weighted matching approach to maximum cardinality matching. *Fundamenta Informaticae*, 154(1–4):109–130, 2017. doi: 10.3233/FI-2017-1555. arXiv:1703.03998.
- [11] Harold N. Gabow. Maximum cardinality f -matching in time $O(n^{2/3}m)$. *ACM Transactions on Algorithms*, 21(1):9:1–9:28, 2025. doi: 10.1145/3696668. arXiv:2311.14236v1.
- [12] Harold N. Gabow and Robert E. Tarjan. Faster scaling algorithms for general graph-matching problems. *Journal of the ACM*, 38(4):815–853, 1991. doi: 10.1145/115234.115366.
- [13] David Gale. A theorem on flows in networks. *Pacific Journal of Mathematics*, 7(2):1073–1082, 1957. doi: 10.2140/pjm.1957.7.1073.
- [14] Mohsen Ghaffari, Fabian Kuhn, and Hsin-Hao Su. Distributed MST and routing in almost mixing time. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 131–140. ACM, 2017. doi: 10.1145/3087801.3087827.
- [15] Andrew V. Goldberg and Alexander V. Karzanov. Maximum skew-symmetric flows and matchings. *Mathematical Programming*, 100(3):537–568, 2004. doi: 10.1007/s10107-004-0505-z. arXiv:math/0304290v2.

- [16] Philip Hall. On representatives of subsets. *Journal of the London Mathematical Society*, s1-10(1):26–30, 1935. doi: 10.1112/jlms/s1-10.37.26.
- [17] Nicholas J. A. Harvey. Algebraic algorithms for matching and matroid problems. *SIAM Journal on Computing*, 39(2):679–702, 2009. doi: 10.1137/070684008.
- [18] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. doi: 10.1137/0202019.
- [19] A. V. Karzanov. An exact estimate of an algorithm for finding a maximum flow, applied to the problem “on representatives”. In *Voprosy Kibernetiki. Trudy Seminara po Kombinatornoi Matematike (Moscow, 1971)*, pages 66–70. Sovetskoe Radio, Moscow, 1973. In Russian.
- [20] Rohit Khandekar, Satish Rao, and Umesh V. Vazirani. Graph partitioning using single commodity flows. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, pages 385–390. ACM, 2006. doi: 10.1145/1132516.1132574.
- [21] Rasmus Kyng, Simon Meierhans, and Maximilian Probst Gutenberg. A dynamic shortest paths toolbox: Low-congestion vertex sparsifiers and their applications, 2023. arXiv:2311.06402v1.
- [22] Rasmus Kyng, Simon Meierhans, Maximilian Probst Gutenberg, and Aurelio Sulser. A simpler and faster min-cost flow solver via min-ratio cycles from distance oracles, 2026. arXiv:2609.23852v1.
- [23] Leonid A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1973. URL <https://www.mathnet.ru/eng/ppi914>. Russian original in Problemy Peredachi Informatsii 9(3), 115–116.
- [24] László Lovász. On determinants, matchings, and random algorithms. In Lothar Budach, editor, *Fundamentals of Computation Theory, FCT ’79*, pages 565–574, Berlin, 1979. Akademie-Verlag.
- [25] Aleksander Mądry. Navigating central path with electrical flows: From flows to matchings, and back, 2013. arXiv:1307.2205v3; full version of the FOCS 2013 paper.
- [26] Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{|V|} |E|)$ algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science*, pages 17–27. IEEE, 1980. doi: 10.1109/SFCS.1980.12.
- [27] Gary L. Miller, Richard Peng, and Shen Chen Xu. Parallel graph decompositions using random shifts. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 196–203. ACM, 2013. doi: 10.1145/2486159.2486180. arXiv:1307.3692.
- [28] Marcin Mucha and Piotr Sankowski. Maximum matchings via Gaussian elimination. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 248–255. IEEE, 2004. doi: 10.1109/FOCS.2004.40.
- [29] Václav Rozhoň, Christoph Grunau, Bernhard Haeupler, Goran Zuzic, and Jason Li. Undirected $(1 + \varepsilon)$ -shortest paths via minor-aggregates: Near-optimal deterministic parallel & distributed algorithms, 2022. arXiv:2204.05874v2.
- [30] Alistair Sinclair and Mark Jerrum. Approximate counting, uniform generation and rapidly mixing Markov chains. *Information and Computation*, 82(1):93–133, 1989. doi: 10.1016/0890-5401(89)90067-9.

- [31] Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983. doi: 10.1016/0022-0000(83)90006-5.
- [32] Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32(3):652–686, 1985. doi: 10.1145/3828.3835.
- [33] Joel A. Tropp. User-friendly tail bounds for sums of random matrices. *Foundations of Computational Mathematics*, 12(4):389–434, 2012. doi: 10.1007/s10208-011-9099-z. arXiv:1004.4389v7.
- [34] W. T. Tutte. The factorization of linear graphs. *Journal of the London Mathematical Society*, s1-22(2):107–111, 1947. doi: 10.1112/jlms/s1-22.2.107.
- [35] W. T. Tutte. A short proof of the factor theorem for finite graphs. *Canadian Journal of Mathematics*, 6:347–352, 1954. doi: 10.4153/CJM-1954-033-3.
- [36] Jan van den Brand, Yin-Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 919–930. IEEE, 2020. doi: 10.1109/FOCS46700.2020.00090. arXiv:2009.01802.
- [37] Jan van den Brand, Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, Sushant Sachdeva, and Aaron Sidford. A deterministic almost-linear time algorithm for minimum-cost flow, 2023. arXiv:2309.16629v1; full version of the FOCS 2023 paper.
- [38] Jan van den Brand, Yang P. Liu, and Aaron Sidford. Dynamic maxflow via dynamic interior point methods. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 1215–1228. ACM, 2023. doi: 10.1145/3564246.3585135. arXiv:2212.06315v1; full version.
- [39] Vijay V. Vazirani. A theory of alternating paths and blossoms from the perspective of minimum length. *Mathematics of Operations Research*, 49(3):2009–2047, 2024. doi: 10.1287/moor.2020.0388.