Change-Driven Parallel Top Trading Cycles via Persistent Choice Forests

Team Member

Vaibhav Shakya

Jayshree Periwal International School

Jaipur, India

Supervising Teacher

Satnam Singh Chhabra

Head of Department, Computer Science

Jayshree Periwal International School

Jaipur, India

Registration Number

Comp-213

2026 S.-T. Yau High School Science Award
仅用于2026丘成桐中学科学奖论文公示

Research Report
Commitments on Academic Honesty and Integrity

2026 S.T. Yau High School Science Award (Asia)

We hereby declare that we

1. are fully committed to the principle of honesty, integrity and fair play throughout the competition.
2. perform the research work ourselves and thus truly understand the content of the work.
3. observe the common standard of academic integrity adopted by most journals and degree theses.
4. have declared all the assistance and contribution we have received from any personnel, agency, institution, etc. for the research work.
5. have truthfully disclosed all AI tool usage in the research report and obtained prior approval from supervising teachers. All supporting documentation has been submitted as required. We hereby affirm that no improper use of AI has occurred at any stage of the research process.
6. undertake to avoid getting in touch with assessment panel members in a way that may lead to direct or indirect conflict of interest.
7. undertake to avoid any interaction with assessment panel members that would undermine the neutrality of the panel member and fairness of the assessment process.
8. observe the safety regulations of the venue(s) where we conduct the experiment(s), if applicable.
9. observe all rules and regulations of the competition.
10. agree that the decision of YHSA(Asia) is final in all matters related to the competition.

We understand and agree that failure to honor the above commitments may lead to disqualification from the competition and/or removal of award, if applicable; that any unethical deeds, if found, will be disclosed to the school principal of the team member(s) and relevant parties if deemed necessary; and that the decision of YHSA(Asia) is final and no appeal will be accepted.

(Signatures of full team below)

X Vbhakya
Name of team member: Vaibhav Shakya

X Satnam Singh
Name of supervising teacher: Satnam Singh Chhabra

Noted and endorsed by

Manisha Razdan
(signature)

Name of school principal: Manisha Razdan

2026 S.T. Yau High School Science Award
仅用于2026丘成桐中学科学奖论文公示

Change-Driven Parallel Top Trading Cycles via Persistent Choice Forests

Vaibhav Shakya
Jayshree Periwal International School

Abstract

Top Trading Cycles (TTC) computes the unique core allocation of a Shapley–Scarf housing market. Parallel implementations recompute the top-choice graph in every round; on markets with bounded preference lists this can cost $\Theta(n^2)$ operations although the input has size $\Theta(n)$, and the best prior parallel bound is $O(n^2 \log n)$ expected work. We determine when the round structure can be avoided and how it can be executed at low cost when it cannot. We prove that deciding the TTC outcome is P-complete under deterministic-logspace reductions once agents have three relevant choices, and is in NC for two. We give a change-driven algorithm that retains unchanged choices in a persistent forest and processes only affected agents, using $O(m \log n)$ expected work, $O(R \log^2 n)$ span with high probability, and $O(m)$ space, where m is the total relevant-list size and R the number of rounds. The implementation is evaluated with exhaustive differential validators and with counter measurements that agree with the analysis.

Keywords: Top Trading Cycles; Shapley–Scarf housing markets; parallel algorithms; P-completeness; batch-dynamic trees; work-span analysis.

2026 S.-T. Yau High School Science Award
仅用于2026丘成桐中学科学奖论文公示

Contents

1 Introduction **3**
1.1 Related work . . . 4

2 Preliminaries **4**

3 The complexity threshold **5**
3.1 Width two . . . 6
3.2 Asynchronous execution . . . 6
3.3 Source problem and wire encoding . . . 7
3.4 The AND/OR cell . . . 8
3.5 The COPY cell . . . 9
3.6 Isolation of unevaluated cells . . . 10
3.7 Circuit simulation . . . 11
3.8 Selector splitting . . . 12
3.9 The P-completeness theorem . . . 13

4 Persistent choice forests **14**
4.1 Data structure invariant . . . 14
4.2 The quotient graph and cycle correspondence . . . 14
4.3 The charging argument . . . 15
4.4 The algorithm . . . 16
4.5 Correctness . . . 16

5 Parallel implementation and complexity **17**
5.1 Preference maintenance . . . 17
5.2 Dynamic forest operations . . . 18
5.3 Quotient cycle detection . . . 19
5.4 The work-span theorem . . . 19

6 Limits and parameter dependence **20**
6.1 The logarithmic factor . . . 20
6.2 The round parameter . . . 21

7 Experimental evaluation **23**
7.1 Setup and reproducibility . . . 23
7.2 Correctness validation . . . 23
7.3 Structural counters . . . 24
7.4 Structural comparison of sequential solvers . . . 24
7.5 Parallel backend . . . 24
7.6 Limitations . . . 25

8 Discussion: structural work versus parallel performance **25**

9 Conclusion **26**

2026 S.-T. Yau High School Science Award
仅用于2026丘成桐中学科学奖论文公示

All code, benchmarks, and validators are publicly available at https://github.com/vaibhavshakya11/TTC.

1 Introduction

In the Shapley–Scarf housing market, each of n agents owns one indivisible house and has a strict preference order over houses [23, 22]. Gale's Top Trading Cycles (TTC) mechanism computes an allocation as follows: every remaining agent points to the owner of its most preferred remaining house, every cycle of the resulting graph trades and is removed, and the process repeats until no agent remains. TTC is individually rational, strategy-proof, and Pareto efficient, and it returns the unique core allocation under strict preferences [21, 19]. Within a single round the pointer graph is functional, its cycles are vertex-disjoint, and cycle detection parallelizes by standard techniques. The difficulty lies in the sequence of rounds: the pointers of round $t+1$ depend on the deletions of round t.

A natural parallel implementation reconstructs the pointer graph at the start of every round. If S_t denotes the set of agents remaining in round t and R the number of rounds, this costs $\sum_{t=1}^{R} |S_t|$ pointer computations in total. There are markets with constant-size preference lists in which $R = \Theta(n)$ and this sum is $\Theta(n^2)$, although the input itself has size $\Theta(n)$. The prior shared-memory parallel algorithm of Garg [11] performs $O(n^2 \log n)$ expected work in its dense representation, and the distributed algorithm of Zheng and Garg [27] uses $O(n)$ rounds and $O(n^2)$ messages. To our knowledge, no previous exact algorithm avoids reconstructing the choices of agents that were not affected by the previous round.

Two questions arise. First, can the dependence on rounds be eliminated altogether, that is, does the problem admit a low-span parallel algorithm? Second, if not, can each round be executed at a cost proportional to the number of agents actually affected? This paper answers the first question negatively, under a standard assumption, at the smallest nontrivial list length, and the second question affirmatively.

For the first question, we study the decision problem TTC_d-Outcome: does a designated agent receive a designated house, when every agent has at most d *relevant* choices (the houses it prefers to its own, plus its own)? We prove that TTC_d-Outcome is P-complete under deterministic-logspace many-one reductions for every $d \geq 3$, and that it is decidable in deterministic NC for $d \leq 2$. Consequently, unless P = NC, no deterministic algorithm with polynomial work and polylogarithmic span decides the outcome once agents have three relevant choices. The proof encodes monotone circuit evaluation into market dynamics using two gadget types and a list-shortening transformation.

For the second question, we introduce the *persistent choice forest*. An agent's top choice changes only when the owner of its current choice departs, so all other choice edges are retained between rounds. The retained edges form a forest; only its roots recompute choices, and the recomputed edges define a functional *quotient* graph on the roots whose cycles correspond exactly to the TTC cycles of the round. A charging argument bounds the total number of root activations over the entire execution by m, the total size of the relevant lists. Combined with batch-dynamic tree data structures, this yields an exact parallel TTC algorithm with $O(m \log n)$ expected work, $O(R \log^2 n)$ span with high probability, and $O(m)$ space.

Our contributions are:

1. a complexity threshold for TTC outcome computation: P-completeness for $d \geq 3$ and membership in deterministic NC for $d \leq 2$ (Theorem 3.13 and Theorem 3.1);

2026 S. T. Yau High School Science Award
应用于2026丘成桐中学科学奖论文公开

2. the persistent-choice-forest formulation, with a correspondence between quotient cycles and TTC cycles and a bound of m on total root activations (Lemmas 4.1–4.4, Theorem 4.5);
3. a parallel algorithm with $O(m \log n)$ expected work and $O(R \log^2 n)$ span w.h.p., built from explicit implementation lemmas for preference maintenance, root queries, forest updates, and disjoint-path reporting (Theorem 5.4);
4. an implementation and experimental study, including validator suites, counter measurements that agree with the analysis, and an explicit account of the instance classes on which the method does and does not improve wall-clock performance (Sections 7 and 8).

1.1 Related work

TTC computation. Abraham, Cechlárová, Manlove, and Mehlhorn give a linear-time algorithm for Pareto-optimal house allocation whose endowment phase computes the unique core [1]; this fixes $O(m)$ as the sequential reference. Zheng and Garg prove that computing the core allocation is CC-hard by reduction from lexicographically first maximal matching (LFMM), show that verifying a core allocation is in NC, and give a distributed algorithm using $O(n)$ rounds and $O(n^2)$ messages [27]. Garg gives a shared-memory lattice-linear algorithm with expected time $O(n \log^2 n)$ and expected work $O(n^2 \log n)$ in a dense representation [11]. Chiang, Damgård, Orlandi, Pancholi, and Simkin compute TTC securely using oblivious functional-graph cycle detection [7]. We are not aware of prior work that retains unchanged choice edges across rounds or whose work bound is stated in terms of the sparse input size m.

Parallel primitives. List ranking and tree contraction provide work-efficient functional-graph primitives [8, 17]. Batch-parallel Euler tour trees [26], batch-dynamic rake-compress trees via change propagation [2], their deterministic work-efficient variants [4], and an arbitrary-degree implementation supporting general batch queries [15] supply the dynamic-forest machinery used in Section 5. Sleator and Tarjan's link-cut trees [24] provide a sequential realization of the same interface.

Complexity. Mayr and Subramanian define the class CC and prove stable matching CC-complete [16, 25]; Cook, Filmus, and Lê characterize CC, prove bounded-degree LFMM CC-complete, and give evidence that CC and NC are incomparable [9]. Degree-threshold P-completeness results are known for the k-core problem, P-complete for $k \geq 3$ and in NC for $k \leq 2$ [5], for Miyano's lexicographically first subgraph problems [18], and for bi-core decomposition [14]. Garg and Psomas prove that Round-Robin allocation is CC-hard when every valuation involves at most three items [12]. Our reduction source is the Alternating Monotone Fan-in-2, Fan-out-2 Circuit Value Problem of Greenlaw, Hoover, and Ruzzo [13]. Cell-probe lower bounds relevant to Section 6 are due to Fredman and Saks [10], Pătraşcu and Demaine [20], and Alstrup, Secher, and Spork [3]. Che and Tercieux analyze the number of TTC rounds in large random two-sided matching markets with full preference lists [6].

2 Preliminaries

A *market* consists of agents $V = \{1, \ldots, n\}$ and houses $H = \{h_1, \ldots, h_n\}$, where agent v initially owns h_v and has a strict total order over H. We identify v with h_v when no confusion arises.

2026 S.-T. Yau High School Science Award
仅用于2026丘成桐中学科学奖公示

Definition 2.1 (Relevant list)**.** The relevant list L_v of agent v consists of the houses that v ranks at least as high as h_v, in preference order; its last entry is h_v. Let $m = \sum_v |L_v|$ and $d = \max_v |L_v|$.

Entries ranked below h_v never matter: as long as v is present, h_v is present, and v prefers h_v to every such entry. An algorithm given full rankings can compute all relevant lists in time linear in the input and discard the remainder, so we assume the input is the collection of relevant lists, of total size $m \le nd$. When comparing our bounds with results stated for dense representations, such as the $O(n^2 \log n)$ bound of [11], part of the difference is attributable to representation and part to the algorithm; we make this explicit where relevant.

For a set $S \subseteq V$ of remaining agents, let $\text{choice}_S(v)$ denote the owner of the first house in L_v owned by a member of S. The *top-choice graph* on S has an edge $v \to \text{choice}_S(v)$ for every $v \in S$; it is functional and therefore contains at least one cycle. A *simultaneous TTC round* assigns to every vertex on a cycle the house it points to and removes all cycle vertices together with their houses. Let R denote the number of simultaneous rounds until the market is empty. An agent v with $\text{choice}_S(v) = v$ forms a *self-cycle* and receives its own house.

Definition 2.2 (Outcome problem)**.** TTC_d-Outcome is the following decision problem: given a market in which every relevant list has length at most d, a designated agent q, and a designated house h, does q receive h under TTC?

Computation model. We use the shared-memory work-span model with concurrent reads and writes (Common CRCW), the model in which the batch-dynamic tree bounds we import are stated. Work is the total number of operations and span the length of the longest dependency chain; by Brent's scheduling bound, a computation with work W and span D runs in $O(W/P + D)$ time on P processors. "With high probability" (w.h.p.) means with probability at least $1 - n^{-c}$ for a constant c that can be made arbitrarily large. NC denotes the class of problems decidable by deterministic algorithms with polynomial work and polylogarithmic span (equivalently, uniform polynomial-size polylogarithmic-depth circuits). P-completeness statements in this paper are under deterministic-logspace many-one reductions and rule out deterministic NC algorithms unless P = NC; excluding randomized polylogarithmic-span algorithms would require the stronger assumption $\text{P} \not\subseteq \text{RNC}$, which we do not assume. Our algorithm in Sections 4 and 5 is randomized.

We use the following primitives, each with linear expected work and $O(\log n)$ span w.h.p.: parallel filter, prefix sums, integer sorting on $O(\log n)$-bit keys, and cycle decomposition of a functional graph via randomized list contraction [8, 17]. Dynamic forests are maintained with batch-dynamic rake-compress trees [2, 4, 15]; the exact interface and cost lemmas appear in Section 5.

3 The complexity threshold

This section proves the two halves of the threshold. Section 3.1 shows that for $d \le 2$ the outcome is determined by a static computation and lies in NC. Sections 3.2–3.9 prove P-completeness for $d \ge 3$.

2026 S. T. Yau High School Science Award
仅用于2026丘成桐中学科学奖公示

3.1 Width two

Theorem 3.1. TTC_2-Outcome *is in NC.*

For $d \leq 2$, each relevant list is either (h_v) or (u, h_v) for a single house $u \neq h_v$. Define the static graph G on V with an edge $v \to u$ whenever $L_v = (u, h_v)$, and a self-loop at v whenever $L_v = (h_v)$. G is functional.

Lemma 3.2. *Under simultaneous TTC on a market with $d \leq 2$: (i) every agent on a nontrivial cycle of G receives the house it points to in G, and all such agents depart in round 1; (ii) every other agent receives its own house; (iii) no nontrivial cycle exists in any round after round 1.*

Proof. In round 1 every agent points to the first entry of its list, so the round-1 top-choice graph equals G. All cycles of G, including nontrivial ones, trade and depart, which proves (i).

For (iii), observe that at any later time, the current choice edge of a remaining agent v is either its original edge of G (if the target house is still present) or a self-loop (if the target was deleted, since the list has no third entry). Hence every nontrivial cycle among remaining agents would be a nontrivial cycle of G all of whose vertices remain, and all vertices of nontrivial G-cycles departed in round 1.

For (ii), let v be an agent not on a cycle of G with a proper edge $v \to u$. By (iii), v can depart only in a self-cycle, which assigns h_v; this happens at the latest once u's house is deleted, and u's house is eventually deleted because every agent departs. □

Proof of Theorem 3.1. By Lemma 3.2, the allocation is: agents on nontrivial cycles of G receive their G-successor's house, and all other agents keep their endowment. It therefore suffices to decide, for each agent, whether it lies on a nontrivial cycle of the functional graph G. Compute the reachability relation of G by repeated Boolean squaring of its adjacency matrix: $O(\log n)$ squarings of polynomial work each, in polylogarithmic depth. An agent v lies on a nontrivial directed cycle if and only if there is a vertex $u \neq v$ with v and u mutually reachable. This places the allocation, and hence the decision problem, in deterministic NC. □

The same static characterization is implemented as a standalone solver and tested against the general solvers in Section 7. For $d = 3$ the characterization fails: a deletion can promote an agent's second proper choice, so the graph evolves with the execution. The remainder of this section shows that this evolution supports circuit evaluation.

3.2 Asynchronous execution

Simultaneous TTC removes all current cycles in one round. The reduction is analyzed cycle by cycle, which the following lemma justifies. Call an execution *asynchronous* if in each step it removes one currently existing cycle (assigning each of its vertices the house it points to).

Lemma 3.3 (Persistence)**.** *Let K be a cycle of the current top-choice graph, and let one step of an asynchronous execution remove a cycle K' vertex-disjoint from K. Then K is a cycle of the resulting top-choice graph, with the same edges.*

Proof. Each vertex of K points to a house owned by a vertex of K. Removing K' deletes neither a vertex of K nor a house owned by K, and deletions never add houses, so no vertex of K gains a more preferred available house. Every vertex of K therefore keeps its edge. □

2026 S.-T. Yau High School Science Award
仅用于2026丘成桐中学科学奖论文公开

Lemma 3.4 (Commutation)**.** *If K_1, K_2 are vertex-disjoint cycles of the current top-choice graph, then removing K_1 and then K_2 yields the same market state and the same assignments as removing K_2 and then K_1.*

Proof. By Lemma 3.3, K_2 is still a cycle with the same edges after K_1 is removed, and vice versa; the assignments made for each cycle depend only on its own edges. The resulting state removes $K_1 \cup K_2$ and their houses in either order. □

Lemma 3.5 (Equivalence with simultaneous TTC)**.** *Every asynchronous execution terminates with the allocation produced by simultaneous TTC.*

Proof. We induct on n. Consider any asynchronous execution E. Every cycle C of the initial top-choice graph persists (by Lemma 3.3, applied repeatedly) until E removes it, and E must remove it eventually because the market empties and C's vertices can only depart on C while C exists. Using Lemma 3.4 repeatedly, reorder E so that all initial cycles are removed first; this changes neither assignments nor the resulting state. After the initial cycles are removed, the state equals the state after one simultaneous round, and the remaining steps of E form an asynchronous execution of the smaller market, to which the induction hypothesis applies. □

Lemma 3.5 also permits *deferring* a currently existing cycle: an asynchronous execution may remove other, disjoint cycles first, and the deferred cycle's edges and eventual assignments are unaffected.

3.3 Source problem and wire encoding

Source problem. An instance of the Alternating Monotone Fan-in-2, Fan-out-2 Circuit Value Problem (AM2CVP) [13] is a monotone Boolean circuit of AND and OR gates of fan-in two in which gate types alternate along every input–output path, circuit inputs feed OR gates, the distinguished output gate is an OR gate, and the fan-out of every input and of every internal gate is exactly two, with the distinguished output gate excepted; the question is whether the distinguished output evaluates to 1. AM2CVP is P-complete under NC^1-computable, hence deterministic-logspace, many-one reductions [13]. Our construction uses only monotonicity, fan-in two, and fan-out at most two, so the additional restrictions of the source problem cause no difficulty. Greenlaw, Hoover, and Ruzzo note that their P-complete circuit-value restrictions remain P-complete when the circuit is supplied in topological order [13]; we take topologically ordered AM2CVP as the source problem.

We reduce from the complement of AM2CVP. Because P is closed under complement, $\overline{\text{AM2CVP}}$ is P-complete under deterministic-logspace many-one reductions. Indeed, let $L \in \mathrm{P}$. Then $\overline{L} \in \mathrm{P}$. Since AM2CVP is P-complete, there is a deterministic-logspace reduction f such that

$$x \in \overline{L} \iff f(x) \in \text{AM2CVP}.$$

Therefore

$$x \in L \iff f(x) \notin \text{AM2CVP} \iff f(x) \in \overline{\text{AM2CVP}},$$

so f reduces L to $\overline{\text{AM2CVP}}$. The construction below reduces $\overline{\text{AM2CVP}}$ directly to TTC_3-Outcome.

2026 S. T. Yau High School Science Award
仅用于2026丘成桐高中科学奖论文公示

Wire encoding. Each circuit wire is represented by an *owner* agent W together with a private *trigger*: a designated agent inside the cell that produces the wire's value. The encoding invariant is:

- value 1 is represented by the owner W having departed;
- value 0 is represented by W remaining and pointing to a designated *consumer target*, an agent inside the cell that consumes the wire.

Before its producing cell has evaluated, W points to its trigger; we call such a wire *unresolved*. The producing cell resolves the wire either by deleting W (value 1) or by deleting the trigger while W remains, upon which W's list advances to the consumer target (value 0). A wire in one of these two states is *normalized*. Cells never write values elsewhere; the presence or absence of owners is the only state.

3.4 The AND/OR cell

Let P and Q be the owners of two normalized input wires. The cell introduces five fresh agents O, A, X, Y, Z and uses two *future targets* F_O, F_A, agents guaranteed to remain present throughout the cell's evaluation (their identity is fixed in Section 3.7). The relevant lists are

$$O:\ Z \succ F_O \succ O, \qquad A:\ Y \succ F_A \succ A, \qquad X:\ P \succ O \succ X,$$
$$Y:\ Q \succ P \succ A \succ Y, \qquad Z:\ Y \succ X \succ Q \succ Z.$$

The consumer target of a zero-valued P is Y, and the consumer target of a zero-valued Q is X. Figure 1 shows the cell and its four input cases.

Lemma 3.6 (AND/OR evaluation)**.** *Suppose both input wires are normalized, all five cell agents are present, and F_O, F_A are present. Then there is an asynchronous continuation that removes exactly the following cycles, in the stated order, after which O is absent if and only if $P \vee Q = 1$ and A is absent if and only if $P \wedge Q = 1$, and every surviving output owner is normalized:*

- *$P = 0, Q = 0$: the cycle (X, P, Y, Q), then the self-cycle (Z); O and A survive.*
- *$P = 1, Q = 0$: the cycle (O, Z, Y, Q, X); A survives.*
- *$P = 0, Q = 1$: the cycle (P, Y), then the cycle (O, Z, X); A survives.*
- *$P = 1, Q = 1$: the cycle (A, Y), then the cycle (O, Z, X); neither output survives.*

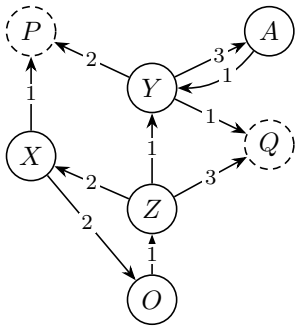
Proof. We trace each case from the stated lists; in each case the displayed cycles are cycles of the current top-choice graph, so Lemma 3.5 permits executing them in the stated order while deferring all other current cycles.

Case $P = 0, Q = 0$. Both owners are present, with $P \to Y$ and $Q \to X$. The cell edges are $X \to P$, $Y \to Q$, $Z \to Y$, $O \to Z$, $A \to Y$. The cycle $X \to P \to Y \to Q \to X$ exists; remove it. Now Y, X, Q are absent, so Z's list advances past Y, X, Q to h_Z, and the self-cycle (Z) is removed. O's first entry Z is now absent, so O points to F_O; similarly A points to F_A. Both outputs survive in normalized zero state, encoding $0 \vee 0 = 0$ and $0 \wedge 0 = 0$.

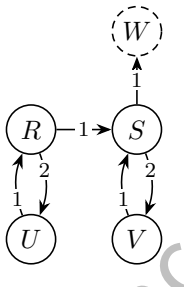
Case $P = 1, Q = 0$. P is absent and $Q \to X$. Then X's list advances past P to O, Y's list advances past Q's position only if Q were absent; here Q is present, so $Y \to Q$. The edges $O \to Z$, $Z \to Y$, $Y \to Q$, $Q \to X$, $X \to O$ form a cycle; remove it. A's first entry Y is absent, so $A \to F_A$ survives, encoding $1 \wedge 0 = 0$; O departed, encoding $1 \vee 0 = 1$.

2026 S. T. Yau High School Science Award
双月王2026丘成桐中学科学奖公示

P	Q	cycles in one valid order	O	A
0	0	(X, P, Y, Q); (Z)	0	0
1	0	(O, Z, Y, Q, X)	1	0
0	1	(P, Y); (O, Z, X)	1	0
1	1	(A, Y); (O, Z, X)	1	1



AND/OR cell



COPY cell

Figure 1: Top: the four input rows of the AND/OR cell; output value one means the output owner is deleted. Bottom: the two cells, with fresh owners solid, input wire owners dashed, and preference ranks on edges. Second-rank edges of O, A, U, and V to their future targets and all endowment entries are omitted.

Case $P = 0, Q = 1$. Q is absent and $P \to Y$. Y's list advances past Q to P, giving the cycle $Y \to P \to Y$; remove (P, Y). Now X's first entry P is absent, so $X \to O$; Z's first entry Y is absent, so $Z \to X$; with $O \to Z$ this forms the cycle (O, Z, X); remove it. $A \to F_A$ survives $(0 \wedge 1 = 0)$; O departed $(0 \vee 1 = 1)$.

Case $P = 1, Q = 1$. Both inputs are absent. Y advances past Q and P to A, and $A \to Y$, giving the cycle (A, Y); remove it. Then $Z \to X$ (past Y), $X \to O$ (past P), $O \to Z$: remove (O, Z, X). Both outputs departed $(1 \vee 1 = 1,\ 1 \wedge 1 = 1)$.

In every case, a surviving output owner has lost its first entry (Z for O, Y for A) and points to its future target, so it is normalized with the future target playing the role of consumer target. □

The cell computes both $O = P \vee Q$ and $A = P \wedge Q$; the construction selects the output it needs and discards the other (Section 3.7). Only Y and Z have four relevant entries; Section 3.8 reduces them to three. The cell templates were found by a bounded exhaustive search over candidate preference profiles, which also ruled out several smaller candidates within the searched bounds (Section 7); the correctness argument above is independent of the search.

3.5 The COPY cell

Fan-out two requires duplicating a wire. For a normalized input wire with owner W and future targets F_U, F_V, the COPY cell introduces four fresh agents U, V, R, S with lists

$$U:\ R \succ F_U \succ U, \qquad V:\ S \succ F_V \succ V, \qquad R:\ S \succ U \succ R, \qquad S:\ W \succ V \succ S,$$

and the consumer target of a zero-valued W is set to R.

Lemma 3.7 (COPY evaluation)**.** *Suppose the input wire is normalized and U, V, R, S, F_U, F_V are present. There is an asynchronous continuation after which U and V are absent if W was absent, and present and normalized if W was present; that is, both output wires carry the value*

2026 S. T. Yau High School Science Award 仅用于2026丘成桐中学科学奖论文公示

of W.

Proof. Case $W = 0$. W is present with $W \to R$. The edges $R \to S$, $S \to W$ close the cycle (R, S, W); remove it. Then U's first entry R is absent, so $U \to F_U$; similarly $V \to F_V$. Both outputs survive normalized.

Case $W = 1$. W is absent. S advances past W to V, and $V \to S$: remove the cycle (V, S). Then R advances past S to U, and $U \to R$: remove (U, R). Both outputs departed. □

Every COPY list has length at most three.

3.6 Isolation of unevaluated cells

The lemmas above describe a cell whose inputs are already resolved. The next two lemmas show that a cell with an unresolved input can take no action at all: none of its agents lies on a cycle, so no owner can be deleted prematurely. Throughout, the *topological index* of a wire or cell agent is the index of the gate, COPY cell, or source that produces or contains it in the fixed topological order of the circuit.

Lemma 3.8 (Unresolved-output barrier)**.** *Let W be an unresolved output wire of a producer cell G, and let H be the cell that will consume W. A successor walk in the current top-choice graph beginning at W cannot enter H while W remains unresolved.*

Proof. We argue by induction on the topological index of G. Since W is unresolved, W points to its private trigger inside G, not to its consumer target in H. Inspect the AND/OR and COPY templates in any state in which the chosen output of G is still unresolved: a successor step from an agent of G either remains inside G or reaches the owner U of an input wire of G. If U represents value 0, its unique consumer target lies inside G, so the next step returns to G. If U represents value 1, U is absent and the walk cannot traverse it. If U is unresolved, U points to its trigger inside its own producer cell, which has strictly smaller topological index, and by the induction hypothesis the walk continued from U cannot enter U's consumer, which is G or a cell other than H; in either case the walk has left G permanently through an earlier cell. No successor edge encountered in this process enters a downstream consumer of an unresolved output, and in particular none enters H. □

Lemma 3.9 (Isolation)**.** *Consider any reachable state of an asynchronous execution of the construction of Section 3.7. If an AND/OR or COPY cell has at least one unresolved input wire, then no agent of that cell lies on a cycle of the current top-choice graph.*

Proof. Let H be a cell with an unresolved input wire and let W be the owner of that wire. We first check that from every agent of H, the successor walk reaches the owner of an unresolved input of H without first returning to its starting agent.

For an AND/OR cell, suppose the input owned by P is unresolved; then P is present and $X \to P$. If Q is unresolved, $Y \to Q$; if Q is resolved to 0, then $Y \to Q \to X \to P$; if Q is resolved to 1, then $Y \to P$. In all cases the walks from X and Y reach an unresolved input owner, and $Z \to Y$, $O \to Z$, $A \to Y$ extend this to all five agents. The case in which only Q is unresolved is symmetric: $Y \to Q$ directly, and from X either $X \to P \to Y \to Q$ (if $P = 0$)

2026 S.T. Yau High School Science Award
仅用于2026日成利用学术论文公示

or $X \to O \to Z \to Y \to Q$ (if $P = 1$). For a COPY cell with unresolved W: $S \to W$, $R \to S$, $U \to R$, $V \to S$.

Having reached the unresolved input owner W, the walk continues into W's producer cell, and by Lemma 3.8 it cannot re-enter H while W remains unresolved. Hence the walk never returns to its starting agent, and no agent of H lies on a directed cycle. □

3.7 Circuit simulation

We now assemble the market $M(C)$ for a topologically ordered source circuit C with g gates.

Terminal. Create an agent C_* with list (h_{C_*}), and defer its self-cycle until the end (Lemma 3.5).

Sources. For every occurrence of an input bit at a gate input, create a fresh source owner. A source with value 0 has list (F, h_K), where F is its consumer target; it is normalized from the start. A source with value 1 has list (h_K) and is removed by its self-cycle at the beginning of the execution. Distinct occurrences of the same input bit get distinct owners, so fan-out at the inputs is unrestricted.

Gates. Replace each gate by an AND/OR cell, taking as output owner A for an AND gate and O for an OR gate. The unused output owner's future target is C_*. If the used output feeds one gate input, its future target is that input's consumer target (Y for a first input, X for a second input) in the consuming cell. If it feeds two gate inputs, its future target is the agent R of a fresh COPY cell, whose outputs U, V are wired to the two consuming inputs in the same way. If the gate is the designated output gate, the used output's future target is C_*.

Theorem 3.10 (Simulation)**.** *Let W_{out} be the used output owner of the designated output gate of C. In the market $M(C)$, under simultaneous TTC, W_{out} receives its own house if and only if C evaluates to 0.*

Proof. By Lemma 3.5 it suffices to exhibit one asynchronous execution and read off the allocation, and by the same lemma we may defer cycles freely.

We claim by induction on k that there is an asynchronous prefix after which cells $1, \dots, k$ (in topological order, counting each COPY cell together with its producing gate) have been evaluated as in Lemmas 3.6 and 3.7, every wire out of these cells is normalized and carries its circuit value, and no agent of any later cell has departed.

For $k = 0$: remove the self-cycles of all value-1 sources; value-0 sources are normalized by construction; no other agent has departed. Cycles of later cells did not exist to be removed, by Lemma 3.9.

For the step, consider cell $k + 1$. By induction its inputs are resolved and carry the correct values, and all its agents are present (no premature deletions occurred: while any of its inputs was unresolved, Lemma 3.9 shows none of its agents lay on a cycle, and afterwards no cycle involving its agents was removed by the prefix, which only evaluated cells $1, \dots, k$, whose evaluation cycles by Lemmas 3.6 and 3.7 involve only their own agents and their input owners). Each required future target is present: it is C_*, the R-agent of the COPY cell evaluated with the current gate, or an agent of a later consumer cell; C_* is deferred throughout, and no agent of the COPY cell or of any later cell has departed. Apply Lemma 3.6, and Lemma 3.7 if the output is copied: the asynchronous execution removes only the evaluation cycles of the cells scheduled so far and defers all other current cycles by Lemma 3.5. The outputs are normalized with the correct values.

2026 S. T. Yau High School Science Award
仅用于2026丘成桐中学生数学奖公示

Agents of later cells are untouched, and Lemma 3.9 continues to protect them.

After all cells are evaluated: if C evaluates to 1, W_{out} departed inside its cell, on a cycle in which it points to another agent's house, so it does not receive $h_{W_{\text{out}}}$. If C evaluates to 0, W_{out} survives pointing to C_*; remove the deferred self-cycle (C_*) and the remaining self-cycles of unused zero-valued outputs in any order. W_{out}'s list then advances to $h_{W_{\text{out}}}$ and it departs in a self-cycle, receiving its own house. □

3.8 Selector splitting

The agents Y and Z of each AND/OR cell have four relevant entries. The following transformation shortens a four-entry list to two three-entry lists.

Lemma 3.11 (Selector splitting)**.** *Let M be a market containing an agent v with relevant list (a, b, c, h_v). Form M' by introducing a fresh agent s_v, replacing v's list by (a, h_{s_v}, h_v), giving s_v the list (b, c, h_{s_v}), and ranking h_{s_v} below h_u for every pre-existing agent $u \neq v$. Then for every agent u present in M, u receives h_u in M if and only if u receives h_u in M'. Moreover, there is a correspondence between asynchronous executions of M and of M' under which the pre-existing agents depart in the same relative order, with the execution of M' possibly containing additional steps that remove only the selector.*

Proof. No pre-existing agent other than v ever points to h_{s_v}, since each ranks it below its own house. We couple executions of M and M' so that at corresponding states the same pre-existing agents and houses are present. In such a pair of states, consider v's current edge.

Case 1: a present. In both markets $v \to a$. No pre-existing agent other than v can point to h_{s_v}, and v points to a, so s_v cannot lie on a nontrivial cycle. If b and c have both disappeared while a is present, s_v's list is exhausted and s_v may self-cycle; the coupled execution of M' removes this selector-only self-cycle as an extra step, which removes no pre-existing agent and leaves v pointing to a. Any current cycle through v uses the edge $v \to a$ and otherwise consists of pre-existing agents, so it is present in both markets; removing it removes the same pre-existing agents.

Case 2: a absent, b present. In M, $v \to b$. In M', $v \to s_v$ and $s_v \to b$. Any cycle through v in M corresponds to the cycle in M' obtained by replacing the edge $v \to b$ with the path $v \to s_v \to b$, and conversely by contraction; the pre-existing agents on the two cycles coincide. (The house v receives differs: h_b in M, h_{s_v} in M'; neither is h_v, so v's own-house status agrees.)

Case 3: a, b absent, c present. As Case 2 with c in place of b; note s_v's list advances past b in M'.

Case 4: a, b, c absent. In M, v forms a self-cycle and receives h_v. In M', s_v's list is exhausted, so s_v self-cycles; after its removal v's list advances past h_{s_v} and v self-cycles, receiving h_v. Both executions remove v with its own house, and no pre-existing agent is affected by the extra step.

Cycles not through v consist of pre-existing agents and the correspondence is the identity on them. The coupling removes the pre-existing agents in the same relative order, interleaved in M' with selector-only steps, and preserves, for every pre-existing u, whether the cycle removing u assigns h_u; Lemma 3.5 transfers the conclusion to simultaneous TTC on both markets. The correspondence does not preserve simultaneous round numbers, and we make no such claim. □

2026 S.-T. Yau High School Science Award
仅用于2026丘成桐中学科学奖论文公示

Corollary 3.12. *Applying Lemma 3.11 to the agents Y and Z of every AND/OR cell of $M(C)$, in any order, yields a market $M_3(C)$ with all relevant lists of length at most three in which W_{out} receives its own house if and only if C evaluates to 0.*

Proof. Each application introduces one fresh agent, ranked below the endowments of all agents present at that time, and preserves the own-house status of all previously present agents by Lemma 3.11; in particular later applications preserve the conclusion of earlier ones. W_{out} is an output owner O or A, never split, so its own-house status in $M_3(C)$ equals that in $M(C)$, and Theorem 3.10 applies. The split lists (a, h_{s_v}, h_v) and (b, c, h_{s_v}) have length three, and all other lists in $M(C)$ already have length at most three. □

3.9 The P-completeness theorem

Theorem 3.13. *For every $d \geq 3$, TTC_d-Outcome is P-complete under deterministic-logspace many-one reductions, even when the query asks whether a designated agent receives its own house. For $d \leq 2$, TTC_d-Outcome is in NC. In particular, unless P = NC, no deterministic algorithm with polynomial work and polylogarithmic span decides TTC_d-Outcome for $d \geq 3$.*

Proof. Membership in P holds because TTC runs in $O(n+m)$ sequential time [1]. The NC half is Theorem 3.1, and every instance with lists of length at most three is an instance with lists of length at most d for any $d > 3$, so hardness for $d = 3$ implies hardness for all $d \geq 3$.

For hardness at $d = 3$: $\overline{\text{AM2CVP}}$ is P-complete under deterministic-logspace many-one reductions (Section 3.3). Given a topologically ordered AM2CVP circuit C, construct the market $M_3(C)$ with the query "does W_{out} receive $h_{W_{\text{out}}}$". By Theorem 3.10 and Corollary 3.12,

$$C \in \overline{\text{AM2CVP}} \iff W_{\text{out}} \text{ receives } h_{W_{\text{out}}},$$

so the map is a many-one reduction from $\overline{\text{AM2CVP}}$ to TTC_3-Outcome.

For the size and computability of the map: a circuit with g gates yields at most $5g$ cell agents, at most $4g$ COPY agents, at most $2g$ source owners, at most $2g$ selectors, and one terminal, hence $O(g)$ agents, each with a constant-length relevant list. Agents are indexed by a gate or occurrence index of $O(\log g)$ bits and a constant-size role. A logspace transducer enumerates agents in a fixed order and, for each, re-scans the topologically ordered circuit description to determine its list: the gate type selects O or A as used output, the consumer occurrences of each output are found by scanning the gate inputs, COPY cells and selectors are emitted per gate from the fixed templates, and, if complete rankings are required by the input convention, the remaining houses are appended below the endowment in a fixed order. Hence TTC_3-Outcome is P-hard, and P-complete. □

Corollary 3.14. *For $d \geq 3$, TTC_d-Outcome $\notin$ CC unless P = CC.*

Proof. CC is defined as the class of languages logspace-reducible to the comparator circuit value problem [16, 9] and is therefore closed under deterministic-logspace many-one reductions. If a logspace-P-complete problem belonged to CC, then every language in P would reduce to a CC language and hence belong to CC, giving P $\subseteq$ CC; together with CC $\subseteq$ P [16], this yields P = CC. □

2026 S. T. Yau High School Science Award 应用于2026丘成桐中学科学奖公示

Theorem 3.13 is consistent with the known CC-hardness of the problem: Zheng and Garg prove CC-hardness of core computation and CC-membership of the endowment-free allocation variant [27]; P-hardness implies CC-hardness and no collapse follows in either direction.

Remark 3.15. Bounded-width hardness does not follow from the LFMM reduction of [27]: there, an agent representing a vertex u has an effective list of length $\deg(u) + 1$, so obtaining $d = 3$ by that route would require LFMM to remain hard on graphs of maximum degree two, where it is computable in NC.

We note one structural feature of the reduction used later: the unsplit market $M(C)$ evaluates the circuit level by level, so its number of TTC rounds grows with the circuit depth (Proposition 6.5), while Lemma 4.4 below bounds the total number of choice recomputations by $O(g)$. The instances produced by the reduction are therefore deep-and-narrow in the sense of Section 6.

4 Persistent choice forests

We now give the algorithmic result. Throughout this section, "round" means simultaneous TTC round, and the algorithm performs exactly one iteration per round.

4.1 Data structure invariant

The algorithm maintains a directed graph F on the remaining agents, with at most one outgoing edge per agent, and a set A of *active* agents. The intended state at the beginning of each round is described by the following invariant.

(I1) F is a forest of in-arborescences: each connected component has a unique agent with no outgoing edge, its *root*, and every edge points toward the root.

(I2) For every non-root v, the edge of v in F is $v \to \mathrm{choice}_S(v)$, where S is the current set of remaining agents.

(I3) A is exactly the set of roots of F, and consists of the agents whose choice edge has not been computed for the current round: initially all agents, and subsequently the agents whose previous choice's owner departed in the previous round.

Initially F has no edges and $A = V$, which satisfies (I1)–(I3). For an agent x, let $\rho(x)$ denote the root of the component of F containing x.

4.2 The quotient graph and cycle correspondence

At the start of a round, each active agent $a \in A$ computes $c(a) = \mathrm{choice}_S(a)$, the owner of the first entry of L_a still present. Define the *quotient graph* Q on the vertex set A with edges

$$a \;\to\; q(a) := \rho(c(a)) \qquad (a \in A).$$

Q is functional. For a quotient edge $a \to q(a)$, let P_a denote the path in F from $c(a)$ to the root $\rho(c(a))$; if $c(a)$ is itself a root, P_a is the single vertex $c(a)$. Figure 2 illustrates one round.

The current top-choice graph on S consists of the stored edges of F (by (I2)) together with the new edges $a \to c(a)$ for $a \in A$.

Lemma 4.1 (Expansion)**.** *Let $a_0 \to a_1 \to \cdots \to a_{r-1} \to a_0$ be a cycle of Q. Replacing each edge $a_i \to a_{i+1}$ by the edge $a_i \to c(a_i)$ followed by the path P_{a_i} yields a cycle of the current top-choice graph.*

2026 S.-T. Yau High School Science Award 仅用于2026丘成桐中学科学奖论文公开

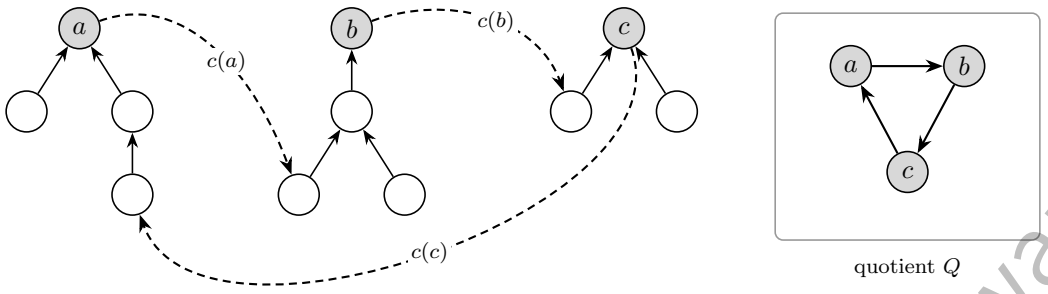


Figure 2: One round through the forest. Solid edges are stored choices and shaded nodes are roots; only a, b, c choose again (dashed), their targets lie in components rooted at b, c, a, and the quotient cycle $a \to b \to c \to a$ expands through the stored paths into exactly one current TTC cycle.

Proof. Every edge of the resulting closed walk is a current top-choice edge: $a_i \to c(a_i)$ by definition of c, and the edges of P_{a_i} by (I2). The walk visits, between a_i (exclusive) and a_{i+1} (inclusive), only vertices of the component of F rooted at a_{i+1}; since the a_i are distinct roots, these vertex sets are pairwise disjoint, so no vertex repeats before the walk closes at a_0. □

Lemma 4.2 (Contraction)**.** *Every cycle K of the current top-choice graph contains at least one edge $a \to c(a)$ with $a \in A$, and contracting every component of F to its root maps K to a cycle of Q whose expansion under Lemma 4.1 is K.*

Proof. F is acyclic by (I1), so K cannot consist of stored edges alone and contains an edge out of a root, which is an edge $a \to c(a)$ with $a \in A$ by (I3). Under contraction, stored edges become self-loops and are discarded, and each edge $a \to c(a)$ becomes $a \to \rho(c(a)) = q(a)$; the image is a closed walk in Q visiting distinct roots, hence a cycle of Q. Between two consecutive root edges, K follows stored edges from $c(a)$ upward within one component; since each vertex of F has exactly one outgoing stored edge, this path is exactly P_a, so expansion recovers K. □

Corollary 4.3 (Disjointness)**.** *The paths $\{P_a : a \to q(a)$ lies on a cycle of $Q\}$ are pairwise vertex-disjoint, and the cycles of the current top-choice graph are exactly the expansions of the cycles of Q.*

Proof. Q is functional, so its cycles are vertex-disjoint and each root has at most one outgoing cycle edge. The path P_a lies in the component rooted at $q(a)$, and distinct cycle edges of Q have distinct heads, so the paths lie in distinct components. The correspondence of cycles is Lemmas 4.1 and 4.2. □

4.3 The charging argument

Let A_t denote the active set in round t. An *activation* of agent v in round t is the event $v \in A_t$.

Lemma 4.4 (Activation bound)**.** $\sum_{t=1}^{R} |A_t| \le m$.

Proof. Agent v is activated in round 1. By (I3), each activation of v in a round $t \ge 2$ occurs because the owner of v's previously computed choice departed; the corresponding house was deleted, so the position of choice(v) in L_v advances by at least one between consecutive activations.

2026 S. T. Yau High School Science Award
仅用于2026丘成桐中学科学奖公示

Algorithm 1 Persistent-choice-forest TTC

1: $F \leftarrow$ empty forest on V; $A \leftarrow V$; $S \leftarrow V$
2: **while** $S \neq \emptyset$ **do**
3: **parallel for** $a \in A$: compute $c(a) = \text{choice}_S(a)$ and $q(a) = \rho(c(a))$
4: find all cycles of the functional graph $Q = (A, \{a \to q(a)\})$
5: **parallel for** each edge $a \to q(a)$ on a cycle of Q: report the path P_a from $c(a)$ to $q(a)$ in F
6: $C \leftarrow$ all vertices on reported paths and cycle roots ▷ cycle vertices
7: **parallel for** $v \in C$: assign v the house of its current top choice
8: delete the houses of C from all relevant lists; let $A' \leftarrow$ the agents of $S \setminus C$ whose list head changed
9: delete the vertices C (and their incident edges) from F
10: **parallel for** each non-cycle $a \in A$ with $c(a) \notin C$: insert the edge $a \to c(a)$ into F
11: $S \leftarrow S \setminus C$; $A \leftarrow A' \cup \{a \in A \setminus C : c(a) \in C\}$
12: **end while**

The position can advance at most $|L_v|-1$ times, so v is activated at most $|L_v|$ times, and summing over v gives $\sum_t |A_t| \le \sum_v |L_v| = m$. □

Three further global counts are used in Section 5. First, every occurrence of a house in a relevant list is deleted at most once, and every house is eventually deleted, so the total number of occurrence deletions is exactly m. Second, every agent is assigned on exactly one cycle, so the total number of cycle vertices reported over all rounds is n. Third, every edge inserted into F is inserted when its tail is active and removed at most once, so the numbers of insertions and removals are each at most $\sum_t |A_t| \le m$.

4.4 The algorithm

Algorithm 1 states the procedure; quotient edges are transient and are never inserted into F.

4.5 Correctness

Theorem 4.5. *Algorithm 1 computes the TTC allocation, and iteration t of the while-loop performs exactly the t-th simultaneous TTC round.*

Proof. We show that (I1)–(I3) hold at the start of every iteration and that each iteration removes exactly the cycles of the current top-choice graph with the correct assignments.

The invariant holds initially. Assume it holds at the start of an iteration with remaining set S. By Corollary 4.3, the set C computed in lines 4–6 is exactly the set of vertices on cycles of the current top-choice graph, and line 7 assigns each the house it points to; this is precisely one simultaneous round.

It remains to restore the invariant. Consider each remaining agent $v \in S \setminus C$ at the end of the iteration.

- *v was a non-root whose stored edge survives* (its head is not in C): the edge still points to choice(v), unchanged because v's head house was not deleted (otherwise the edge's head would be in C); v remains a non-root satisfying (I2).
- *v was a non-root whose stored edge's head is in C*: line 9 removes the edge; v's list head was deleted, so $v \in A'$ by line 8, and v enters the next active set as a root, consistent with (I3).

2026 Siyau High School Science Award 仅用于2026汇成青年科学奖论文公示

- *v was active with $c(v) \notin C$ and $v \notin C$*: line 10 inserts $v \to c(v)$, which is v's current choice edge; v becomes a non-root satisfying (I2).
- *v was active with $c(v) \in C$ but $v \notin C$*: no edge is inserted; v's computed choice was deleted, and line 11 keeps v active, consistent with (I3).

Acyclicity (I1) is preserved: contracting each old component to its root maps F after line 10 into the graph Q minus its cycle edges, which is acyclic, and deleting vertices preserves acyclicity. Each component has a unique root because every inserted edge removes exactly one root and every removed stored edge whose head lies in C creates a new component root at its surviving tail, matching the case analysis. Finally, (I3) holds by lines 8 and 11: the next active set is exactly the set of remaining agents whose current list head differs from their stored or computed choice.

Every functional graph has a cycle, so each iteration removes at least one agent and the loop terminates with all agents assigned. □

5 Parallel implementation and complexity

This section supplies cost bounds for each step of Algorithm 1 and derives the main theorem. Fix round t and write $a_t = |A_t|$ for the number of active agents, b_t for the number of preference-list occurrences deleted, c_t for the number of cycle vertices reported, h_t for the number of surviving agents whose list head changes, and u_t for the number of forest edge insertions and removals. Every surviving agent whose head changes is active in the next round, so $h_t \le a_{t+1}$. By Lemma 4.4 and the counts following it,

$$\sum_t a_t \le m, \qquad \sum_t b_t = m, \qquad \sum_t c_t = n, \qquad \sum_t h_t \le m, \qquad \sum_t u_t \le 2m. \tag{1}$$

5.1 Preference maintenance

Lemma 5.1. *After $O(m)$-work preprocessing, the relevant lists support the following batch operation: given the set of houses deleted in a round, remove all their occurrences and report every remaining agent whose list head changed, together with its new head. A batch that deletes b occurrences and changes h heads costs $O((b+h)\log n)$ work and $O(\log n)$ span, and the structure uses $O(m)$ space.*

Proof. Store each list L_v as the leaves, in preference order, of a static binary segment tree in which every internal node records whether its subtree contains a live occurrence, and store with each list a current-head pointer to its leftmost live leaf. Preprocessing builds all trees, the head pointers, and an inverted index mapping each house to *all* of its occurrences across all lists; the total size of the inverted lists is m, although a single house may have $\Theta(n)$ occurrences. This uses $O(m)$ total work and space.

To process a batch that deletes b occurrences: (i) obtain the b occurrence positions from the inverted lists of the deleted houses; (ii) determine the set of surviving agents whose current head is deleted by comparing the deleted positions against the stored head pointers; (iii) mark the b leaves dead; (iv) generate the $O(b \log n)$ ancestor node identifiers of the marked leaves, each tagged with its depth; (v) semisort the full collection of tagged identifiers once, deduplicating repeated nodes; (vi) recompute the affected nodes bottom-up, one depth level at a time, using

2026 S. T. Yau High School Science Award 仅用于2026丘成桐中学科学奖公示

the precomputed per-depth groups; (vii) for each of the h surviving agents whose head was deleted, descend its tree once to the new leftmost live leaf and update its head pointer. Steps (i)–(iv) and (vii) cost $O((b+h)\log n)$ work with $O(\log n)$ span. The single global semisort in step (v) costs $O(b \log n)$ expected work and $O(\log n)$ span, and because grouping is done once up front, step (vi) performs only $O(1)$-span synchronization per depth level over ready-made groups, for $O(\log n)$ span in total; sorting separately at every level would instead incur $O(\log n)$ span per level. Every occurrence is deleted at most once over the whole execution, giving the totals in (1). □

5.2 Dynamic forest operations

We maintain F in a batch-dynamic rake-compress (RC) forest [2, 4, 15]. These structures maintain a forest under batches of k edge insertions or deletions in $O(k \log n)$ expected work and $O(\log^2 n)$ span w.h.p., and support batch queries over the contracted hierarchy. The arbitrary-degree RC representation of [15] supplies this dynamic-forest interface directly on the original forest; its internal representation introduces auxiliary vertices, which carry an owner label where needed, are excluded from all root marks below, and contribute a factor absorbed by the stated logarithmic bounds.

Lemma 5.2 (Root maintenance and queries)**.** *The directed root of each component can be maintained so that a batch of k queries $\rho(x_1), \ldots, \rho(x_k)$ costs $O(k \log n)$ expected work and $O(\log^2 n)$ span w.h.p., and each batch of edge updates maintains the root information within the same bounds as the updates themselves.*

Proof. For each original live agent v, maintain

$$\text{rootFlag}(v) = 1 \iff v \text{ has no outgoing stored edge in } F,$$

and set rootFlag $= 0$ permanently on every auxiliary vertex of the RC representation. By invariant (I1), every component contains exactly one original vertex with rootFlag $= 1$, namely its directed root; this uniqueness is what makes the query well-defined. Augment every RC component with two aggregate values: the count of rootFlag-one original vertices in it, and the identity of that vertex whenever the count is one. Aggregates are recomputed along the $O(\log n)$-height contraction hierarchy whenever flags or edges change, within the update bounds of the RC forest.

Inserting a stored edge with tail v changes rootFlag(v) from 1 to 0; removing a stored edge whose tail v survives changes it from 0 to 1. These are $O(1)$ local metadata changes per forest edge update and are processed as part of the same RC update batch, so root information is maintained within the update bounds. A query $\rho(x)$ reads the aggregate identity at the component of x in $O(\log n)$ expected work; k independent queries run in parallel with $O(\log^2 n)$ span w.h.p. □

Lemma 5.3 (Disjoint path reporting)**.** *Given k pairs (x_i, r_i) such that the tree paths from x_i to r_i are pairwise vertex-disjoint and contain c original vertices in total, all path vertices can be enumerated in $O((k+c)\log n)$ expected work and $O(\log^2 n)$ span w.h.p.*

This enumeration lemma is derived in this paper; we use [2, 15] for the RC path-decomposition and arbitrary-degree machinery, whose cited path-query results are aggregate queries and do not by themselves enumerate path vertices.

2026 S.T. Yau High School Science Award
仅用于2026丘成桐中学科学奖论文公示

Proof. In an RC forest, the path between two vertices of a component decomposes into $O(\log n)$ canonical clusters of the contraction hierarchy, computable from the two root-to-vertex cluster sequences in $O(\log n)$ expected work per pair [2, 15]. Augment every binary cluster with pointers to exactly those child clusters that contribute to its cluster path, maintained during contraction at no asymptotic cost. To enumerate the path of a pair, expand the path-bearing pointers of its $O(\log n)$ canonical clusters recursively; this traversal visits only clusters whose cluster paths lie on the semantic path, ignoring raked-off subtrees, so a canonical cluster spanning ℓ path vertices is expanded with $O(\ell)$ work plus the $O(\log n)$-height overhead of the arbitrary-degree representation, and auxiliary representation vertices are translated to their owner labels and suppressed from the output. The expansions of distinct clusters and distinct pairs are independent and run in parallel with $O(\log^2 n)$ span w.h.p. Because the k semantic paths are pairwise vertex-disjoint, no original vertex lies on two reported paths, so no vertex is enumerated twice and no deduplication is needed.

For the work bound, we count the representation nodes visited. Every original vertex is represented by $O(\log n)$ nodes along the arbitrary-degree contraction hierarchy, and the path-bearing pointer expansion visits a representation node only if some original vertex of the semantic path being expanded lies in that node's cluster path. Hence expanding the $O(\log n)$ canonical clusters of a semantic path containing ℓ original vertices visits $O(\ell \log n)$ representation nodes. Summing over the k disjoint paths, decomposition costs $O(k \log n)$ expected work and expansion visits $O\big(\sum_i \ell_i \log n\big) = O(c \log n)$ nodes, for $O((k + c) \log n)$ expected work in total. □

5.3 Quotient cycle detection

Given the a_t edges $a \to q(a)$, the cycles of the functional graph Q are found by randomized list contraction in $O(a_t)$ expected work and $O(\log n)$ span w.h.p. [17, 8], and the cycle edges are extracted by a parallel filter.

5.4 The work-span theorem

Theorem 5.4. *In the Common-CRCW work-span model of Section 2, after $O(m)$-work preprocessing, Algorithm 1 computes the TTC allocation using $O(m \log n)$ expected work, $O(R \log^2 n)$ span w.h.p., and $O(m)$ space.*

Proof. Consider iteration t. Line 3 performs a_t head lookups ($O(a_t \log n)$ work by Lemma 5.1's structure) and a_t root queries ($O(a_t \log n)$ expected work, $O(\log^2 n)$ span, Lemma 5.2). Line 4 costs $O(a_t)$ expected work and $O(\log n)$ span (Section 5.3). Line 5 reports pairwise vertex-disjoint paths (Corollary 4.3) with at most a_t pairs and c_t total vertices, costing $O((a_t + c_t) \log n)$ expected work and $O(\log^2 n)$ span (Lemma 5.3). Lines 7 and 11 are linear in c_t and a_t. Line 8 costs $O((b_t + h_t) \log n)$ work and $O(\log n)$ span (Lemma 5.1). Lines 9 and 10 perform u_t edge updates in $O(u_t \log n)$ expected work and $O(\log^2 n)$ span w.h.p., maintaining root marks (Lemma 5.2).

Summing over rounds and applying (1), the total work is

$$O\Big(\sum_t (a_t + b_t + c_t + h_t + u_t) \log n\Big) = O(m \log n)$$

in expectation, and each of the R iterations has span $O(\log^2 n)$ w.h.p. Space is $O(m)$ for the lists and inverted index and $O(n)$ for the RC forest.

2026 S. Tao High School Science Award
仅用于2026年成尚中学科学奖公示

method	work or messages	parallel time or rounds
Abraham et al. [1]	$O(m)$ time	sequential
Zheng–Garg [27]	$O(n^2)$ messages	$O(n)$ distributed rounds
Garg LLP [11]	$O(n^2 \log n)$ exp. work	$O(n \log^2 n)$ exp. time
this paper	$O(m \log n)$ exp. work	$O(R \log^2 n)$ span w.h.p.

Table 1: Exact TTC algorithms. The Garg bound uses its dense shared-memory representation; R is the number of simultaneous TTC rounds.

For the high-probability claims, we union-bound over all randomized batch primitive calls: the quotient cycle decompositions, the semisorting steps of Lemma 5.1, and the RC-forest update and query batches. Each round issues a constant number of such calls, so there are $O(R) = O(n)$ calls of each type over the execution. Choosing the per-call failure probability to be $n^{-(c+3)}$, a union bound gives overall failure probability $O(n^{-c})$ for all calls simultaneously, and the expected-work bounds compose by linearity of expectation. □

Table 1 compares the bounds with prior exact algorithms. For bounded lists, $m \le nd$, so the work bound is $O(nd \log n)$, within a logarithmic factor of the $O(m)$ sequential algorithm; we do not claim asymptotic work-optimality. The comparison with [11] combines a representation difference (sparse relevant lists versus a dense matrix) with the algorithmic difference; on inputs with $m = \Theta(n^2)$ the two work bounds coincide up to constants.

Sequential realization. The invariant of Section 4.1 can also be maintained by a sequential link-cut tree [24] in $O(m \log n)$ total time. This realization is used in Section 7 to measure the structural effect of the forest separately from parallel execution; it is asymptotically slower than the $O(m)$ stack-based sequential algorithm and is not proposed as a practical solver.

6 Limits and parameter dependence

Theorem 5.4 contains a logarithmic factor and a span parameter R. This section records what is known about both.

6.1 The logarithmic factor

The sequential stack-based algorithm runs in $O(n+m)$ time with no data structure beyond arrays [1]. In Theorem 5.4, the $O(\log n)$ factor arises only from the dynamic-forest and segment-tree operations; the counting argument itself is linear. A natural question is whether the factor can be removed by path compression, since root queries resemble union-find.

TTC deletions have a special structure that makes cached shortcuts easy to validate. Departures are ancestor-closed: when a cycle is removed, an agent loses its parent only together with all of its ancestors, because the removed vertices above it formed the cycle.

Lemma 6.1 (Shortcut validity)**.** *Suppose that, after a root query, each traversed vertex caches a shortcut to the root found. At any later time, a cached shortcut $v \to s$ is usable (i.e., s is an ancestor of v or v itself is s) if and only if s is still present.*

Proof. If s is present, then no deletion has separated v from s: deletions remove ancestor-closed sets, so any deletion on the v–s path would have removed s. If s is absent, the shortcut is unusable. □

2026 S.-T. Yau High School Science Award
仅用于2026年成利中学奖论文公示

Compression is therefore correct, but it does not help in the worst case. We prove this for the following class of algorithms.

Definition 6.2. A *pointer-walking root maintainer* answers each root query by following parent pointers and previously cached shortcuts from the queried vertex until it reaches the root, and may, after each query, cache arbitrary shortcuts from traversed vertices to the root just found. It maintains no other auxiliary structure across operations.

Proposition 6.3 (Compression does not amortize)**.** *There is a family of markets with $d \le 3$ and $m = \Theta(k)$ on which every pointer-walking root maintainer performs $\frac{3}{2}k^2 + O(k)$ pointer traversals over the execution of Algorithm 1.*

Proof. The market (Figure 3) consists of a chain $x_1, \dots, x_k$ with lists $L_{x_1} = (h_{x_1})$ and $L_{x_i} = (h_{x_{i-1}}, h_{x_i})$; a bristle $y_1, \dots, y_k$ with $L_{y_1} = (h_{x_k}, h_{y_1})$ and $L_{y_j} = (h_{y_{j-1}}, h_{y_j})$; and probers $p_1, \dots, p_k$ with $L_{p_t} = (h_{x_t}, h_{y_k}, h_{p_t})$. Then $m \le 7k$ and $d = 3$.

The execution proceeds as follows. In round 1, x_1 self-cycles and departs. For $1 \le t \le k-1$, the departure of x_t at the end of round t activates both x_{t+1} and p_t in round $t+1$: x_{t+1} self-cycles and departs, while p_t recomputes its choice to y_k and issues the root query $\rho(y_k)$. At the start of round $t+1$ the component of y_k is the bristle together with the surviving chain $x_{t+1}, \dots, x_k$, rooted at x_{t+1}, so the walk from y_k traverses at least the k bristle vertices and the $k-t$ surviving chain vertices if no usable shortcut exists.

We claim no usable shortcut ever exists at query time. All shortcuts cached by the query of round $t+1$ point to the root found, x_{t+1}, which departs in that same round. By Lemma 6.1, every such shortcut is unusable in all subsequent rounds. Since the queries of distinct probers occur in distinct rounds, each query pays full depth:

$$\sum_{t=1}^{k-1} \big(k + (k-t)\big) = \tfrac{3}{2}k^2 + O(k). \qquad \square$$

The restriction to pointer-walking maintainers is necessary for the statement as proved; the general question is open. Define Churn-Root: maintain a rooted forest under edge insertions at roots and ancestor-closed top deletions, answering root queries. Known results bracket the problem. With insertions only, it is union-find, with an $\Omega(\alpha(n))$ amortized cell-probe lower bound matched by compression [10]. With deletions only on a static forest, constant amortized time is achievable by decremental techniques in the style of [3]. With insertions and unrestricted deletions, root queries require $\Omega(\log n)$ amortized time in the cell-probe model even on paths [20]; the hard sequences there reuse the same vertices across epochs, which ancestor-closed top deletions do not permit, so the bound does not transfer directly. Any upper bound $\beta(n)$ for Churn-Root in a batch-parallel form would replace the $\log n$ factor of Theorem 5.4 by $\beta(n)$.

6.2 The round parameter

The span of Theorem 5.4 is proportional to R. We record its worst case, its behavior on the hardness instances, an exact identity, and measurements on random markets.

Proposition 6.4. *$R \le n$ for every market, and the chain market with lists $L_{x_1} = (h_{x_1})$, $L_{x_i} = (h_{x_{i-1}}, h_{x_i})$ satisfies $R = n$ with $d = 2$.*

2026 S.T. Yau High School Science Award
仅用于2026丘成桐中学科学奖公示

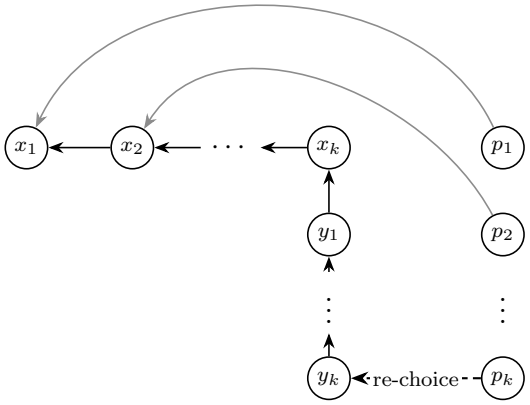


Figure 3: The broom. The churning chain (top) replaces its root every round, the static bristle hangs beneath x_k, and each prober spends one list entry to force one full-depth root query whose cached shortcuts are invalidated within the same round. Gray arcs are the probers' stored first choices.

Proof. Every round removes at least one agent, since every functional graph has a cycle. In the chain market, in round t the only cycle is the self-cycle of x_t: every agent x_i with $i > t$ still points to the present agent x_{i-1}. □

The worst case for R thus occurs at $d = 2$, inside the NC regime of Theorem 3.1, where the algorithm of that theorem bypasses the rounds entirely. The unsplit hardness construction provides instances with many rounds; the split ($d = 3$) construction exhibits the same behavior in the experiments below.

Proposition 6.5. *Let $M(C)$ be the unsplit market produced by the reduction of Section 3 from a circuit C of depth D. Then $R(M(C)) \geq D$.*

Proof. We show by induction on gate depth that no agent of a cell at depth j departs before round j. For $j = 1$ this is immediate. For the step, a cell at depth $j + 1$ has an input produced by a cell at depth j; by induction no agent of that cell departs before round j, so the input wire is not resolved before the end of round j, and by Lemma 3.9 no agent of the depth-$(j + 1)$ cell lies on a cycle before round $j + 1$. Hence $R(M(C)) \geq D$. □

The coupling of Lemma 3.11 preserves the relative order of asynchronous departures, which does not by itself determine simultaneous-round timing, so we make no corresponding claim for the split market $M_3(C)$; the P-completeness result does not depend on this proposition.

In simulations of the reduction on chain circuits up to depth 320, split and unsplit, the measured value is $R = D + 2$ at every depth, with at most three agents departing per round from round three onward; this is a measurement, and we claim no theoretical upper bound on $R(M_3(C))$ in terms of D. The same simulations verify $\sum_t |A_t| \leq m$ with equality-order behavior: the reduction instances are deep with $O(1)$ average round width.

Lemma 6.6 (Round-width identity)**.** *In every execution, $\sum_{t=1}^{R}$(departures in round t) $= n$; equivalently, the average number of departures per round is n/R, and by Lemma 4.4 the average of $|A_t|$ over rounds is at most m/R.*

This is an identity about averages; it does not bound the width of individual rounds. A family in which the width is uniform over rounds, and which we therefore use for the model of Section 8,

2026 S. T. Yau High School Science Award
仅用于2026丘成桐中学科学奖论文公示

n	10^3	$3 \cdot 10^3$	10^4	$3 \cdot 10^4$	10^5
mean R, $d = 3$	56	114	212	338	666
mean R, $d = 4$	62	104	190	353	

Table 2: Rounds on uniform random markets, five seeds per entry. Fitted exponents: 0.525 at $d = 3$ and 0.512 at $d = 4$.

is the interleaved-chain family: w disjoint chains of length n/w give exactly w departures in every round and $R = n/w$, for any $1 \le w \le n$.

Random markets. We measured R on the following generator: each agent draws $d-1$ distinct proper choices uniformly at random without replacement from the other $n-1$ houses, ordered as drawn, with its own house appended last. For each (n, d) we ran five independent seeds and report the mean; at $n = 5 \cdot 10^5$, $d = 3$, eight seeds gave $R \in [1305, 1728]$ with mean 1549. Table 2 shows means for $d = 3$ and $d = 4$. A least-squares fit of $\log \overline{R}$ against $\log n$ over the tabulated range gives slopes 0.525 ($d = 3$) and 0.512 ($d = 4$); these are empirical fits, not asymptotic claims. The first-round cyclic-vertex count is empirically consistent with the $\Theta(\sqrt{n})$ scale, and with the leading $\sqrt{\pi n/2}$ behavior of the classical random-mapping model; the generator excludes self-loops, so the initial top-choice graph is not literally a uniform random mapping, and the comparison is one of scale rather than distribution. Neither observation implies the growth rate of R in later rounds. For two-sided markets with full uniform preference lists, Che and Tercieux prove sublinear completion time with square-root-scale dynamics [6]; whether an analogous theorem holds for bounded-list endowment markets is open.

7 Experimental evaluation

7.1 Setup and reproducibility

The artifact contains five exact solvers over a common market format: the $O(m)$ stack-based sequential algorithm (used as the oracle), a sparse per-round rebuilding solver, a per-round pointer-jumping solver, the sequential link-cut realization of the persistent-choice invariant, and the parallel rake-compress (RC) implementation of Sections 4–5. Keeping the invariant identical across the last two solvers separates the effect of the structure from the effect of the backend. All wall-clock measurements were taken on an Apple M4 (10 cores); reported RC times include structure construction and are medians of three runs. All generators are deterministically seeded; the uniform benchmark instance uses seed 17. Code, generators, and validator scripts are in the repository referenced on the title page.

7.2 Correctness validation

All applicable solvers agree on 16,282 differential tests, comprising all 10,129 markets with $d \le 3$ and $n \le 4$, 5,450 random profiles across sizes and list lengths, the chain and cascade adversarial families, randomized RC schedules, and high-degree stress instances; the prototype suite contributes 15,777 of these. The reduction validator checks the gadget semantics of Section 3 on all 66,904 exhaustive input cases for circuits of up to five gates and on 10,500 random circuits of up to thirty gates, before and after selector splitting. The splitting transformation is additionally checked on all 65,536 four-agent markets, 100,000 random markets, and 41,650 ordinal instances.

2026 S. T. Yau High School Science Award
仅用于2026丘成桐中学科学奖公示

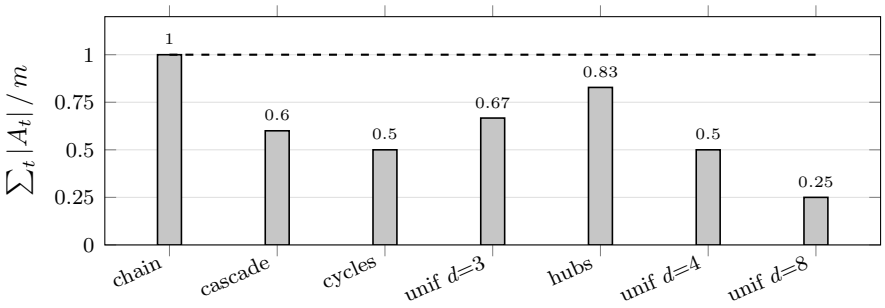


Figure 4: Frontier volume as a fraction of the charging budget m, from the artifact counters at the sizes of Table 3. The dashed line is the budget of Lemma 4.4; the chain saturates it, and no scenario exceeds it.

market	n	d	R	rebuild (ms)	quotient (ms)	ratio
length-two chain	20,000	2	20,000	886.0	1.63	543.3×
adaptive cascade	20,001	3	10,001	327.6	1.92	170.6×
uniform random	500,000	3	1,546	5,206.9	284.2	18.3×
hub random	500,000	3	66	173.4	243.4	0.71×
initial 8-cycles	500,000	2	1	1.94	5.53	0.35×

Table 3: Wall-clock times of the sparse rebuilding solver and the sequential link-cut quotient solver. Both are single-threaded, so the ratio reflects the change in structural work rather than multicore execution.

After every differential test, the counters m, R, $\sum_t |S_t|$, $\sum_t |A_t|$, occurrence deletions, quotient vertices, path endpoints, reported vertices, and forest insertions and removals are checked against the bounds of Section 4. These tests validate the implementation and the finite gadget instances; they are not part of the proofs.

7.3 Structural counters

Figure 4 shows the measured ratio $\sum_t |A_t|/m$ for the benchmark families. All instances satisfy the bound of Lemma 4.4; the chain family attains it with equality, as predicted by the analysis of Proposition 6.4. On the chain with $n = 20{,}000$, the rebuilding solver performs 200,010,000 live-vertex scans while the forest solver performs 19,999 head advances, matching the counts $\sum_t |S_t| = \Theta(n^2)$ and $\sum_t |A_t| = O(m)$ exactly.

7.4 Structural comparison of sequential solvers

Table 3 compares the rebuilding solver with the sequential link-cut quotient solver. Both are single-threaded, so the ratio reflects the change in structural work. The forest reduces time by large factors on families with many rounds and small active sets (chain, cascade, uniform random) and increases it on families with few rounds and large active sets (hub, initial cycles), where rebuilding performs a single contiguous scan and the quotient solver pays per-operation data-structure overhead on most of the market. The hub instance, with $R = 66$, lies near the boundary between the two regimes; on such instances the faster solver depends on per-operation constants and can differ across machines. Section 8 gives a cost model for this boundary.

7.5 Parallel backend

The RC implementation attains a self-speedup of 2.04× at eight workers on the hub family and 1.51× on a 105,000-agent preference-derived instance. Its absolute times do not beat the optimized sequential baselines: on the uniform benchmark, eight-worker RC takes 2,532 ms against 11.0 ms for the stack-based algorithm and 44.2 ms for the link-cut quotient solver. The

2026 S. T. Yau High School Science Award
仅用于2026丘成桐中学科学奖公示

constant-factor overhead of the general-purpose batch-dynamic structure dominates at these sizes, and the implementation does not establish a wall-clock improvement over sequential computation. The structural measurements above and the model below identify the instance regime in which a lower-constant backend could do so.

7.6 Limitations

The evaluation covers markets up to $n = 5 \cdot 10^5$ on one machine; the wall-clock conclusions are specific to this hardware and these implementations. The random-market statistics use five to eight seeds per configuration. The RC backend was not tuned per family, and no NUMA or multi-socket effects were measured.

8 Discussion: structural work versus parallel performance

The results of Section 7 show a consistent pattern: the forest reduces work precisely on instances with many rounds, and on such instances the active sets are small. This is a consequence of the identities of Section 6.2: departures sum to n, so an execution with many rounds has small average width, and the average parallelizable work per round is at most m/R. Conversely, executions with large average width have few rounds, and with few rounds the rebuilding cost $\sum_t |S_t|$ is already close to linear. The work saved by the forest and the work available to parallelize per round are governed by the same quantities.

A two-parameter cost model quantifies the boundary on the interleaved-chain family, where the width is uniform. Let c_r be the rebuilding cost per scanned vertex and c_F the forest cost per active-set operation, including the root query, so that c_F carries the logarithmic factor of Theorem 5.4. With width w and $R = n/w$ rounds, the operation totals in this model are

$$\text{rebuild}(w) = c_r\Big(nR - \tfrac{wR(R-1)}{2}\Big), \qquad \text{forest}(w) = c_F\,(2n - w),$$

which are exact within the model that assigns one calibrated aggregate cost c_r or c_F per scanned vertex or per activation; they are not counts of low-level machine or tree operations. For $R \le 2$ the two operation counts coincide at $2n - w$, so at large widths the comparison reduces to the constants c_r versus c_F, which is the hub and initial-cycles behavior of Table 3. When the per-operation costs differ substantially, the model predicts a crossover width w^*, and $w^* = \Theta(n/L)$ when the forest pays an additional factor L per operation.

We tested the prediction by charging each forest operation a seventeen-step random pointer chase (an elision-guarded synthetic cost that a compiler cannot remove), at $n = 131{,}072$ on the benchmark machine. The measured constants are $c_r = 5.99 \times 10^{-10}\,\text{s}$ and $c_F = 3.72 \times 10^{-8}\,\text{s}$; the model predicts $w^* = 532$, and the measured curves cross at $w \approx 551$, within the sampling bracket $[532, 648]$ (Figure 5). With the synthetic charge removed, $c_F < c_r$ on this machine and no crossover occurs: the forest is faster at every width, and the ratio approaches the constant c_r/c_F at $R = 2$, as the model predicts.

A production implementation could monitor $|A_t|$ and $|S_t|$ and an online estimate of c_F, and switch between the forest and a flat scan per round. We have not implemented this adaptive solver; we state it as future work, together with an evaluation on families that interpolate between the interleaved chains and the broom of Section 6.1.

2026 S.T. Yau High School Science Award
仅用于2026丘成桐中学科学奖论文公示

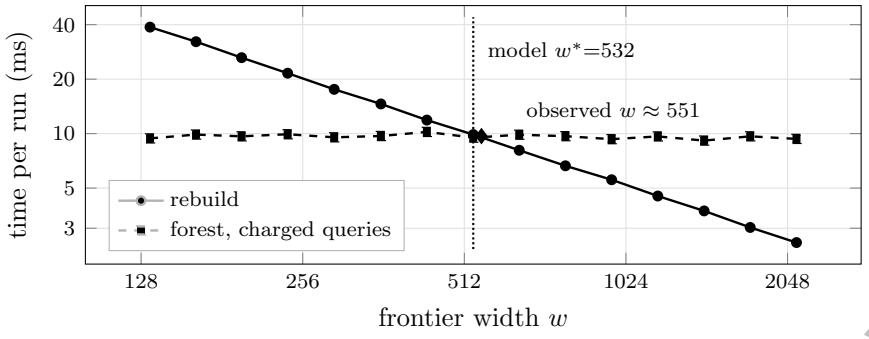


Figure 5: Backend crossover on the interleaved-chain family, $n = 131{,}072$, benchmark machine. The exact model, fed only the two measured per-operation constants, predicts the crossing at $w^* = 532$; the measured curves cross at $w \approx 551$ (diamond).

9 Conclusion

We studied the parallel complexity of Top Trading Cycles as a function of the relevant list length d and the round count R. For $d \leq 2$ the outcome is determined by a static functional graph and is computable in deterministic NC; for $d \geq 3$ the outcome problem is P-complete under logspace reductions, via an encoding of monotone circuit evaluation into market dynamics. For the algorithmic side, the persistent choice forest retains all unchanged choice edges between rounds, reduces each round to a functional quotient on the changed roots, and admits a charging argument bounding total root activations by m; with batch-dynamic trees this gives an exact parallel algorithm with $O(m \log n)$ expected work, $O(R \log^2 n)$ span w.h.p., and $O(m)$ space.

Zheng and Garg asked about the exact complexity of the problem: "It is interesting to know whether this problem is CC-complete. Or can we show that it is P-complete?" [27]. Theorem 3.13 answers the second alternative, at list length three, and Corollary 3.14 places the problem outside CC unless P = CC.

Three questions remain open: the complexity of Churn-Root, which determines whether the logarithmic factor in the work bound is necessary; the growth of R on random bounded-list markets, for which we only have measurements; and the design and evaluation of an adaptive backend around the crossover identified in Section 8.

References

[1] D. J. Abraham, K. Cechlárová, D. F. Manlove, and K. Mehlhorn. Pareto optimality in house allocation problems. In *ISAAC 2004*, LNCS 3341, pp. 3–15. Springer, 2004.

[2] U. A. Acar, D. Anderson, G. E. Blelloch, L. Dhulipala, and S. Westrick. Parallel batch-dynamic trees via change propagation. In *ESA 2020*, LIPIcs 173, pp. 2:1–2:23, 2020.

[3] S. Alstrup, J. P. Secher, and M. Spork. Optimal on-line decremental connectivity in trees. *Information Processing Letters*, 64(4):161–164, 1997.

[4] D. Anderson and G. E. Blelloch. Deterministic and low-span work-efficient parallel batch-dynamic trees. In *SPAA 2024*, pp. 247–258. ACM, 2024.

[5] R. J. Anderson and E. W. Mayr. A P-complete problem and approximations to it. Technical report, Stanford University, 1984.

[6] Y.-K. Che and O. Tercieux. Top trading cycles in large markets: the asymptotic irrelevance of priorities. arXiv:2607.10819, 2026.

2026 S. T. Yau High School Science Award
仅用于2026丘成桐中学科学奖公示

[7] J. H.-Y. Chiang, I. Damgård, C. Orlandi, M. Pancholi, and M. Simkin. Securely computing one-sided matching markets. In *FC 2025*, LNCS 15751, pp. 126–143. Springer, 2026.

[8] R. Cole and U. Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986.

[9] S. A. Cook, Y. Filmus, and D. T. M. Lê. The complexity of the comparator circuit value problem. *ACM Transactions on Computation Theory*, 6(4):15:1–15:44, 2014.

[10] M. L. Fredman and M. E. Saks. The cell probe complexity of dynamic data structures. In *STOC 1989*, pp. 345–354. ACM, 1989.

[11] V. K. Garg. A lattice linear predicate parallel algorithm for the housing market problem. In *SSS 2021*, LNCS 13046, pp. 108–122. Springer, 2021.

[12] R. Garg and A. Psomas. Fairly allocating goods in parallel. In *AAMAS 2025*, 2025.

[13] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press, 1995.

[14] Y. Huang, C. Wang, J. Shi, and J. Shun. Efficient algorithms for parallel bi-core decomposition. In *APOCS 2023*, pp. 17–32. SIAM, 2023.

[15] H. Ikram, A. C. Brady, D. Anderson, and G. E. Blelloch. Parallel batch queries on dynamic trees: algorithms and experiments. In *SPAA 2025*, pp. 525–539. ACM, 2025.

[16] E. W. Mayr and A. Subramanian. The complexity of circuit value and network stability. *Journal of Computer and System Sciences*, 44(2):302–323, 1992.

[17] G. L. Miller and J. H. Reif. Parallel tree contraction and its application. In *FOCS 1985*, pp. 478–489, 1985.

[18] S. Miyano. The lexicographically first maximal subgraph problems: P-completeness and NC algorithms. *Mathematical Systems Theory*, 22:47–73, 1989.

[19] T. Morrill and A. E. Roth. Top trading cycles. *Journal of Mathematical Economics*, 112:102984, 2024.

[20] M. Pătraşcu and E. D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 35(4):932–963, 2006.

[21] A. E. Roth. Incentive compatibility in a market with indivisible goods. *Economics Letters*, 9(2):127–132, 1982.

[22] A. E. Roth and A. Postlewaite. Weak versus strong domination in a market with indivisible goods. *Journal of Mathematical Economics*, 4(2):131–137, 1977.

[23] L. Shapley and H. Scarf. On cores and indivisibility. *Journal of Mathematical Economics*, 1(1):23–37, 1974.

[24] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.

[25] A. Subramanian. A new approach to stable matching problems. *SIAM Journal on Computing*, 23(4):671–700, 1994.

2026 S. T. Yau High School Science Award
仅用于2026丘成桐中学科学奖论文公示

[26] T. Tseng, L. Dhulipala, and G. E. Blelloch. Batch-parallel Euler tour trees. In *ALENEX 2019*, pp. 92–106. SIAM, 2019.

[27] X. Zheng and V. K. Garg. Parallel and distributed algorithms for the housing allocation problem. In *OPODIS 2019*, LIPIcs 153, pp. 23:1–23:16, 2020.

2026 S.-T. Yau High School Science Award
仅用于2026丘成桐中学科学奖论文公示

Acknowledgement

Vaibhav Shakya (V. S.) would like to thank his mentor, **Mr. Satnam Singh Chhabra**, for his guidance throughout this project. The idea for this research began in an unexpected place. While watching an episode of *House M.D.*, V. S. came across a scene involving a conflict over a parking spot, which made him wonder whether there was an optimal way to assign parking spaces when different people had different preferences. Following this question somewhat seriously exposed him to the Shapley–Scarf housing market model and its Top Trading Cycles (TTC) mechanism. The idea remained in the back of his mind until a later discussion of development economics, specifically the 2008 housing crash, in **Dr. Laxmi Sharma's** class brought the housing-allocation problem back to his attention. Looking at TTC again from an algorithmic perspective, V. S. noticed a source of inefficiency in round-based implementations: much of the surviving market may be revisited after every round even though only a relatively small number of agents have actually had their choices invalidated. He brought this observation to Mr. Chhabra, who encouraged him to investigate parallel implementations and, more importantly, whether a different data-structural formulation could preserve unchanged information rather than repeatedly reconstructing it. This suggestion ultimately shaped the persistent, change-driven perspective developed in the paper.

As the project progressed, Mr. Chhabra continued to push V. S. beyond simply producing a faster implementation. In particular, he encouraged V. S. to pursue what he referred to as the "holy grail" of the project: a formal complexity classification that could explain whether the remaining sequential dependence in TTC was merely a weakness of existing algorithms or something more fundamental. This led V. S. to study the problem from both an algorithms and complexity-theoretic perspective, eventually developing the persistent-choice forest and changed-root quotient formulation, analyzing its parallel work and span, and separately investigating the intrinsic difficulty of TTC outcome determination. V. S. would also like to thank Dr. Sharma for unknowingly providing the setting in which his earlier curiosity about allocation mechanisms resurfaced and developed into a concrete research direction.

The project also made limited use of **artificial intelligence tools** as auxiliary research, coding, and writing aids. A detailed account of when and how these tools were used, together with the relevant chat records, is provided in the accompanying disclosure document. The models used included GPT-5.6 Sol, GPT-5.6 Terra, Claude Opus 4.6, and Claude Opus 4.7. During research, AI was used only to surface leads, for example, names of parallel-algorithmic techniques, dynamic data structures, complexity-theoretic concepts, or literature that V. S. could then investigate directly. These suggestions were not accepted as results or verification. V. S. independently designed the algorithmic formulation, developed the data structures and reductions used in the paper, wrote and tested the implementation, ran the experiments, derived the complexity bounds, and manually verified the correctness arguments and cited sources. AI was additionally used for permitted auxiliary tasks such as code-debugging suggestions, LaTeX formatting, grammar and language polishing, and structural organization; every suggested code change or technical claim was inspected and tested by V. S. before use. The algorithm design, proofs, implementation decisions, experimental verification, complexity classification, and final academic expression were completed and checked by V. S. under Mr. Chhabra's guidance.

2026 S. T. Yau High School Science Award
仅用于2026丘成桐中学科学奖论文公示